

Fault-Tolerant Causal Delivery in Group Communication

Kenji Shima, Hiroaki Higaki, and Makoto Takizawa
Tokyo Denki University
Ishizaka, Hatoyama, Saitama 350-03
e-mail{simas, hig, taki}@takilab.k.dendai.ac.jp

Abstract

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 the 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 the 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 requires the processes in the group to receive reliably the messages in the causal order [2]. If the communication system provides the group of processes with the 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 systems 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 objects. Here, it is required to achieve the *intra-group communication* [15, 16, 17, 22] where the processes communicate with each other in the group.

The processes may suffer from kinds of faults in the distributed system. An approach towards making the system fault-tolerant is to replicate the processes. In this paper, in order to support the fault-tolerant group communication, each process is replicated into a col-

lection of multiple *replicas*, which is named a *cluster*. Our protocol supports the communication among the clusters in order to tolerate the 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 the *causal order* [3, 17]. Takizawa *et al.* [15, 16, 22] discuss 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. Ezhilhelvan *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 process stop-fault and message loss. However, the protocol can neither tolerate the Byzantine fault nor provide the causally ordered delivery of messages. Our protocol can tolerate the Byzantine faults.

A *logical group* is a collection of multiple processes p_1, \dots, p_n . A *group* is composed of multiple *clusters* where each cluster is a collection of replicas of each process in the logical group. The replicas may suffer from the Byzantine faults, i.e. they may send messages including not only incorrect data but also incorrect header, may not send messages, and may send messages to incorrect destinations. In order to realize the fault-tolerant transmission of a message m from p_i to p_j , each replica in a cluster of p_i sends a message m to multiple replicas in a cluster of p_j in the group. 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 order to detect faulty replicas, each replica receives messages from multiple replicas. In this paper, we would like to discuss how the clusters communicate with each other in the group and how the replicas support the processes with the reliable and causally 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 what faults occur in group communication. In section 5, we discuss the fault-tolerant ordered delivery. In section 6,

we discuss the inter-cluster communication. Lastly, we present the evaluation of the inter-cluster communication protocols in section 7.

2 Replication Schemes

We would like to consider how to replicate a process p_i into replicas p_{i1}, \dots, 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 ($j = 1, \dots, l_i$). Even if some replica of p_i is faulty, the computation of p_i can be continued without stopping. By comparing the messages received from the replicas, if a replica sends a message different from one which the majority of the replicas send, the replica can be considered to be faulty.

In the primary-backup approach (*passive replication*) [4], there is one *primary* replica p_{i1} and backup replicas p_{i2}, \dots, p_{il_i} . p_{i1} communicates with the primary replicas in other clusters and computes while no backup replica computes. p_{i1} saves its local state information ls_{i1}^k in its local stable storage at the k th 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 p_{ij} saves ls_{i1}^k into the stable storage. If p_{i1} is detected to be faulty, one backup 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 rolling back 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 faults, and the computation can be taken over by the other replicas immediately if some replica is faulty. Moreover, the replicated processes could tolerate the Byzantine faults if the majority of the replicas are operational. 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, \dots, p_n ($n \geq 2$) communicating with each other by using the communication system. A collection of p_1, \dots, p_n is referred to as *group G*, written as $G = \langle p_1, \dots, p_n \rangle$.

3.1 Causally ordered delivery

Each process p_i in the group G is modeled as a finite state machine [19]. A state of p_i is transited when an event occurs. There 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 , $e_1 \rightarrow e_3 \rightarrow e_2$. \square

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 $s_i(m_1) \rightarrow s_j(m_2)$. \square

m_1 and m_2 are referred to as *causally coincident* ($m_1 \parallel m_2$) iff 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 destinations in the group G . If m is received by 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 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 $m_1 \prec m_2$. \square

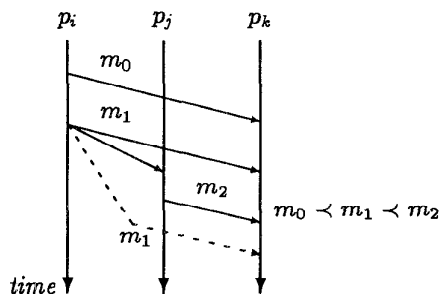


Figure 1: Causally ordered delivery

In Figure 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 . Hence, $m_0 \prec m_1 \prec m_2$ is satisfied. If p_k delivers m_0 , m_1 , and m_2 in this order, the messages are causally delivered. If p_k delivers m_1 after m_2 as denoted by the dotted line, the messages are not causally delivered.

As presented in papers [17, 21], each message m sent by p_i carries the confirmation field ack_j which denotes the sequence number sn of the message which p_i expects to receive next from p_j ($j = 1, \dots, n$). For example, if p_i sends a message m_i after receiving m_j

from p_j , $m_i.ack_j = m_j.sn + 1$. The messages can be causally ordered as follows [17]:

[Causally ordering (CO) rule] For every pair of messages m_1 and m_2 , m_1 *causally precedes* m_2 ($m_1 < m_2$) if

- (1) $m_1.sn < m_2.sn$ if m_1 and m_2 are sent by the same process,
- (2) $m_1.sn < m_2.ack_i$ otherwise (m_1 is sent by p_i). \square

3.2 Reliable delivery

Since p_i sends a message m to multiple processes in the group G , m is required to 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 . \square

Unless some process receives 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 reliably* delivers a message m if all the operational destination processes of m receive m . \square

In other words, the operational processes *agree* on m .

In order to achieve the reliable delivery of m , m can be retransmitted to a process p_i unless p_i receives m . If p_i is faulty, m can be sent to p_i after p_i recovers from the fault. Thus, it takes time to recover from the fault. This means that 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. \square

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

4 Faults in Group

In order to tolerate the process faults in the group, the processes are actively replicated to multiple replicas. That is, 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. For a logical group composed of n processes $\langle p_1, \dots, p_n \rangle$, a *group* G is composed of n clusters $c(p_1), \dots, c(p_n)$. Each cluster $c(p_i)$ is a collection of replicas p_{i1}, \dots, p_{il} of p_i ($i = 1, \dots, n$) [Figure 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.

Faults on the processes are assumed to be the Byzantine faults and to be independent in this paper. Due to the processor fault like memory error,

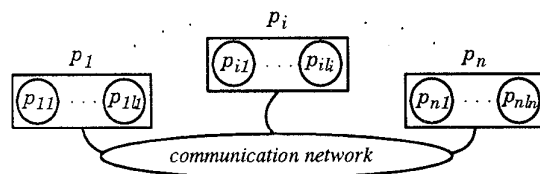


Figure 2: Group

the process on the faulty processor may behave incorrectly. That is, the process is viewed to suffer from the Byzantine fault. The faulty replica p_{ij} behaves as follows:

- (1) p_{ij} sends incorrect messages.
- (2) p_{ij} sends no message.
- (3) p_{ij} sends messages to incorrect destinations.

Each message m includes *data* and control information like sequence number sn and acceptance confirmation ack_1, \dots, ack_n . The faulty replica p_{ij} may send messages including incorrect data and incorrect control information. For example, the faulty p_{ik} sends a message m with data a to some p_{jh} while sending m with b ($\neq a$) to other replicas in $c(p_j)$. Even if p_{ik} sends a message m_1 after receiving m_2 from $c(p_j)$, p_{ik} includes the incorrect confirmation of " $m_1.ack_j > m_2.sn$ " in m . If a replica p_{fg} receives m_1 and m_2 , p_{fg} considers that p_{ik} sends m_1 before receiving m_2 . In addition, p_{ik} may not send messages to the destinations. For example, p_{ik} sends a message m to replicas in $c(p_j)$ but not to p_{jh} . p_{ik} may send a message m to incorrect destinations. For example, p_{ik} may send m to p_{jh} which is not the destination of m . A message m sent by a process p_i is sent by multiple replicas in $c(p_i)$. The messages sent by the replica to deliver m are referred to as *instances* of m .

In addition, we make the following assumptions on how each replica can manipulate messages:

- (C1) 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.
- (C2) 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 instances of the same message if $m_1.sn = m_2.sn$ and $m_1.src = m_2.src$ even if m_1 and m_2 are not the same. For every pair of messages m_1 and m_2 , $m_1.sn < m_2.sn$ and $m_1.src = m_2.src$ iff m_1 is sent before m_2 . A faulty replica may change a content of m but can change neither $m.sn$ nor $m.src$. 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_{ik} is faulty.

The communication system is assumed to satisfy

the following properties:

- (C3) The communication system is *reliable* and *synchronous* [9], i.e. messages are neither lost, contaminated, nor duplicated, and the propagation delay time (δ) is bounded.
- (C4) If a replica p_{ik} sends p_{jh} a message m_1 before m_2 , p_{jh} receives m_1 before m_2 . That is, the communication system supports the FIFO ordered delivery of messages for every pair of replicas.

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.

We make the following assumption on the number of faulty replicas in the cluster.

- (C5) At most f_i ($\leq l_i$) replicas are faulty at the same time in each cluster $c(p_i)$.

The faulty replica can be detected by comparing the message instances sent by the replicas in the cluster. A message m sent by the cluster $c(p_i) = \{p_{i1}, \dots, p_{il_i}\}$ has to be delivered to the clusters of the destination processes, say $c(p_j) = \{p_{j1}, \dots, p_{jl_j}\}$. Each replica p_{jh} receives message instances from multiple replicas in $c(p_i)$ while p_{jh} sends a message to multiple replicas in other cluster. Some message instances may be sent by faulty replicas. Hence, the replicas in $c(p_j)$ have to detect message instances sent by the operational replicas in $c(p_i)$. The replicas in $c(p_j)$ have to receive message instances from more than $2f_i$ replicas in $c(p_i)$ if at most f_i replicas are faulty in $c(p_i)$. If the replicas in $c(p_j)$ receive the same instances of a 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 message instances 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, \dots, p_n are required to be fault-tolerantly and causally delivered to the application processes even if process faults occur. 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. $c(p_i) = \{p_{i1}, \dots, p_{il_i}\}$ ($i = 1, \dots, n$) where each p_{ij} is a replica of p_i . In $c(p_i)$, at most f_i replicas are assumed to be faulty in $c(p_i)$. Each replica p_{ik} in $c(p_i)$ receives instances of a message m sent by the replicas in $c(p_j)$, and accepts the instances of m as the message m sent by p_j . It would be discussed how to accept m in the presence of the faulty replicas in the next section.

On sending a message m , p_i stores the following information to m : the sequence number $m.sn$, the acceptance confirmation $m.ack_1, \dots, m.ack_n$, and the data $m.data$. Each time p_i sends a message m , m includes the confirmation $m.ack_j$ which denotes a sequence number of message which p_i expects to accept next from each process p_j ($j = 1, \dots, n$). That is, if p_i sends a message m_1 just after accepting m_2 from p_j , $m_1.ack_j := m_2.sn + 1$. The replica p_{ik} sends an instance m_k of m to the replicas of the destination processes. In the instance m_k , $m_k.sn = m.sn$, $m_k.ack_j$

$= m.ack_j$ ($j = 1, \dots, n$), and $m_k.data = m.data$.

It is noted that the faulty replica can change the confirmation fields ack_1, \dots, ack_n in a message instance while it cannot change the fields denoting the sequence numbers and source replica. This means that the faulty replica may send the incorrect information on the message precedence to other replicas. For example, in Figure 3, 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 from another process p_l . 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, say $m_1 \parallel m_2$ while p_{i1} and p_{i2} send m_2 with $m_1 \prec m_2$.

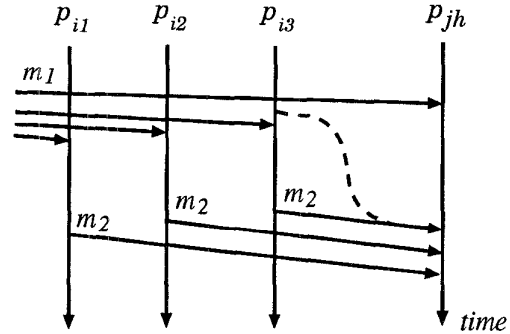


Figure 3: Causally ordered delivery

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 $m_1 \prec_{ik} m_2$) because $m_1.sn \leq m_2.ack_l$ by the CO rule.

[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. $m_1 \prec m_2$.

We would like to present a procedure for the replica perception. Each replica p_{ik} has an *instance receipt* queue IRQ_{jh} to store the message instances received from each replica p_{jh} ($h = 1, \dots, l_j$) in $c(p_j)$. The message instances in IRQ_{jh} are sorted in the sequence number. First, the top message instance m_h in each IRQ_{jh} is checked. Here, it is noted that $m_1.sn = \dots = m_{l_j}.sn$. If $m_h.sn$ is different from the others for some IRQ_{jh} , p_{ik} considers that p_{jh} sends no message to p_{ik} if IRQ_{jh} is not empty or more than f_j instance receipt queues have the top messages whose sequence numbers are the same.

Next, p_{ik} has to accept the message m by using the instances m_1, \dots, m_{l_j} sent by the replicas in $c(p_j)$. p_{ik} has one *message receipt* queue RQ_j to store the messages accepted from p_j ($j = 1, \dots, n$) [Figure 4]. m is constructed from the instances m_1, \dots, m_{l_j} as follows:

$$m.sn := m_h.sn \text{ if } |\{m_l \mid m_l.sn = m_h.sn\}| \geq f_j + 1;$$

$$m.data := m_h.data$$

$$\text{if } |\{m_l | m_l.data = m_h.data\}| \geq f_j + 1;$$

In addition, the acceptance confirmation has to be stored in ack_1, \dots, ack_n of m .

$$m.ack_l := m_h.ack_l$$

$$\text{if } |\{m_g | m_g.ack_l = m_h.ack_l\}| \geq f_j + 1;$$

$$(l = 1, \dots, n)$$

Then, m is enqueued into RQ_j . The top message m_h is removed from every IRQ_{jh} if $m_h.sn = m.sn$ ($h = 1, \dots, l_j$). Here, p_{ik} considers that p_{jh} is faulty if $m \neq m_h$. In RQ_j , the messages are sorted in the sequence number and each message includes the correct data and correct confirmation which p_{ik} perceives.

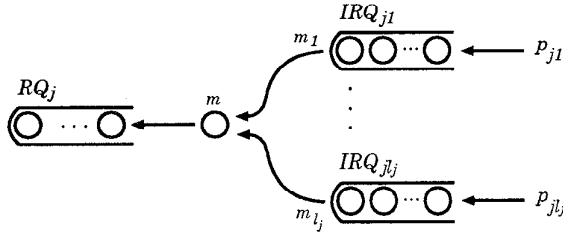


Figure 4: Receipt queues of p_{ik}

In Figure 3, 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 sending the instance of m_2 . p_{i3} notifies p_{jh} of " $m_1 || m_2$ " by m_2 while p_{jh} sends an instance of 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 notify p_{jh} of " $m_1 \prec m_2$ ".

Next, the messages stored in RQ_1, \dots, RQ_n of the p_{ik} have to be ordered in the causal order. The messages are ordered by the *CO* rule. p_{ik} has a *delivery* queue DQ . The messages are enqueued into DQ from RQ_1, \dots, RQ_n by the following procedure [Figure 5].

[**Delivery procedure**] while every RQ_h is not empty, p_{ik} executes the following three steps:

- (1) let m_h be the top message of RQ_h ($h = 1, \dots, n$);
- (2) select m_j such that $m_j \prec m_h$ for every m_h ($h = 1, \dots, n$) by the *CO* rule.
- (3) enqueue m_j to DQ and remove m_j from RQ_j ; \square

The messages in DQ are causally ordered.

The processes decide the causality among the messages based on the messages accepted by the replicas.

[**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 $m_1 \prec m_2$. \square

Each process p_i takes the top messages m_1, \dots, m_{i_l} from the delivery queues DQ_1, \dots, DQ_{l_i} of $p_{i1}, \dots, p_{i_{l_i}}$. p_i accepts m_h such that $|\{m_j | m_j = m_h\}| \geq f_i + 1$. p_i considers that p_{ik} is faulty if $m_k \neq m_h$.

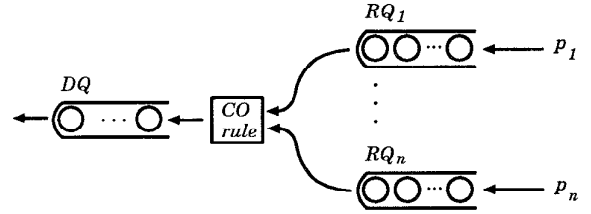


Figure 5: Delivery procedure

[**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 by using the perception rule if $m_1 \prec m_2$. \square

6 Inter-Cluster Communication

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 . The replicas in $c(p_i)$ send m to the replicas in $c(p_j)$ in G by some inter-cluster communication method to be presented here. p_j accepts m if at least $f_j + 1$ replicas in $c(p_j)$ accept m from $c(p_i)$ by the delivery procedure presented in the previous section. 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 replica 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}, \dots, p_{jl_j} in $c(p_j)$ for $k = 1, \dots, l_i$.
- (RSB) Each p_{ik} sends m to a subset $I_j(p_{ik}) \subseteq c(p_j)$ for $k = 1, \dots, l_i$.
- (SSB) Each replica p_{ik} in a subset $S(p_i) \subseteq c(p_i)$ sends m to all replicas in $c(p_j)$.
- (HSB) 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. If the broadcast network is used, 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) $I_j(p_{i1}) \cup \dots \cup I_j(p_{il_i}) = c(p_j)$, 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 $|I_j(p_{i1})| + \dots + |I_j(p_{il_i})|$. 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, $l_i - [(2f_i + 1)l_j \text{ modulo } l_i]$ replicas send m to $\lceil (2f_i + 1)l_j / l_i \rceil$ replicas and $(2f_i + 1)l_j \text{ modulo } l_i$ replicas send m to $\lfloor (2f_i + 1)l_j / l_i \rfloor$ replicas if $(2f_i + 1)l_j \text{ modulo } l_i \neq 0$. This is a *receiver selective broadcast (RSB)* distribution method.

In the third way, only a subset $S(p_i)$, not necessary 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 $|S(p_i)|$. $l_i \geq g_i \geq 2f_i + 1$. 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 *sender selective broadcast (SSB)* 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 second method and the replicas send m to not necessarily all the replicas in $c(p_j)$ like the third method. Only g_i ($\geq 2f_i + 1$) 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) $K_j(p_{i1}) \cup \dots \cup K_j(p_{il_i}) = c(p_j)$ where $K_j(p_{ik}) = \phi$ if p_{ik} does not send m ,
- (2) $|\{p_{ik} \mid p_{jh} \in K_j(p_{ik})\}| = 2f_i + 1$ for every p_{jh} , and
- (3) $g_i = |\{p_{ik} \mid K_j(p_{ik}) \neq \phi\}| \geq 2f_i + 1$.

Here, we assume that every replica in $T(p_i)$ sends m to e_i replicas in $c(p_j)$, i.e. $e_i = (|K_j(p_{i1})| + \dots + |K_j(p_{il_i})|) / g_i$. 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 *SSB* 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* one. This method is named a *hybrid selective broadcast (HSB)* distribution one.

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

replicas can send messages because each p_{jh} receives three messages. Here, four messages are transmitted.

Figure 7 shows the *HSB*. Each of four replicas in five replicas of $c(p_i)$ sends three or four messages. Each replica in $c(p_j)$ receives three messages, but does not receive from all the replicas in $c(p_j)$. Totally 15 messages are transmitted.

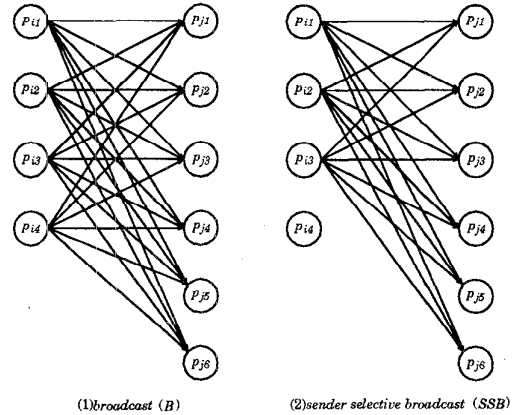


Figure 6: Inter-cluster communication

The replicas which receive messages from other clusters and send messages to other clusters are referred to as *input* and *output* replicas, respectively. In Figure 6(1), all the replicas are output ones. In the *MB* and *MSB* methods, g_i output replicas send messages. In Figure 6(2), p_{i1} , p_{i2} , and p_{i3} are output replica.

In the *RSB* and *HSB* methods, we have to decide to which replicas in $c(p_j)$ each replica in $c(p_i)$ sends m . In the *SSB* and *HSB* methods, it has to be decided which replicas in $c(p_i)$ are output ones. One way is to decide it statically when the group is established. If some output replica is faulty and the constraints of the methods are not satisfied, some non-output replica is selected to be an output one. In another way, the input replicas in $c(p_j)$ for each p_{ik} and the output replicas in $c(p_i)$ are dynamically decided each time the replicas in $c(p_i)$ send messages to $c(p_j)$. The faulty replica may send a message to incorrect destinations. In the static method, it is easier for each replica p_{ik} to detect the fault than the dynamic method because it is fixed which replicas p_{ik} receives. Hence, the static allocation of the destination replicas is adopted in our system. If p_{jh} detects p_{ik} to be faulty on receipt of m from p_{ik} , p_{ik} sends m with the digital signature to all the replicas in $c(p_j)$. On receipt of the message m from p_{jh} , p_{j1} forwards m with the signature to all the replicas in $c(p_j)$. By the exchanging messages, the operational replicas in $c(p_j)$ make an agreement on that p_{ik} is faulty. After that, p_j neither sends nor receives messages with p_{ik} .

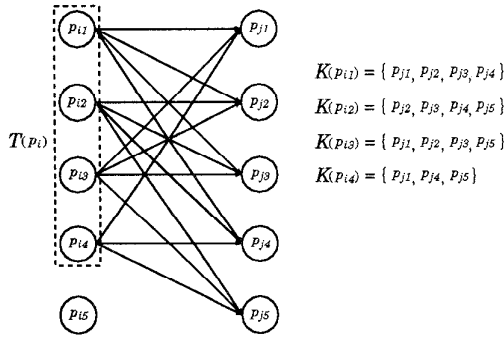


Figure 7: Inter-cluster communication (*HSB*)

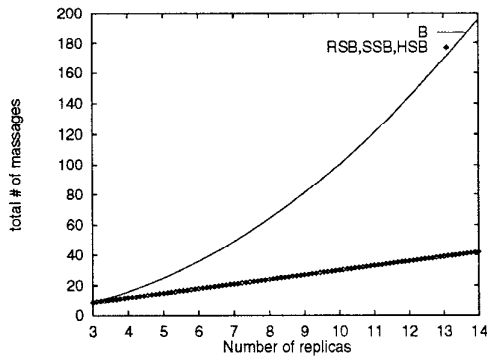


Figure 8: Total number of messages

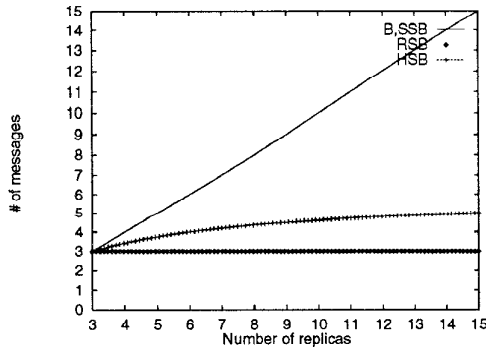


Figure 9: Number of messages sent by output replicas

7 Evaluation

Table 1 shows the number of messages for the distribution methods presented here, where $g_i = (l_i + 2f_i + 1)/2$. 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 replica p_{i_k} and input replica p_{j_h} , respectively.

Method	# of messages	#/output	#/input
B	$l_i \cdot l_j$	l_j	l_i
RSB	$(2f_i + 1) \cdot l_j$	$(2f_i + 1)l_j/l_i$	$2f_i + 1$
SSB	$(2f_i + 1) \cdot l_j$	l_j	$2f_i + 1$
HSB	$(2f_i + 1) \cdot l_j$	$(2f_i + 1)l_j/g_i$	$2f_i + 1$

Table 1: Number of messages in the one-to-one network

Method	# of messages
B	l_i
RSB	l_i
SSB	$2f_i + 1$
HSB	$2f_i + 1$

Table 2: Number of messages in the broadcast network

From Table 1, in the *RSB*, *SSB*, and *HSB* methods, each replica receives less number of messages than the *B* method. From Table 2, in the *SSB* and *HSB* methods, each replica sends less number of messages than the *B* and *RSB*. Figure 8 shows the total number of messages transmitted from $c(p_i)$ to $c(p_j)$ for l_i where $l_i = l_j$ and $f_i = f_j = 1$. From Figure 8, in the *RSB*, *SSB*, and *HSB* methods, the same number of messages are transmitted, but less number of messages are transmitted than the *B* method. Figure 9 shows “#/output”, i.e. the number of messages transmitted by each output replica in $c(p_i)$ for l_i . In the *RSB* method, each output replica sends the smallest number of messages. Therefore, the *RSB* method is the best one in the one-to-one network and the *RSB* method is the best one for broadcast network.

8 Concluding Remarks

This paper has discussed 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 finite state machine and is actively replicated to a set of multiple replicas, named a cluster. In this paper, we have presented 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, may send to incorrect destinations, and may include the incorrect ordering information on the messages. We have discussed how to treat the faults of the unreliable and incorrectly or-

dered delivery of messages. In addition, we have discussed protocols for inter-cluster communication and evaluated them in terms of the number 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, 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] Cooper, E. C., "Replicated Distributed Programs," *Proc. of the 10th ACM Symp. on Operating Systems Principles*, 1985, pp.63-78.
- [7] Ellis, C. A., Gibbs, S. J., and Rehn, G. L., "Groupware: Some Issues and Experiences," *Comm. of the ACM*, Vol.34, No.1, 1991, pp.39-58.
- [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] Fischer, J. M., Nancy, A. L., and Michael, S. P., "Impossibility of Distributed Consensus with One Faulty Process," *ACM TOCS*, Vol.32, No.2, 1985, pp.374-382.
- [10] Garcia-Molina, H. and Spauster, A., "Ordered and Reliable Multicast Communication," *ACM TOCS*, Vol.9, No.3, 1991, pp.242-271.
- [11] Higaki, H. and Soneoka, T., "Group-to-Group Communications for Fault-Tolerance in Distributed Systems," *IEICE Trans. on Information and Systems*, Vol.E76-D, No.11, 1993, pp.1348-1357.
- [12] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM*, Vol.21, No.7, 1978, pp.558-565.
- [13] 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.
- [14] Moser, L. E., Amir, Y., Melliar-Smith, P. M., and Agarwal, D. A., "Extended Virtual Synchrony," *Proc. of the 14th IEEE ICDCS*, 1994, pp.56-65.
- [15] Nakamura, A. and Takizawa, M., "Reliable Broadcast Protocol for Selectively Ordering PDUs," *Proc. of the 11th IEEE ICDCS*, 1991, pp.239-246.
- [16] Nakamura, A. and Takizawa, M., "Priority-Based Total and Semi-Total Ordering Broadcast Protocols," *Proc. of the 12th IEEE ICDCS*, 1992, pp.178-185.
- [17] Nakamura, A. and Takizawa, M., "Causally Ordering Broadcast Protocol," *Proc. of the 14th IEEE ICDCS*, 1994, pp.48-55.
- [18] Powell, D., Chereque, M., and Drackley, D., "Fault-Tolerance in Delta-4," *ACM Operating System Review*, Vol.25, No.2, 1991, pp.122-125.
- [19] Schneider, B. F., "Replication Management using the State-Machine Approach," *Distributed Computing Systems*, *ACM Press*, 1993, pp.169-197.
- [20] Schneider, B. F., "Byzantine Generals in Action: Implementing Fail-stop Processors," *ACM TOCS*, Vol.2, No.2, 1984, pp.145-154.
- [21] 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.
- [22] Takizawa, M., "Cluster Control Protocol for Highly Reliable Broadcast Communication," *Proc. of the IFIP Conf. on Distributed Processing*, 1987, pp.431-445.