Fault-Tolerant Intra-Group Communication *

7U−6

Kenji Shima and Makoto Takizawa †
Tokyo Denki University ‡
e-mail{sima,taki}@takilab.k.dendai.ac.jp

1 Introduction

In distributed applications like teleconferences, a group of multiple application processes have to be communicated. Multiple ***system*** processes have to support the application processes with the *atomic*, *ordered*, and ***non-loss*** delivery of messages. Even if the processes in the group fault, the group of application processes have to be communicated. In this paper, we assume that the underlying network is reliable.

In the ***intra-group*** communication, the processes in the group communicate with one another. We discuss the fault-tolerant intra-group communication. In this paper each process is realized by a group of multiple ***replicas*** of the process. The process is composed of ***input***, ***computation***, and ***output*** processing units. There are three kinds of replication strategies, i.e. *active*, *passive*, and *semi-active* replications [3] where the whole processes are replicated. In this paper, we would like to present a new strategy named ***unit*** replication where the three kinds of units are independently allocated to the replicas in order to reduce the total processing and communication overhead and to tolerate the Byzantine fault [1] of ***replicas***.

In section 2, we present a system model. In section 3, we present how to replicate the system processes. In section 4, we present fault treatment.

2 System Model

A communication system is composed of *application*, ***system***, and ***network*** layers [Figure 1]. The network layer provides the system layer with high-speed communication. There are two kinds of networks, i.e. ***one-to-one*** and ***broadcast*** ones. A ***logical*** group G is composed of n (≥ 2) ***system*** processes $p_1, \ldots, p_n$ ($G = \langle p_1, \ldots, p_n \rangle$). Each p_i supports the application process A_i with the ordering group communication. A ***physical*** group P_G of G is composed of replicas of the ***system*** processes, i.e. $P_G = \langle \{ p_{11}, \ldots, p_{1l_1} \}, \ldots, \{ p_{n1}, \ldots, p_{nl_n} \} \rangle$ where p_{ij} is a replica of p_i.

The replicas of p_i support A_i with the group communication in the presence of faults of the replicas. $\{p_{i1}, \ldots, p_{il_i}\}$ is a ***replica group*** of p_i. The replicas of p_i are located in different processors. We assume that the maximum number f_i of replicas of p_i which fault at the same time is fixed.

3 Unit Replication

Let us consider a group $G = \{ p_1, \ldots, p_n \}$ ($n \geq 2$). There are three kinds of processing units in p_i:

(1) to receive messages from other replica groups,
(2) to do the computation of p_i, and
(3) to send messages to other replica groups.

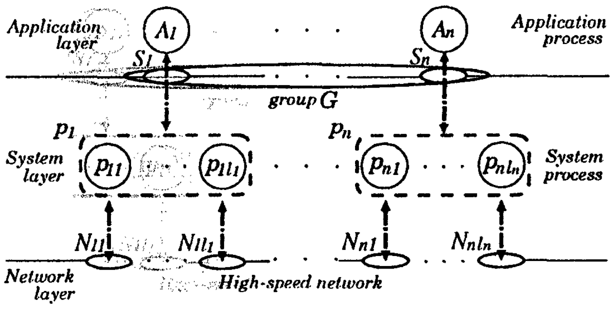

Figure 1: System model

The first, second, and third are *input*, *computation*, and *output* units, respectively. These units are independently allocated to the replicas in the replica group so that the communication and processing overhead can be minimized and the Byzantine fault of the replica is tolerated. This replication is named a ***unit*** replication. The replicas do not always execute all the three units. The replicas with the input, output, and computation units are ***input***, ***output***, and ***computation*** ones, respectively. The replicas with no unit are ***dormant*** replicas.

The input replicas of p_i receive messages from the output replicas of the replica group of another p_j:

(1) to detect the faulty output replicas of p_j, and
(2) to distribute the correct messages received to the computation replicas of p_i.

On receipt of the messages from p_j, the input replica adopts the ***majority*** rule, i.e. take the majority messages from p_j and detect the faulty output replicas of p_j which send messages different from the majority ones. Then, the input replicas forward the correct message to the computation replicas.

The computation replicas of p_i do the followings on receipt of the messages from the input replicas of p_i:

(1) to do the computation, and
(2) to send the messages to the output replicas.

The computation replica has to receive the messages from more than $2f_i$ input replicas. By using the majority rule, the computation replicas can detect the faulty input replicas.

The output replicas of p_i do the followings:

(1) to receive the messages from the computation replicas of p_i, and
(2) to send the message to the input replicas of the other replica groups.

The output replica decides what messages are correct among the messages collected from the computation ones by applying the majority rule to the messages received from the computation ones. Then, the output replica sends the message to the other replica group.

The input, computation, and output units are allocated to x_i ($\leq l_i$) input, y_i ($\leq l_i$) computation, and

*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

z_i ($\leq l_i$) output replicas, respectively. For each p_i, let I_i, C_i, and O_i denote sets $\{I_{i1}, \ldots, I_{ix_i}\}$, $\{C_{i1}, \ldots, C_{iy_i}\}$, and $\{O_{i1}, \ldots, O_{iz_i}\}$ of input, computation, and output replicas of p_i, respectively. Here, x_i, y_i, and z_i $\geq 2f_i + 1$ in order to adopt the majority rule to detect the Byzantine fault.

If $I_i \cap C_i = C_i \cap O_i = O_i \cap I_i = \phi$, the replica group is ***fully dispersed***. If $I_i = C_i = O_i$, the replica group is ***fully multiplexed***. If not fully dispersed, the replica group is ***multiplexed***. In the fully dispersed replica group, even if one replica faults, only one unit faults since the replica has the unit. If one computation replica faults, three units fault in the fully multiplexed one. The fully dispersed replica group is required to include more replicas than the fully multiplexed ones for given x_i, y_i, and z_i.

Since the non-computation replicas have to catch up with the computation replicas, the computation ones take the checkpoints where the local states are saved into the logs, and they send the local state to the non-computation ones. The non-computation replicas restore the local states on receipt of the states. The computation replicas taking the checkpoints are the *checkpoint* ones:

(1) to make all the checkpoint replicas to be synchronized to take the *consistent* checkpoint,
(2) to take the checkpoints, and
(3) to send the local state taken at the checkpoint to the non-computation replicas.

Since the checkpoint replicas may be faulty, more than $2f_i + 1$ checkpoint ones are required. The non-computation replicas can detect the faulty checkpoint replicas by using the majority rule.

4 Communication between Replica Groups

Suppose that output replicas $O_{i1}, \ldots, O_{iz_i}$ of p_i send message m to the input replicas $I_{j1}, \ldots, I_{jx_j}$ of p_j. There are two ways for p_i to send message m to p_j:

(1) each O_{ik} sends m to all the input replicas $I_{j1}, \ldots, I_{jx_j}$, and
(2) each O_{ik} sends m to a subset $I_j(O_{ik})$ of I_j.

In the first method, O_{ik} sends totally $x_1 + \cdots + x_{i-1} + x_{i+1} + \cdots + x_n$ messages to $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$ and each I_{jh} receives $z_1 + \cdots + z_{j-1} + z_{j+1} + \cdots + z_n$ messages [Figure 2] in the one-to-one network. In order to deliver m to $n-1$ processes $p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_n$, the replica group of p_i sends totally $z_i \cdot (x_1 + \cdots + x_{i-1} + x_{i+1} + \cdots + x_n)$ messages. It is named ***broadcast*** distribution one.

In the broadcast network, each I_{jh} needs to receive $(2f_i + 1)$ messages, and only $(2f_i + 1)$ ($\leq z_i$) output replicas in O_i can broadcast messages. Here, totally $(2f_i + 1)$ messages are transmitted.

Another way is that each O_{ik} sends m to not all the input replicas of p_j, but only $I_j(O_{ik}) \subseteq I_j$. I_{jh} has to receive at least $2f_i + 1$ messages from p_i and does not need to receive more than $2f_i + 1$ messages. Hence, $|\ \{\ O_{ik}\ |\ I_{jh} \in I_j(O_{ik})\ \}\ | \geq 2f_i + 1$ for every I_{jh}, and $I_i(O_{i1}) \cup \cdots \cup I_i(O_{iz_i}) = I_i$. The number of messages transmitted is $|\ I_i(O_{i1})\ | + \cdots + |\ I_i(O_{iz_i})\ |$. If each O_{ik} sends m to $(2f_i + 1)x_j/z_i$ input replicas of p_j, the minimum number $(2f_i + 1)x_j$ of messages are transmitted. It is ***selective*** broadcast distribution one.

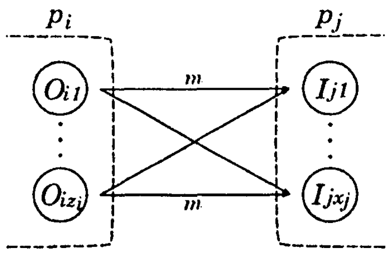

Figure 2: Input-output communication

5 Fault Treatment

We have to consider the fault of input, computation, and output replicas in the replica group of p_i:

(1) how to select a replica p_{ik} to which the units of p_{ij} are allocated, and
(2) how to allocate the units and start p_{ik}.

Here, suppose that p_{ij} with $\langle i_{ij}, c_{ij}, o_{ij} \rangle$ faults. The faulty unit is allocated to a replica, i.e. the replica activates the unit. If $i_{ij} = 1$, one operational p_{ik} with $i_{ik} = 0$ is selected and the input unit is given to p_{ik}. If $c_{ij} = 1$ and $o_{ij} = 1$, one operational p_{ik} with $c_{ij} = 0$ and $o_{ij} = 0$ is selected, and the computation and output units are given to p_{ik}, respectively. In the fully dispersed replica group, one replica p_{ik} is first selected among the dormant replicas. If there is no dormant replica, one unit with the minimum number of units is selected. The unit of p_{ij} is newly allocated to p_{ik}. If the replica group of p_i is multiplexed, one replica p_{ik} which has the maximum number if no candidate replica can be found in the replica group, a *clone* has to be created.

6 Concluding Remarks

In this paper, we have discussed how to make the group communication more fault-tolerant by replicating the protocol processes. The unit replication has been proposed as the replication, where the input, computation, and output units are independently allocated to the replicas, in order to support the robustness for the Byzantine fault of the process.

Reference

[1] 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.

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

[3] Turek, J. and Shasha, D., "The Many Faces of Consensus in Distributed Systems," *IEEE Computer Society Press*, 1994, pp.94-97.