6P－3

Fault-Tolerant Group Communication *

Kenji Sima, Takahiro Kimura, Hiroya Mita, and Makoto Takizawa †
Tokyo Denki University ‡
e-mail{sima,kimu,mita,taki}@takilab.k.dendai.ac.jp

1 Introduction

In distributed applications, a group of processes have to be communicated. Multiple ***system*** processes have to support the application processes with the *atomic* and *ordered* delivery of messages by using the network. By using the group communication supported by the system processes in the group, the application processes can send messages to the others atomically and in some delivery order. The distributed systems suffer from failures, i.e. system process and network faults. In this paper, we discuss a *fault-tolerance* group communication in the presence of process faults including Byzantine fault. We assume that the network is ***dependable***, i.e. messages are delivered to all the destinations atomically and in some specified order in the presence of message loss. From here, let ***processes*** mean system processes. Even if the processes in the group fault, the group has to support the application with the group communication.

In order to support the fault-tolerance group communication in the presence of process faults, the processes are replicated. That is, each process is realized by a ***replica*** group which is a set of multiple ***replicas*** of the process. Even if the replicas of the process fault, if at least one replica is operational, the process is considered to be operational. There are various replication strategies, ***passive***, ***active***, and ***semi-active*** replications [1]. In this paper, we present a ***hybrid*** replication method for replicating processes in order to support *fault-tolerancy*.

In section 2, we present a model of the system. In section 3, we discuss kinds of replications. In section 4, we present how to construct a group of replicas.

2 System Model

A communication system is composed of *application*, *system*, and *network* layers [Figure 1]. The network layer provides system processes with reliable group communication. That is, every system process receives every message sent without any message loss in the same order [3, 4]. The system processes p_1, ..., p_n cooperate with each other to support fault-tolerant group communication service for application processes by using the underlying network service.

A *logical* group G is composed of *logical* system processes, i.e. $G = \langle p_1, \ldots, p_n \rangle$. A ***physical*** group P_G of G is composed of replicas of the system processes, i.e. $P_G = \langle \{p_{11}, \ldots, p_{1m_1}\}, \ldots, \{p_{n1}, \ldots, p_{1m_n}\} \rangle$. Here, p_{ij} is a replica of p_i which is located in different processor. $\{p_{i1}, \ldots, p_{im_i}\}$ is a ***replca group*** of p_i. We make no assumption on process fault, i.e. Byzantine fault.

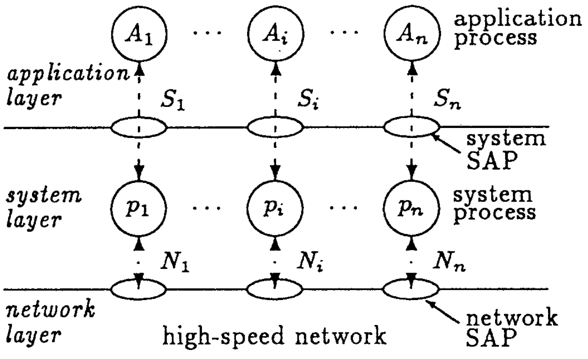

Figure 1: System model

3 Replication

Each replica group p_i is composed of replicas p_{i1}, ..., p_{im_i}. There are three kinds of replications [1]. *active* replication, *passive* replication, and *semi-active* replication.

In the active replications, every replica p_{ij} takes the same input and outputs the same result. Even if some replica of p_i faults, the computation of p_i can be continued as long as at least one replica is operational. The computation of p_i has to be deterministic because every replica has to do the same computation.

In the passive replication, there is one replica named a *coordinator*, say p_{i1}. p_{i2}, ..., p_{in} are *participants*. p_{i1} takes the input and outputs the result while any participants do no computation. p_{i1} takes the checkpoint where p_{i1} saves the local state ls_{i1} in the stable storage. Here, p_{i1} sends ls_{i1} to all the participants. On receipt of ls_{i1} from the coordinator, every p_{ij} saves ls_{i1} as the checkpoint in the stable storage and changes the local state to ls_{i1}. p_{ij} has to roll back to the checkpoint ck_{ij} taken most recently if the coordinator faults. Then, a new coordinator is selected among the operational participants and restarts the computation from the checkpoint. Hence, it takes some time for the replicas to roll back and restart. The passive replication can be adopted to the non-deterministic processes because only coordinator does the computation and the others catch up with it by receiving the checkpoint.

In the semi-active replication, all the replicas take the inputs and do computations while only the coordinator outputs the results. Like the passive replication, the coordinator p_{i1} takes a checkpoint and sends the local state ls_{i1} taken in the checkpoint to all the participants. On receipt of 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} is changed to ls_{i1}. The processes may be non-deterministic.

In this paper, we assume that one replica fails at the same time. In order to detect the coordinator fault, each replica group includes at least three coordinators which are actively replicated. If one coordinator replica outputs results different from the others, it is isolated as the fault process. Here, one replica has to

---

*フォールトトレラントグループ通信
†島 健司, 木村 貴博, 三田 浩也, 滝沢 誠
‡東京電機大学

be selected as a new coordinator replica. If the participant replica selected is passively replicated, it takes time to catch up with the current state of the coordinator because it has to start from the most-recent checkpoint. Therefore, some participants are semi-actively replicated, i.e. they take the same input as the coordinators but do not output the results. They are referred to as *candidates*. We propose a *hybrid* replication where a group includes multiple active coordinators, semi-active candidates, and passive participants.

4 Replica Groups

Each replica group p_i is composed of three kinds of replicas, i.e. coordinators $c_{i1}, \ldots, c_{it_i}$ $(t_i \geq 1)$, coordinator candidates $a_{i1}, \ldots, a_{iu_i}$ $(u_i \geq 0)$, and participants $s_{i1}, \ldots, s_{iv_i}$ $(v_i \geq 1)$.

4.1 Coordinators

The coordinators $c_{i1}, \ldots, c_{it_i}$ are realized by the active replications. Each coordinator c_{ij} takes the checkpoint, where the local state ls_{ij} is saved in the stable storage. Each c_{ij} sends the local state ls_{ij} to all the participants $s_{i1}, \ldots, s_{iv_i}$ and candidates $a_{i1}, \ldots, a_{iu_i}$. On receipt of ls_{ij} from the coordinators, the participants and candidates change the states.

In order to synchronize the computations of the coordinators, the protocol similar to the *two-phase commitment* (*2PC*) one is adopted. Since every coordinator replica must do the same computation, if same c_{ij} disagrees with the others, the coordinators consider that c_{ij} faults. Let *majority*(S) give a value which a majority of S takes for a set S of values.

[Checkpoint procedure]

(1) If c_{ij} would like to take a checkpoint, c_{ij} sends *CReq* message to all the coordinations of P_i.

(2) On receipt of *CReq*, c_{ij} takes a temporary checkpoint tc_{ij}. If c_{ij} succeeds in taking tc_{ij}, c_{ij} sends *Yes* with tc_{ij} to all the coordinators. Otherwise, c_{ij} sends *No* to all the coordinators.

(3) Each c_{ij} receives the reply r_{ih}, i.e. *Yes* or *No* from every coordinators. If $majority(r_{i1}, \ldots, r_{it_i}) = Yes$, c_{ij} sends *Chk* to all the coordinators which send *Yes*, and considers that coordinators which send *No* fault. Otherwise, c_{ij} sends *Abort* to all the coordinators which send *Yes*.

(4) On receipt of *Chk* from all the coordinators which c_{ij} thinks to be operational, c_{ij} changes tc_{ij} to be permanent.

(5) On receipt of *Abort* from someone sending *Yes*, c_{ij} removes tc_{ij} and tries to take the checkpoint again. □

The group communication has to support the delivery of messages to all the processes in the group. In each replica group of p_i, message m sent to p_i have to be delivered to $c_{i1}, \ldots, c_{im_i}$. Here, the replies are carried back by the messages.

[Atomic delivery]

(1) Each c_{ij} sends message to all the coordinator replicas in the group G.

(2) On receipt of m from c_{kh}, c_{ij} sends *Yes* as the reply rm_{ij} to all the coordinators in G. If c_{ij} fails to receive m, c_{ij} sends *No*.

(3) Each c_{ij} collects the replies $rm_{k1}, \ldots, rm_{kt_k}$ for each p_k. If $majority(rm_{k1}, \ldots, rm_{kt_k}) = Yes$, c_{ij} considers that m is received by p_k. Otherwise, p_k fails to receive m. In either case, c_{ij} considers that replicas sending the reply different from the majority fault and isolates them from G. □

4.2 Candidates

Each replica group p_i includes the candidates $a_{i1}, \ldots, a_{iu_i}$, which could be a coordinator if some coordinator faults. In order for some a_{ij} to take over the fault coordinator, the candidates are realized by the semi-active replication. Each a_{ij} takes the same inputs as the coordinators, but does not output any result. a_{ij} listens to the communication in G but does not send message to G. a_{ij} collects the checkpoints from all the coordinators. In the same way as the coordinators, a_{ij} obeys the majority of the checkpoints $ls_{i1}, \ldots, ls_{it_i}$.

4.3 Participants

The participant replicas $s_{i1}, \ldots, s_{iv_i}$ are passively replicated. Each s_{ij} neither takes inputs, outputs results, nor does the computation. On receipt of the checkpoint, ls_{ik} from c_{ik}, the participants change the local state to ls_{ik}. Each time s_{ij} receives the checkpoint, s_{ij} catches up with the coordinator.

4.4 Promotions and recovery

If a coordinator c_{ik} of p_i faults, one a_{ij} of the coordinator is selected to be the coordinator. Since the candidates are semi-active replicated, every candidate has the same state as the coordinator. Hence, a_{ij} takes over c_{ik} as soon as c_{ik} is detected to be fault by starting to output the results. At the same time a_{ij} takes over c_{ik}, one s_{ih} is selected to be a coordinator. s_{ih} can start the computation from the most recent checkpoint tk_{ih}. Since the current state ls_{ik} of a_{ik} might be different from tk_{ih}, s_{ih} has to catch up with the ls_{ik}. Each time a coordinator faults, the coordinators invoke the checkpoint procedure and send the local states to all the candidates. A procedure where candidates get coordinators and participants get candidates is referred to as *promotion*. If a fault replica recovers, the replica gets a participant.

5 Concluding Remarks

In this paper, we have discussed how to make the group communication more fault-tolerant by duplicating the protocol processes. The hybrid replication has been proposed as the replication in order to support the robustness for the Byzantine fault of the process and fail-safeness.

Reference

[1] Powell, D., Barrett, P., Bonn, G., Chereque, M., Seaton, D., and Verissimo, P., "The Delta-4 Distributed Fault-Tolerant Architecture," in *Readings in Distributed Computing Systems*, *IEEE COMPUTER SOCIETY PRESS* 1994, pp.223-248.

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

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

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