マルチメディア通信と分散処理 69－2
（1995．3．2）

一般合意プロトコル

矢羽田 千哲　滝沢 誠
東京電機大学
E-mail {chii, taki }@takilab.k.dendai.ac.jp

分散型アプリケーションは、通信網で相互接続された複数のプロセスから構成される。分散型アプリケーションでは、協調動作を行なうために、複数プロセス間での合意をとる事が必要である。本論文では、合意手順を4つのステップに分割し各ステップ毎に一般化を行なうことによる、一般合意プロトコルを提案する。本論文では、様々な合意プロトコルを、本プロトコルを構成する4つのステップで記述した例を示す。一般合意プロトコルでは、意志表示の変化、様々な決定論理、分散型制御に対処できる。

General Consensus Protocols

Chiaki Yahata and Makoto Takizawa
Tokyo Denki University

Distributed applications are realized by the cooperation of multiple processes. In the distributed applications, a group of processes have to make consensus to do the cooperation. In this paper, consensus protocol is decomposed into four general steps, i.e. pre-voting, voting, global decision, and final local decision. In the general consensus protocol, the process can change the mind after notifying other processes of the opinion, various kinds of global decision logics can be adopted, and the cooperation among the processes can be coordinated in centralized and distributed schemes.

1 Introduction

Distributed applications are realized by the cooperation of multiple processes, each of which is computed in one processor. The distributed applications like groupware [1] are realized by a *group* of multiple processes which are cooperated by communicating with one another by using communication networks. The processes in the group have to make some consensus in order to do the cooperation among them. In the consensus problem, one value has to be decided by a group of processes starting from a set of values of the processes. There are kinds of consensus protocols required by various distributed applications [9]. For example, the two-phase commitment (2PC) [6] protocol is used to realize the *atomic commitment* [2] of multiple subtransaction. In the commitment protocols, each process, i.e. subtransaction, cannot change the mind after notifying other processes of the vote, i.e. *Yes*(*commit*) or *No*(*abort*). After sending the vote to the coordinator process, the process is in an *uncertain* state [10], where all the processes can do is wait for the decision from the coordinator because every process cannot change the vote. However, in the human society, individuals often change the minds even after notifying others of the votes. For example, individuals often change the schedules. In other applications, some processes make an agreement even if the others disagree with them. For example, in a meeting of multiple members, something may be decided if a majority of the members agree on it. In addition to the atomic commitment, various kinds of decision logics have to be considered. When considering the cooperation of multiple processes, we have to think about what process coordinates the cooperation among the processes. In the 2PC protocol, the coordinator process plays a role of the centralized controller. Some meeting has no chair, i.e. every participant makes decision by itself. Thus, in addition to the centralized control, we have to consider the distributed control where there is no centralized controller.

In this paper, we assume that the communication network is reliable, i.e. each process can deliver messages to any processes with no message loss, and the network is not partitioned. We would like to discuss a general framework of consensus protocols in the presence of process fault, i.e. stop-by-failure. The following points have to be taken into account about the general consensus model:

1 each process can change the opinion even after notifying other processes of the opinion,
2 each process can express the opinion *No-idea* and *Anyone-OK* in addition to *Yes* and *No*,
3 various kinds of decision logics like *all-or-nothing* and *majority-consensus* can be adopted,
4 each process may be autonomous for the group i.e. it may not obey the global decision, and
5 how to control the coordination among the processes, i.e. *centralized* and *distributed* controls.

In section 2, we discuss what has to be taken into account on considering the consensus protocols. In section 3, we present a general model of consensus protocol. In section 4, we would like to

discuss various consensus protocols based on the general model.

2 Examples

A distributed system is composed of multiple processors interconnected by communication networks. A distributed application is realized by the cooperation of n (> 0) processes $p_1, \ldots, p_n$, each of which is computed in one processor. In the distributed applications, $p_1, \ldots, p_n$ have to make some consensus among themselves.

[Example 1] In the distributed database system [9], if a transaction manipulating multiple database systems, it has to be guaranteed that the transaction either updates all the database systems or none of them. It is an *atomic commitment* [6, 10]. There is one coordinator process p_0 in the two-phase commitment (2PC) protocol [2, 6]. If a transaction would terminate, p_0 sends $VoteReq$ message to all the processes $p_1, \ldots, p_n$. Each p_i sends Yes message to p_0 if p_i could commit the transaction. If not, p_i sends No to p_0 and then aborts the transaction. If p_0 receives Yes message from every process, p_0 sends *Commit* to $p_1, \ldots, p_n$. If p_0 receives No, p_0 sends *Abort* to all the processes voting Yes. On receipt of *Commit*, p_i commits the transaction. □

The commitment protocols like the 2PC protocol assume the following points:

1 no process can change the opinion after voting it,
2 the decision logic is based on the atomic commitment, i.e. *all-or-nothing* principle,
3 there is one centralized controller, i.e. the *coordinator* which coordinates the cooperation of the participate processes $p_1, \cdots, p_n$.
4 process is not autonomous, i.e. it obeys the decision of the coordinator, and
5 *No* dominates *Yes*, i.e. processes voting *No* abort unilaterally without waiting for the decision from the coordinator, and processes voting Yes may abort if the decision of the coordinator is *Abort*.

[Example 2] Let us consider an example that a group of individuals would like to go eating lunch together. First, the individuals in the group exchange the tentative opinions on going out. Here, one individual may say "I would like to go eating lunch together". Someone may say "No, I would not like to go eating lunch together". One may say "I have no idea". After listening to them, each individual expresses the opinion, i.e. *Yes*, *No*, *No-idea*, or *Anyone-OK*. Someone may express the opinion different from one which he expressed first. This means that he/she may change his/her mind here.

Now, the group obtains the opinions from all the individuals. The group has some logic to decide whether to go lunch. For example, only if all the individuals in the group agree on going eating lunch, they may go eating. They may go eating lunch if a majority in the group agree on it.

Next point is whether each individual obeys the global decision or not. One individual p has to obey the global decision if p depend on the group. If p is autonomous for the group, p may not obey the global decision. For example, some individual does not go eating lunch together with the group if he/she would not like. □

When considering the applications like the groupware as presented in the example, the assumptions on the commitment protocols have to be relaxed. Following the examples, the general consensus protocol has to take into account the following points :

1 each process can change the opinion even after notifying other processes of the opinion,
2 each process can express the opinion *No-idea* and $Anyone-OK$ in addition to *Yes* and *No*,
3 various kinds of decision logics like *all-or-nothing* and *majority-consensus* can be adopted,
4 each process may be autonomous, i.e. it may not obey the global decision, and
5 there are kinds of coordination among the processes, i.e, the cooperation of the process can be coordinated in the *centralized* on *distributed* scheme.

3 General Consensus Models

3.1 Basic procedure

A *consensus protocol* coordinates the cooperation among processes $p_1, \ldots, p_n$ in order to reach some decision. The general consensus protocol is composed of the following four steps.

[General consensus protocol]

1 First, each process p_i is required to express the opinion. p_i notifies all the processes of its opinion pv_i which is named *pre-vote* of p_i. This step is referred to as *pre-voting*.
2 p_i receives all the pre-votes $pv_1, \ldots, pv_n$ from $p_1, \ldots, p_n$. p_i makes a local decision on the basis of $pv_1, \ldots, pv_n$. Here, p_i can change the tentative opinion again. Formally, p_i obtains the *vote* $v_i = V_i(pv_1, \ldots, pv_n)$. Here, V_i is a function which gives some value for a tuple of values $pv_1, \ldots, pv_n$. p_i sends v_i to all the processes. This step is referred to as *voting*.
3 For the votes $v_1, \ldots, v_n$ obtained from $p_1, \ldots, p_n$, a global decision $v = GD(v_1, \ldots, v_n)$ is obtained. GD is a function which gives v for a tuple of the votes $v_1, \ldots, v_n$. This step is referred to as *global decision*.
4 p_i obtains the global decision v. Based on v and the votes $v_1, \cdots, v_n$, p_i makes the final local decision and obtains $d_i = LD_i(v_1, \ldots, v_n, v)$. LD_i is a function which gives the final local decision d_i from the votes $v_1, \ldots, v_n$ and v. This step is referred to as *final local decision*. □

Let D be a set $\{d_1, \cdots, d_m, \bot, \top\}$ of values. Here, $\bot$ means that it is not decided which one from $d_1, \cdots, d_m$ is taken, e.g. process p_i has no idea on the decision. $\top$ means that any of $d_1, \cdots, d_m$ is allowed, e.g. p_i can vote any one of $d_i, \cdots, d_n$. Initially, p_i has one value pv_i in D as the pre-vote. V_i is a function from D^n into D, i.e. for every $pv_j \in D$ $(j = 1, \ldots, n)$, $V_i(pv_1, \ldots, pv_n) = v_i \in D$. For example, if p_i has no idea, p_i notifies all the processes of $\bot$. p_i receives the pre-votes $pv_1, \ldots, pv_n$ from all the processes. Based on the pre-votes obtained, p_i makes the final local decision by V_i. For example, if p_i obeys p_j's opinion, $v_i = V_i(pv_1, \ldots, pv_n) = pv_j$. Here, it is noted that v_i may be different from pv_i. While listening to other opinions, i.e. pre-votes, p_i can change the opinion. p_i notifies all the processes of the vote v_i obtained by V_i.

Here, all the votes $v_1, \ldots, v_n$ are collected by one process or every process, depending on the coordination scheme. For example, $v_1, \cdots, v_n$ are sent to one coordinator in the centralized scheme. The global decision $v = GD(v_1, \ldots, v_n)$ is obtained from the votes. GD is a function from D^n into D. As an example, let us consider the 2PC protocol where $D = \{1, 0\}$. Each process means a database server. Each process p_i votes $v_i \in \{1, 0\}$. If all the processes vote 1, they commit. If at least one process votes 0, all the processes abort. Hence, $GD(v_1, \ldots, v_n) = 1$ if $v_j = 1$ for $j = 1, \ldots, n$. $GD(v_1, \ldots, v_n) = 0$ if some $v_j = 0$. If the global decision is a value voted by a majority of the processes, $GD(v_1, \ldots, v_n) = v$ if $|\{v_i | v_i = v\}| > \frac{n}{2}$.

Each process p_i receives the global decision v. Problem is how p_i behaves on obtaining v, i.e. p_i obeys v or not. p_i has to obey v if p_i is not autonomous. If p_i is autonomous, p_i may not obey v even if v is decided globally as presented in Example 2. p_i makes a final local decision by $LD_i(v_1, \ldots, v_n, v)$. LD_i is a function from D^{n+1} to D. For example, if p_i makes the decision of v_i independently of v, $LD_i(v_1, \ldots, v_n, v) = v_i$. If p_i agrees on v, $LD_i(v_1, \ldots, v_n, v) = v$. If p_i depends on another p_j, $LD_i(v_1, \cdots, v_n, v) = v_j$.

3.2 Process states

A local state of each process p_i is given as a tuple $\langle pv_i, v_i, d_i \rangle$ where pv_i is the pre-vote, v_i is the vote, and d_i is the value finally decided by p_i. The local state denotes how p_i makes the decision. p_i changes the local state on receipt of messages. Here, suppose that D is a set $\{1, 0, \bot, \top\}$ for simplicity. First, let us consider an initial state of p_i.

For every state $\langle a, b, c \rangle$, $b = c = \bot$ if $a = \bot$, and $c = \bot$ if $b = \bot$. A state $\langle a, b, c \rangle$ is referred to as *transitable* if $b = \bot$ or $c = \bot$. $\langle a, b, c \rangle$ is referred to as *mind-changeable* if $b = \bot$. For example, after expressing the pre-votes 1, p_i can vote 0 different from the pre-vote, i.e. $\langle 1, \bot, \bot \rangle$ is transited to $\langle 1, 0, \bot \rangle$.

[Definition] For a, b, and $c \in D$, if $\langle a, b, \bot \rangle$ is transited to $\langle a, b, c \rangle$, c is referred to as *dominate* b if $b \neq c$ (written as $b \prec c$). □

$a \prec b$ means that process p_i can change a to b. $a \equiv b$ means that $a \succeq b$ and $a \succ b$, and $a \prec b$. $a \succeq b$ means that $a \succ b$ or $a \equiv b$. For example, in the commitment protocol, $0 \succ 1$ because the process voting 0 only aborts. Figure 1 shows the state transition of the 2PC protocol. $\langle 0, 0, 0 \rangle$ means that a process voting 0 aborts. $\langle 1, 1, \bot \rangle$ means that the process votes 1. Up to the global decision, $\langle 1, 1, \bot \rangle$ is transited to $\langle 1, 1, 1 \rangle$ if the process commits, $\langle 1, 1, 0 \rangle$ if the process aborts.

D is partially ordered on $\prec$. Since $\top$ can be changed to any value in D, $\top$ is a *bottom* of D, i.e. for every d in D, $\top \prec d$. A value d in D is referred to as *minimal* in D iff there is no value d_k in D such that $d_k \prec d$. For example, in the 2PC protocol, $D = \{1, 0, \bot, \top\}$, $\top$ is minimal in D. If D has only one minimal value d, *minimum*, i.e. for every d_h in D, $d \prec d_h$. Each process p_i can vote the minimum d instead of voting $\top$. For example, in the 2PC protocol, p_i can vote 1 if p_i can commit and abort, i.e. $\top \prec 1$ and 1 is minimum. A value d in D is referred to as *maximal* in D iff there is no value d_k in D such that $d \prec d_k$. If there is only one maximal value d in D, i.e. for every d_h in D, $d_h \prec d$, d is the *top* of D. If p_i votes the maximal value d, p_i never changes the mind. Because d cannot be changed to any value.

For every pair of d_k and d_h in D, the *upper bound* of d_k and d_h is a set $\{d | d \in D, d_k \prec d$, and $d_h \prec d\}$ of values dominating both d_h and d_k. $d_k \cup d_h$ denotes the least upper bound (*lub*) of d_k and d_h. That is, $d_k \cup d_h$ is a value d such that $d_k \preceq d$, $d_h \preceq d$, and there is no d' such that $d_k \prec d' \prec d$ and $d_h \prec d' \prec d$. If not exists, $d_k \cup d_h = \bot$

Let $\langle a, b, c \rangle$ be a state. If b is maximal in D, c has to be b because process voting b cannot change the vote. Hence, if b is maximal, $c = b$, i.e. $\langle a, b, b \rangle$. For example, if $b = 0$, i.e. process p_i votes *No*, p_i aborts $(c = 0)$, i.e. $\langle 0, 0, 0 \rangle$. Thus, if b is maximal, $\langle a, b, c \rangle$ cannot be transited into another state.

[Definition] A state $\langle a, b, c \rangle$ where b is maximal is referred to as *maximal*.□

State which is not maximal is referred to as *transitable*. Here, let us consider a state transition from a state $\langle a, b_1, c_1 \rangle$ into $\langle a, b_2, c_2 \rangle$ where b_1 is not maximal. If $b_1 \prec b_2$, or $b_1 = b_2$ and $c_1 \prec c_2$, $\langle a, b_1, c_1 \rangle$ can be transited into $\langle a, b_2, c_2 \rangle$. For example, $1 \prec 0$ in the 2PC protocol. $\langle 1, 1, \bot \rangle$ can be transited into $\langle 1, 1, 0 \rangle$ and $\langle 1, 1, 1 \rangle$ while $\langle 0, 0, 0 \rangle$ cannot be transited Figure 1. Processes which are in a transitable state after voting have to wait for the global decision. On the other hand, processes which are in a maximal state can terminate, because they made their final decisions already. For

example, processes voting 0 aborts and processes which have aborted cannot commit.

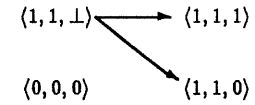


Figure 1: State transition of the 2PC protocol

3.3 Global decision

After obtaining the votes $v_1, \ldots, v_n$ from all the processes $p_1, \ldots, p_n$, the global value v is globally decided by using the function $GD : D^n \rightarrow D$. For every tuple $\langle v_1, \cdots, v_n \rangle \in D^n$, $GD(v_1, \cdots, v_n)$ gives some value v in D. If $v_1 \cup \cdots \cup v_n \preceq v$, every process p_i can change the vote v_i to v. Unless $v_i \prec v$ for some p_i, p_i cannot change the vote to v. For example, suppose that there is a transaction manipulating three database systems A, B and C, which votes 0, 1, and 1, respectively, in the 2PC protocol. If $GD(0,1,1) = 0$, B and C can change the vote to 0, i.e. can abort. On the other hand, suppose that $GD(0,1,1) = 1$. A cannot commit because B aborts already although B and C can commit.

[Definition] GD is referred to as *regular* if for every $\langle v_1, \cdots, v_n \rangle \in D^n$, $v_1 \cup \cdots \cup v_n \preceq GD(v_1, \cdots, v_n)$. □

If GD is regular, every process can change the vote into the global decision. If not, some process p_i may not obey the global decision unless $v_i \preceq v$. For example, process aborting cannot obey the global decision if the global decision is *commit*.

There are the following kinds of global decisions:

1 Commitment decision : $GD(v_1, \ldots, v_n) = 1$ if every $v_i = 1$, $GD(v_1, \ldots, v_n) = 0$ if some $v_i = 0$ where $D = \{1, 0, \bot, \top\}$.
2 Majority-consensus decision on v: $GD(v_1, \ldots, v_n) = v$ if $|\{v_i | v_i = v\}| > \frac{n}{2}$, otherwise $GD(v_1, \ldots, v_n) = v_1 \cup \cdots \cup v_n$.
3 $\binom{n}{n}$-decision on v: $GD(v_1, \ldots, v_n) = v$ if every $v_i = v$, otherwise $GD(v_1, \ldots, v_n) = v_1 \cup \cdots \cup v_n$.
4 $\binom{n}{r}$-decision on v: $GD(v_1, \ldots, v_n) = v$ if $|\{v_i | v_i = v\}| \geq r$, otherwise $GD(v_1, \ldots, v_n) = v_1 \cup \cdots \cup v_n$.
5 Minimal-decision: $GD(v_1, \cdots, v_n) = v_1 \cup \cdots \cup v_n$.
6 Super-vote: $GD(v_1, \cdots, v_n) = v_i$ if p_i has the highest priority.

In the commitment decision, only if all the votes are 1, 1 is globally decided. If some process votes 0, 0 is decided. It is used by the 2PC protocol. The commitment decision on $\{0, 1, \bot, \top\}$ can be extended to $D = \{d_1, \cdots, d_m, \bot, \top\}$. $GD(v_1, \cdots, v_n) = v$ if every $v_i = v$. Otherwise, $GD(v_1, \cdots, v_n) = v_1 \cup \cdots \cup v_n$. $v_1 \cup \cdots \cup v_n$ means a value to which every v_i can be changed. If such a value does not exist, i.e. $v_1 \cup \cdots \cup v_n = \bot$, nothing is decided. In the 2PC protocol, $1 \prec 0$. Hence, if some $v_j = 0$, $v_1 \cup \cdots \cup v_n = 0$.

In the majority decision on v, if a majority of the processes vote some v, v is globally decided. Otherwise, nothing is decided if $v_1 \cup \cdots \cup v_n = \bot$.

In the $\binom{n}{n}$-decision on v, if all the processes vote some v, v is globally decided. Otherwise, nothing is decided if $v_1 \cup \cdots \cup v_n = \bot$.

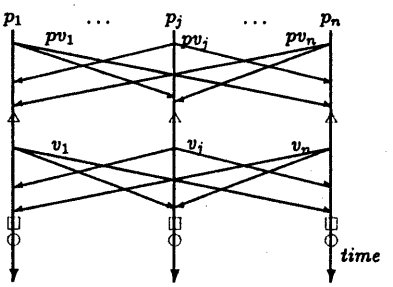
In the $\binom{n}{r}$-decision on v, if $r(\leq n)$ processes vote some v, v is globally decided. If $r > \frac{n}{2}$, the $\binom{n}{r}$-decision is the majority one on v.

In the minimal decision, every process p_i agrees on the value $v = v_1 \cup \cdots \cup v_n$ where v is minimal values which every v_i can be changed to. If $v_1 \cup \cdots \cup v_n = \bot$, nothing is decided. The binary commitment decision is a kind of the minimal decision. In the super-vote, the global decision is value voted by the higher priority process.

In addition, GD can be defined based on the application semantics. For example, if every process obeys p_i's opinion, $GD(v_1, \ldots, v_n) = v_i$.

3.4 Coordination schemes

Another point is concerned with which process coordinates the cooperation among the processes $p_1, \cdots, p_n$. If one process p_0 named a *coordinator* coordinates the cooperation of the processes $p_1, \cdots, p_n$, it is referred to as *centralized control*. The 2PC [6] and 3PC [10] protocols are the examples of the centralized control. In the centralized control, every p_i first sends the pre-votes pv_i to p_0. p_0 collects $pv_1, \ldots, pv_n$, and sends $\langle pv_1, \ldots, pv_n \rangle$ to $p_1, \ldots, p_n$. On receipt of $\langle pv_1, \ldots, pv_n \rangle$, p_i decides the vote $v_i = V_i(pv_1, \ldots, pv_n)$. p_i sends v_i to p_0. On receipt of all the votes $v_1, \ldots, v_n$, p_0 makes the global decision of $v = GD(v_1, \ldots, v_n)$, and then sends v to $p_1, \ldots, p_n$. On receipt of v, p_i makes the final local decision of $d_i = LD_i(v_1, \ldots, v_n, v)$.



△ : local decision (V)
□ : global decision (GD)
○ : final local decision (LD)
Figure 2: Distributed control

If there is no centralized controller, it is referred to as ***distributed control***. In the distributed con-

trol [Figure 2], each process p_i sends the pre-vote pv_i to $p_1, \ldots, p_n$. On receipt of $pv_1, \ldots, pv_n$, p_i makes the local decision of $v_i = V_i(pv_1, \ldots, pv_n)$ by itself. p_i sends the vote v_i to $p_1, \ldots, p_n$. On receipt of $v_1, \ldots, v_n$, every p_i makes the same global decision of $v = GD(v_1, \ldots, v_n)$. Then, p_i makes the final local decision of $d_i = LD_i(v_i, \ldots, v_n, v)$. Each p_i has the same GD and makes the decision by itself on the basis of GD. p_i can make the decision without waiting for the decision from the coordinator. In the distributed control, every process p_i has to send message m to all the other processes. If the broadcast network is used, p_i can send m to all the processes by issuing one data transmission request of m to the network. If not, p_i has to issue n times requests of m.

4 Consensus Protocols

We would like to describe various consensus protocols in terms of the general model.

4.1 Atomic commitment protocol

First, we would like to present the atomic commitment protocol. In the commitment protocol, suppose that 1 means *commit* and 0 means *abort*.

Since each process cannot change the mind, the pre-vote is the same as vote. All the processes voting 0 *abort* unilaterally, i.e. without waiting for the global decision. The initial state of process p_i is either $\langle 1, 1, \perp \rangle$ or $\langle 0, 0, 0 \rangle$. $\langle 0, 0, 0 \rangle$ is maximal. On the other hand, processes voting 1 may commit or abort up to the global decision. Hence, 0 dominates 1, i.e. $0 \succ 1$.

Only if all the processes vote 1, they commit. If some process votes 0, all the processes abort. GD is the commitment decision, $GD(1, \ldots, 1) = 1$ and $GD(\ldots, 0, \ldots) = 0$. GD is regular.

The final local decision is $LD_i(v_1, \ldots, v_n, v) = v$ because p_i voting 1 obeys the global decision. That is, the processes voting 1 are not autonomous.

4.2 Extended commitment protocol

As presented before, each process can vote either 1(Yes) or 0(No) in the conventional 2PC protocol. We would like to extend the commitment protocol so that each process can vote $\perp$ (*No_idea*) and $\top$ (*Anyone-OK*). In the 2PC protocol, each process p_i may not be able to vote even if p_i receives *VoteReq* from the coordinator p_0, e.g. p_i is too heavy-loaded to vote. In such a case, p_i can vote $\perp$ instead of voting *1* or *0*, or p_i can be considered to vote $\perp$ if no reply of *VoteReq* is received in some time units. Processes voting $\perp$ or $\top$ are referred to as *undecided*. Processes voting 0 or 1 are referred to as *decided*. First, we would like to present the basic protocol.

[Basic protocol]

1 First, the coordinator p_0 sends *VoteReq* to all the processes $p_1, \ldots, p_n$, e.g. if a transaction T finishes all the operations.

2 On receipt of *VoteReq* from p_0, each p_i sends 1 or 0 to p_0. In addition, p_i may send $\perp$ to p_0 if p_i could not decide whether to vote 1 or 0. p_i may send $\top$ to p_0 if p_i could commit or abort the transaction.

3 If p_0 receives 1 from all the processes and p_0 would like to commit, p_0 sends *Commit* to $p_1, \ldots, p_n$. If p_0 receives 0 from at least one process and p_0 would not like to commit, p_0 sends *Abort* to all the processes voting 1, $\perp$, or $\top$. If p_0 receives $\top$ from all the processes, every p_i obeys p_0's decision, i.e. if p_0 would like to commit, p_0 sends *Commit* to $p_1, \ldots, p_n$, otherwise i.e. if p_0 would not like to commit, p_0 sends *Abort* to $p_1, \ldots, p_n$.

4 Here, some process p_i votes $\perp$. If all the decided processes vote 1, p_0 sends *Commitable* to the undecided processes.

5 If p_i votes $\perp$, on receipt of *Commitable*, p_i sends 1 to p_0 if p_i could commit, 0 to p_0 if p_i could abort. p_i sends $\perp$ to p_0 again if p_i still could not decide 1 or 0.

6 If p_0 could not receive 1 or 0 from all the undecided processes after sending *Commitable* $m(\geq 1)$ times, p_0 sends *Abort* to all the processes, i.e. voting 1 or 0.

7 After voting 1, $\perp$, or $\top$ if p_i receives *Abort* from p_0, p_i aborts. After voting $\top$ and 1, if p_i receives *Commit* from p_0, p_i commits. □

Next, suppose that p_0 faults after each process p_i votes before sending the reply to all the processes. After voting, process p_i voting 1 or $\perp$ invokes the following termination protocol if p_i times out.

[Termination protocol]

1 p_i sends *StateReq* to all the processes.

2 On receipt of *StateReq* from p_i, each process p_i sends the local state to p_i, i.e. 1 if p_j votes 1, 0 if p_j votes 0, $\perp$ if p_j votes $\perp$, $\top$ if p_j votes $\top$, *Commitable* if p_j receives *Commitable*, *Commit* if p_j receives *Commit*, and *Abort* if p_j receives *Abort*.

3 p_i makes the decision by the *termination* rule if p_i receives the replies of *StateReq*. □

[Termination rule]

1 If p_i receives *Commitable* from some process, p_i commits if p_i votes 1 or $\top$.

2 If p_i receives *Abortable* from some process, p_i aborts.

3 If p_i receives $\perp$ from some process and p_i votes $\perp$, p_i aborts.

4 If p_i receives 1 from all the processes, p_i blocks.

5 If p_i votes $\perp$ and receives the states except *Commit* or *Abort*, p_i votes 1 or 0. □

If all the operational processes are in the state of 1, they have to wait, i.e. *block* [10].

Next, suppose that process p_i recovers from the failure. Suppose that p_i records the local state

in the log L_i. p_i invokes the following recovery protocol if p_i recovers.

[Recovery protocol]

1 p_i restores from L_i the state where p_i failed.
2 If p_i is not in a state of *Commitable*, *Aborted*, $\perp$, or $\top$, p_i asks other processes in the same way as the termination protocol.
3 If p_i is in a state of $\perp$ or $\top$, p_i aborts. □

4.3 Distributed extended commitment protocol

Processes $p_1, \ldots, p_n$ cooperate as follows without any centralized controller.

[Basic protocol]

1 Some p_i broadcasts *VoteReq* to all the processes.
2 On receipt of *VoteReq*, p_j broadcasts the votes, i.e. 1, 0, $\perp$, or $\top$.
3 On receipt of 1 from all the processes, p_i commits.
4 On receipt of 0 from some process, p_i aborts.
5 On receipt of $\top$ from all the processes, p_i can commit or abort by itself.
6 If p_i receives 1 from at least one process and $\top$ from all the other processes, p_i commits.
7 If p_i receives 1 from at least one process and $\perp$ from all the other processes, p_i waits. If p_i times out, p_i sends *VoteReq* to the processes voting $\perp$.
8 If p_j votes $\perp$, on receipt of *VoteReq*, p_j votes by step 2. □

Suppose that p_j stops by failure. If p_i had not received the vote of p_j in some time units p_i invokes the termination protocol.

[Termination protocol]

1 p_i sends *StateReq* to all the operational processes.
2 On receipt of *StateReq*, p_i sends the local state to all the processes.
3 On receipt of the states, p_i makes the decision by the termination rule. □

[Termination rule]

1 If p_i receives committed state from some processes, p_i commits.
2 If p_i receives aborted state from some processes, p_i aborts.
3 If all the processes are undecided, p_i aborts. □

p_j records the local state in the log L_j. p_j invokes the following recovery protocol if p_j recovers.

[Recovery protocol]

1 p_j restores from L_j the state where p_j failed.
2 If p_j is not in a state of *Commitment*, *Abort*, *undecided*, p_j asks other processes in the same way as the termination protocol.
3 If p_j is in a state of *undecided*, p_i aborts. □

5 Concluding Remarks

This paper discusses general framework of various consensus protocols. The general consensus protocol is composed of four steps, i.e. pre-voting, voting, global decision, and final local decision. We have described various consensus protocols in terms of the model. By composing the procedures for pre-voting, voting, global decision, and final local decision, we can make the consensus protocols required in the applications.

References

[1] Barborak, M., Malek, M., and Dahbura, A., "The Consensus Problem in Fault-Tolerant Computing," *ACM Computing Surveys*, Vol.25, No.2, 1993, pp.182-184,198-199.

[2] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addison-Wesley Publishing Company*, 1987, pp.222-261.

[3] 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.

[4] Ellis, C. A., Gibbs, S. J., and Rein, G. L., "Groupware," *Comm. ACM*, Vol.34, No.1, 1991, pp.38-58.

[5] Fischer, J. M., Lynch, A. N., and Paterson, S. M., "Impossibility of Distributed Consensus with One Faulty Process," *Journal of ACM*, Vol.32, No.2, 1985, pp.374-382.

[6] Gray, J., "Notes on Database Operating Systems, An Advanced Course," *Lecture Notes in Computer Science*, No.60, 1978, pp.393-481.

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

[8] Lamport, L. and Shostak, R., "The Bysantine Generals Problem," *ACM-Trans.Programming Languages and Systems*, Vol.4, No3,1982, pp.382-401.

[9] Ozsu, M. T. and Valduriez, P., "Principle of Distributed Database Systems," *Prentice-Hall*, 1990.

[10] Skeen, D. and Stonebraker, M., "A Formal Model of Crash Recovery in a Distributed System," *IEEE Computer Society Press*, Vol.SE-9, No.3, 1983, pp.219-228.

[11] Turek, J. and Shasha, D., "The Many Faces of Consensus in Distributed Systems," *Distributed Computing Systems, IEEE Computer Society Press*, 1994, pp.83-91.