システムソフトウェアと オペレーティング・システム 74－2
マルチメディア通信と分散処理 81－2
（1997. 2. 27）

Consensus Protocol with Partially Ordered Domain

Iwao Shimojo, Takayuki Tachikawa, Hiroaki Higaki, and Makoto Takizawa

Tokyo Denki University
E-mail {gan,tachi,hig,taki}@takilab.k.dendai.ac.jp

Distributed applications are realized by the cooperation of multiple processes. A group of the processes have to make consensus to do the cooperation. The processes exchange the values with the other processes to make consensus. The processes are referred to as consent if each process takes one value which satisfies a consensus condition. A dominant relation among the values is defined to show what values the processes can take after taking a value in the consensus protocol. In this paper, we discuss how to make consensus for a group of the processes by using the dominant relation.

半順序領域における合意プロトコル

下城 巌　立川 敬行　桧垣 博章　滝沢 誠

東京電機大学理工学部経営工学科

複数プロセスが協調動作を行なうためには、プロセス間での合意が必要となる。合意プロトコルでは、プロセス間で値の交換を行ない、各プロセスが合意条件を満足する値を取るならば、プロセスは合意したとする。合意プロトコルでは、ある値を取った後に、次にどの値が取られるべきであるかを値間の支配関係で定義する。即ち、各プロセスは、支配関係を用いることで、ある値を取った後に、次にどの値を取るかを決定する。本稿では、支配関係を用いることで、プロセスグループで、いかに合意を取るかを議論する。

1 Introduction

A distributed system is composed of multiple processors interconnected by communication networks. In distributed applications like groupware [1], a group of multiple processes are cooperated, each of which is computed in one processor. In the cooperation, the processes make consensus if they take values which satisfy a consensus condition by exchanging values.

In this paper, we discuss a consensus protocol where a group of multiple processes take values among m (≥ 2) values $v_1, \ldots, v_m$ while only two values, i.e. 1 (commit) and 0 (abort) are considered in the two-phase commitment (2PC) protocol [2, 6]. In the 2PC protocol, the process taking 1 can take 1 or 0 although the process taking 0 can only take 0. After notifying other processes of 0, the process is *uncertain* [11] where the processes have to wait for the decision of the coordinator. Here, the process may block if the coordinator is faulty. In more general consensus protocols, after taking some value v, the process can take one value which depends on v. Takizawa *et. al.* [14,15] introduce a *dominant* relation $\preceq$ in which the values in the domain D are partially ordered. That is, a value v_1 in D *dominates* v_2 ($v_1 \preceq v_2$) if the process can take v_2 after taking v_1. There are two cases, i.e. homogeneous and heterogeneous ones. In the first case, one dominant relation holds for all the processes in the group. In the heterogeneous case, a dominant relation is defined for each process, that is, even if one process takes a value v' after taking v, another process might not take v' after v. We discuss how to make the consensus among the processes by using $\preceq$. In addition, we make clear how the processes block in the presence of the process faults.

In the 2PC protocol, the processes *commit* only if all the processes take 1. Otherwise, the processes abort. In addition to the atomic commitment, various kinds of decision logics like *majority-consensus* have to be considered. We discuss what kinds of the consensus decisions can be adopted based on the dominant relation.

In the 2PC, the coordinator process makes the consensus decision and delivers the decision to all the processes. Some meeting has no chair, i.e. every participant makes decision by itself. Thus, we have to consider the distributed protocol [9, 12] in addition to the centralized one. In the 2PC, only two values are coordinated. In this paper, we discuss the m-ary consensus protocol with the ordered domain composed of multiple values obtained by extending the 2PC.

In sections 2 and 3, we overview the consensus protocols and discuss the dominant relation. In sections 4 and 5, we present the basic consensus protocol and discuss the decision logics of consensus protocols. In section 6, we discuss the m-ary consensus protocol.

2 Commitment Protocol

If a distributed transaction T [10] manipulates multiple database systems, it has to be guaranteed that T either updates all or none of the database systems. It is *the atomic commitment* [6, 11]. There is one *coordinator* process p_0 in the 2PC protocol [2, 6]. If T would commit, p_0 sends a *Prepare* message to all the *participant* processes $p_1, \ldots, p_n$. Otherwise, p_0 sends *Abort*. Each p_i sends 1 to p_0 if p_i could commit. Otherwise, p_i sends 0 to p_0 and then aborts. If p_0 receives 1 from every process, p_0 sends *Commit* (1) to $p_1, \ldots, p_n$. If p_0 receives 0, p_0 sends *Abort* to all the processes sending 1. Here, p_0 may send *Abort* (0) even if p_0 receives 1 from all the processes. For example, if the application process p_0 in the client receives an

interrupt signal from the user after sending *Prepare*, p_0 sends *Abort*. On receipt of *Commit*, p_i commits. On receipt of *Abort*, p_i aborts. The commitment protocols like the 2PC make the following assumptions:

1 No participant process can change the value after notifying the others of 1 or 0.
2 The decision logic is based on the atomic commitment, i.e. *all-or-nothing* principle.
3 The coordinator p_0 makes a global decision by using the values obtained by the processes. Even after all the processes take 1, p_0 may make a decision 0.
4 0 dominates 1. The processes sending 0 abort unilaterally without waiting for the decision from p_0. The processes sending 1 may abort if the decision of p_0 is *Abort*.
5 The process is not autonomous, i.e. it obeys the global decision of the coordinator.

p_i is *uncertain* [11] after taking 1 until receiving the decision from p_0. The uncertain process may block if p_0 is faulty because all p_i can do after sending 1 is wait for the decision from p_0.

3 Dominant Relation

We discuss how each process can take values in the consensus protocol in a group G of multiple processes $p_1, \ldots, p_n$.

3.1 Precedency

Let D be a domain, i.e. a set of possible values to be taken by the processes in the group G. In the consensus protocol, each process p_i first takes an initial value v_{i0} in D. p_i takes a value v_{i1} in D given $\langle v_{10}, \ldots, v_{n0} \rangle$ and notifies the other processes of v_{i1}. Then, p_i obtains a new tuple $\langle v_{11}, \ldots, v_{n1} \rangle$ by exchanging the values with the others. Thus, p_i takes a value $v_{i,j+1}$ based on $\langle v_{1j}, \ldots, v_{nj} \rangle$ obtained at the jth round. This step is the $(j+1)$th *round*. Finally, the consensus protocol terminates if $\langle v_{1t}, \ldots, v_{nt} \rangle$ satisfies some consensus condition M. A global state of G is given in a tuple $\langle v_{1j}, \ldots, v_{nj} \rangle$ where each value $v_{ij} \in D$ is taken by p_i at the jth round. The global state $\langle v_{1j}, \ldots, v_{nj} \rangle$ is transited to $\langle v_{1,j+1}, \ldots, v_{n,j+1} \rangle \in D^n$ at the $(j+1)$th round.

For example, one person can go swimming after saying "go skiing" but another cannot. Thus, values which p_i can take at the $(j+1)$th round depend on v_{ij} taken at the jth round.

[Definition] For every pair of values x and y in D, x *precedes* y in a process p_i ($x \Rightarrow_i y$) iff p_i can take y at the next round after taking x. □

$\Rightarrow_i$ is not transitive. Let $\Pi_i(x)$ be $\{ y \mid x \Rightarrow_i y \}$. p_i selects one value $v_{i,j+1}$ in $\Pi_i(v_{ij})$ at the jth round. A value x *transitively* precedes y in p_i ($x \Rightarrow\!\!\!\!\Rightarrow_i y$) if $x \not\Rightarrow_i y$ and there is some value z such that $x \Rightarrow_i z \Rightarrow\!\!\!\!\Rightarrow_i y$.

For every pair of values x and y in D, y is *reachable* from x in p_i ($x \rightarrow_i y$) iff $x \Rightarrow_i y$ or $x \Rightarrow\!\!\!\!\Rightarrow_i y$. Here, let H, K, and W show "I would like to go to hot spring, go skiing, go swimming," respectively. If p_i can take H just after taking K, $K \Rightarrow_i H$. If $K \Rightarrow_i H \Rightarrow_i W$ but $K \not\Rightarrow_i W$, $K \Rightarrow\!\!\!\!\Rightarrow_i W$. $x \rightarrow_i y$ means that p_i can take y at one or more than one step after taking x. On the other hand, $x \Rightarrow_i y$ means that p_i can take y just after taking x. x and y are *equivalent* in p_i ($x \equiv_i y$) iff $x \Rightarrow_i y$ and $y \Rightarrow_i x$. $x \Rightarrow y$ and $x \equiv y$ iff $x \Rightarrow_i y$ and $x \equiv_i y$, respectively, for every p_i.

For every pair of values x and y in D, $x \cup_i y$ means the least upper bound (*lub*) of x and y on $\rightarrow_i$. That is, $x \cup_i y$ shows a value z such that (1) $x \rightarrow_i z$ and $y \rightarrow_i z$, and (2) there is no value w such that $x \rightarrow_i w$, $y \rightarrow_i w$, and $w \rightarrow_i z$. $x \cup y$ is the *lub* of x and y on $\rightarrow$. $x \cup_* y$ is such a value z that $x \cup_i y \rightarrow_i z$ and there is no w where that $x \cup_i y \rightarrow_i w \rightarrow_i z$ for every p_i. The greatest lower bounds (*glb*) $\cap_i$, $\cap$, and $\cap_*$ are defined in the similar way as $\cup_i$, $\cap$, and $\cup_*$. For every tuple $\langle v_1, \ldots, v_n \rangle \in D^n$, the upper bound of $\langle v_1, \ldots, v_n \rangle$ is $\{v \mid v_i \rightarrow_i v$ for every $p_i \}$. $\cup\langle v_1, \ldots, v_n \rangle$ is the *lub* of $\langle v_1, \ldots, v_n \rangle$. $\cup_*\langle v_1, \ldots, v_n \rangle$ shows a value x such that $v_i \rightarrow_i x$ for every p_i and there is no value y such that $v_i \rightarrow_i y \rightarrow_i x$ for every p_i.

[Definition] For every pair of values x and y in D and every process p_i, y *dominates* x in p_i ($x \prec_i y$) iff $x \rightarrow_i y$ but $y \not\rightarrow_i x$. □

$x \prec_i y$ means that p_i can take y after x but p_i cannot take x after y. $x \preceq_i y$ iff $x \prec_i y$ or $x \equiv_i y$. $x \prec y$ and $x \preceq y$ iff $x \prec_i y$ and $x \preceq_i y$ for every p_i, respectively. x and y are comparable in the group G if $x \preceq y$ or $y \preceq x$. Let $\pi_i(x)$ be $\{y \mid x \preceq_i y\}$.

The group G is *homogeneous* iff $\preceq_i = \preceq_j$ for every pair of p_i and p_j in G. That is, every process has the same dominant relation $\preceq$. Here, $\cup_* = \cup$ and $\cap_* = \cap$. G is *heterogeneous* iff G is not homogeneous.

D includes a special *bottom* value $\perp$ which denotes that a process can take any value in D. D also includes λ which denotes that p_i does not decide which value p_i takes. Values in D which are neither $\perp$ nor λ are *proper*. $\top$ denotes the top of D if $x \preceq_* \top$ for every x in D, i.e. $\top$ means some value which satisfies the requirement of every process if $\top$ exists in D.

A proper value x is *minimal* in p_i iff there is no proper y such that $y \prec_i x$ in D. x is *minimum* in p_i iff $x \preceq_i y$ for every proper y in D. If p_i had taken the minimum x, p_i can take any proper value in D. For every proper x in D, x is *maximal* in p_i iff there is no proper y in D such that $x \prec_i y$. Once p_i takes a maximal x in D, p_i can take no value. x is *maximum* in p_i iff $y \prec_i x$ for every proper y in D. For example, D is $\{0, 1\}$ in the 2PC. A process taking 0 aborts independently of the others. A process taking 1 aborts if some process takes 0. Hence, $1 \preceq 0$. 1 is minimum and 0 is maximum.

A lattice-based domain is discussed by Yahata and Takizawa [14, 15].

3.2 Non-blocking condition

In the 2PC, the process taking 1 is *uncertain* [11] because the global decision may be 0. The uncertain process blocks if the coordinator is faulty before sending the decision to the processes and all the operational processes are uncertain. The uncertain process waits for the decision. We generalize the blocking concept.

[Definition] A process p_i is *uncertain* if p_i takes a value which is not maximal and does not make consensus yet. □

A process p_i taking the maximal value is *certain* since p_i cannot change the value. The uncertain

process may take another value.

Suppose that some p_j is faulty and another operational p_i is uncertain. Let v be $\cup_*\langle v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_n\rangle$. That is, every operational process can take v. If p_j is sure to be able to take v, all the operational processes make a consensus v. Otherwise, they have to wait for the recovery of p_j, i.e. p_i blocks. Even if all the operational processes cannot make consensus, they do not block if the NB condition is satisfied.

[Non-Blocking (NB) condition] The protocol is *non-blocking* if

(1) all the processes, whether or not operational, can take an upper bound of the values taken by all the operational processes, or

(2) all the operational processes can take values which satisfy the consensus condition. □

Unless the NB condition holds, every operational p_i blocks, i.e. cannot take a value following v_i.

4 Basic Commitment Protocol

4.1 Procedure

A *consensus protocol* coordinates the cooperation of processes $p_1, \ldots, p_n$ in a group G in order to make a consensus. We show the basic consensus protocol as follows.

[Basic protocol][Figure 1]

1 Each process p_i takes an initial value v_{i0}. p_i notifies all the processes in G of v_{i0}. $j := 0$.

2 p_i obtains $v_{1j}, \ldots, v_{nj}$. p_i makes a local decision $v_{i,j+1} = F_{ij}(v_{1j}, \ldots, v_{nj})$. F_{ij} is a *local decision* function. p_i notifies all the processes of $v_{i,j+1}$.

3 A global decision $v = GD(v_{1j}, \ldots, v_{nj})$ is obtained. GD is a *global decision* function. If the *termination* $M(v_{1j}, \ldots, v_{nj}, v)$ holds, p_i obtains v. Otherwise, $j := j + 1$ and go to step 2.

4 p_i obtains the global decision v, obtains $d_i = LD_i(v_{1j}, \ldots, v_{nj}, v)$. LD_i is a final local decision function. □

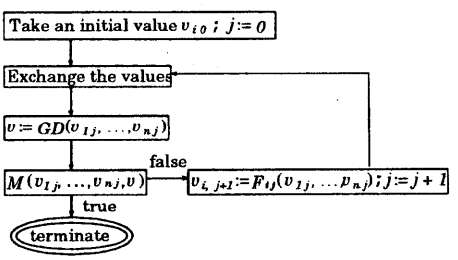

Figure 1: Basic protocol

Initially, each process p_i has v_{i0} in D. F_{ij} is a jth *local decision* function of p_i from D^n into D. For example, p_i notifies all the processes of λ if p_i has no idea. p_i obtains $v_{1j}, \ldots, v_{nj}$. If p_i obeys p_k's opinion, $v_{kj} = F_{ij}(v_{1j}, \ldots, v_{nj})$. Here, $v_{i,j+1}$ may not be different from v_{ij}. p_i notifies all the processes of $v_{i,j+1}$ obtained by F_{ij}.

The global decision $v = GD(v_{1j}, \ldots, v_{nj})$ is obtained by using a *global decision* function GD: $D^n \rightarrow D$. In the 2PC, $D = \{1, 0\}$. If all the processes take 1, they commit. If at least one process takes 0, all the processes abort. Hence, $GD(v_{1j}, \ldots, v_{nj}) = 1$ if $v_{ij} = 1$ for every p_i. $GD(v_{1j}, \ldots, v_{nj}) = 0$ if some $v_{ij} = 0$. If the global decision is a value taken by a majority of the processes, $GD(v_{1j}, \ldots, v_{nj}) = v$ if $|\{v_{ij} | v_{ij} = v\}| > \frac{n}{2}$.

M is the termination condition. After obtaining $v = GD(v_{1j}, \ldots, v_{nj})$, p_i can terminate the protocol if $M(v_{1j}, \ldots, v_{nj}, v)$ holds. For example, unless $v_{ij} \preceq_i v$, $M(v_{1j}, \ldots, v_{nj}, v)$ does not hold because p_i cannot take v. $M(v_{1j}, \ldots, v_{nj}, v)$ may not hold if $v = \lambda$, i.e. nothing is decided. If steps 2 and 3 are iterated more times than the specified number, the protocol terminates in order to avoid the indefinite computation. Unless terminated, step 2 can be executed again, that is, p_i notifies the other processes of $v_{i,j+1}$ $F_{ij}(v_{1j}, \ldots, v_{nj})$.

p_i obeys the global decision v if p_i is not autonomous. If p_i is autonomous, p_i may not obey v. p_i makes a final local decision by a *final local decision* function $LD_i : D^{n+1} \rightarrow D$. For example, if p_i makes the decision of v_i independently of v, $LD_i(v_1, \ldots, v_n, v) = v_i$.

4.2 Coordination schemes

There are two points on the coordination among the processes $p_1, \ldots, p_n$. The first point is which process makes a global consensus decision. In the centralized decision, every participant p_i obeys the decision made by the coordinator p_0. In the distributed decision, every p_i makes the consensus decision by itself. The other point is how to deliver values taken by the processes to the other processes in the group G. In the centralized delivery, every p_i first sends messages to p_0 and then p_0 forwards the messages to all the processes in G. On the other hand, every p_i sends messages to all the processes in the distributed delivery.

The protocol with the centralized decision and delivery is *centralized*. The 2PC [6] and 3PC [11] protocols are centralized. Here, every p_i first sends a value v_i to p_0. On receipt of $\langle v_1, \ldots, v_n\rangle$, p_0 decides a value $v = GD(v_1, \ldots, v_n)$. p_0 sends v to $p_1, \ldots, p_n$.

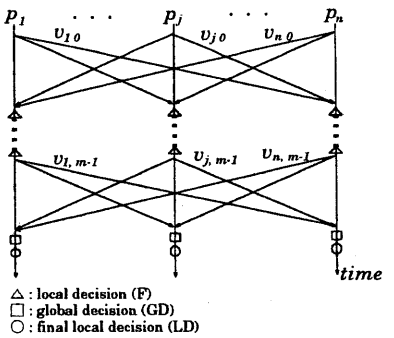

Figure 2: Distributed protocol

The distributed protocol adopts the distributed

decision and delivery. In the distributed protocol [Figure 2] [9, 12], each p_i sends v_i to $p_1, \ldots, p_n$. On receipt of the values, p_i makes the consensus decision by itself. If p_i cannot make the consensus, p_i obtains $v_i' = F_{ij}(v_1, \ldots, v_n)$ and sends v_i' to $p_1, \ldots, p_n$. Thus, every p_i has to send a message m to all the other processes.

5 Decision Logics

5.1 Local decision

For a tuple $\langle v_{1j}, \ldots, v_{nj} \rangle$ of values obtained at the jth round, each process p_i takes a value $v_{i,j+1}$ in D by the *local decision function* F_{ij}, i.e. $v_{i,j+1} = F_{ij}(v_{1j}, \ldots, v_{nj})$. Here, $v_{ij} \preceq_i v_{i,j+1}$. p_i has its own F_{ij} whose semantics depend on p_i at each jth round.

For some proper value v in D and every process p_i, $\langle v_{1j}, \ldots, v_{nj} \rangle$ is *agreeable* on v if $v_{ij} \preceq_i v$ for every p_i. That is, every process can take v. Unless $v_{ij} \preceq_i v$ holds for some p_i, p_i cannot take the same v as the others. In the homogeneous group, every p_i has the same dominant relation $\preceq$, i.e. $\preceq_i = \preceq$. Each p_i can estimate what value another process p_k takes. Suppose that p_i and p_j take x and y, respectively. If p_i takes a $v = x \cup y$, p_j also can take v. On the other hand, $\preceq_i \neq \preceq_j$ for some pair of p_i and p_j in the heterogeneous group. p_i cannot estimate the values to be taken by another p_k if $\preceq_i \neq \preceq_k$ and p_i does not know of $\preceq_j$. By recording the values taken by p_j, p_i can learn some part of $\preceq_j$. Here, $\preceq_{ik}$ is a subset of $\preceq_j$ which p_i knows. p_i receives a value v_j' at the next round after receiving v_j from p_j. $\pi_i(x)$ is a set $\{y \mid x \preceq_i y\}$ of values dominating x in p_i. $\Pi_i(x) = \{y \mid x \Rightarrow_i y\}$. Here, $\Pi_i(x) \subseteq \pi_i(x)$. Similarly, $\Rightarrow_{ik}$ is $\{ \langle x, y \rangle \mid p_i \text{ knows that } x \Rightarrow_k y \}$. On receipt of v_j', p_i can know that $v_j \preceq_j v_j'$ holds in p_j. Here, let $\pi_{ij}(x)$ be a set of values $\{y \mid x \preceq_{ij} y\}$ which p_i knows dominates x in p_j. Here, $\pi_{ii}(v) = \pi_i(v)$ and $\Pi_{ii} = \Pi_i$ for p_i. Let $\Pi_{ij}(x)$ be $\{y \mid x \Rightarrow_{ij} y\}$. Each time p_i obtains a tuple $\langle v_1', \ldots, v_n' \rangle$ of values after p_i has $\langle v_1, \ldots, v_n \rangle$, p_i includes v_j' in $\Pi_{ij}(v_j)$ and $\pi_{ij}(v_j)$ $(j = 1, \ldots, n)$.

We discuss which value p_i takes after obtaining $\langle v_1, \ldots, v_n \rangle$. Here, let π be $\pi_{i1}(v_1) \cap \cdots \cap \pi_{in}(v_n)$. If $\pi \neq \phi$, there is some value x reachable from v_j in every process p_j, i.e. $v_j \preceq_j x$ for every p_j. If $\pi = \phi$, p_i finds one value x in $\pi_{ii}(x)$ which is reachable from the most processes. Here, p_i takes x if $v_i \Rightarrow_i x$, i.e. p_i could take x just after taking v_i. Otherwise, p_i has to find such a value y that $v_i \Rightarrow_i y$ and $y \preceq_i x$.

Here, suppose that p_i obtains $\langle v_{1j}, \ldots, v_{nj} \rangle$ at the jth round. If $\cup_*\langle v_{1j}, \ldots, v_{nj} \rangle = \lambda$, p_i cannot take a value at the next step. Here, p_i sends $\Pi_i(v_{ij})$ so that every other process p_k can obtain $\Pi_{ki}(v_{ij}) = \Pi_i(v_{ij})$. p_i receives $\Pi_k(v_{kj})$ from p_k. Then, p_i tries to obtain $\cup_*\langle v_{1j}, \ldots, v_{nj} \rangle$ again by using $\Pi_{ki}(v_{kj})$ obtained here. If one value $v_{i,j+1}$ is obtained, p_i sends $v_{i,j+1}$. Otherwise, some process goes back to the previous value. In this paper, the processes p_i whose values are different from the majority of $v_{1j}, \ldots, v_{nj}$ are selected. p_i sends $\Pi_i(v_{i,j-1})$ to $p_1, \ldots, p_n$. On receipt of $\Pi_i(v_{i,j-1})$, p_k changes $\Pi_{ki}(v_{i,j-1})$ to $\Pi_i(v_{i,j-1})$. Then, p_k tries to find $\cup_*\langle v_{1j}, \ldots, v_{i-1,j}, v_{ij}, v_{i+1,j}, \ldots, v_{nj} \rangle$. If p_k cannot find $\cup_*$, another process p_k is selected to go backward to $v_{k,j-1}$. Even if every process goes backward to the $j-1$th round, the protocol terminates if $\cup_*$ could not be obtained.

5.2 Global decision

After obtaining the values $v_1, \ldots, v_n$, the global consensus value v is globally decided by using the global decision function GD. For every $\langle v_1, \ldots, v_n \rangle \in D^n$, $GD(v_1, \ldots, v_n)$ gives a value v in D. If $v_j \preceq_j v$ for every p_j, every process p_i can change the value v_i to v. Unless $v_i \preceq_i v$, p_i cannot change the value to v. The global decision function GD is *regular* if $v_i \preceq_i GD(v_1, \ldots, v_n)$ for every $\langle v_1, \ldots, v_n \rangle \in D^n$ and every p_i. If GD is regular, every p_i can change v_i to a value $v = GD(v_1, \ldots, v_n)$. Otherwise, p_i cannot obey the global decision v unless $v_i \preceq_i v$. For example, a process aborting cannot obey the global decision if the global decision is 1 (*commit*) in the 2PC protocol.

There are the following kinds of global decisions:

1 Binary 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 $D = \{1, 0\}$.
2 m-ary commitment decision : $GD(v_1, \ldots, v_n) = v$ if $v_i = v$ for every p_i, otherwise $\cup_*\langle v_1, \ldots, v_n \rangle$ where $D = \{x_1, \ldots, x_m\} (m \geq 2)$.
3 Majority-consensus decision on a value v: $GD(v_1, \ldots, v_n) = v$ if $|\{v_i | v_i = v\}| > \frac{n}{2}$, otherwise $GD(v_1, \ldots, v_n) = \lambda$.
4 $\binom{n}{n}$-decision on v: $GD(v_1, \ldots, v_n) = v$ if every $v_i = v$, otherwise $GD(v_1, \ldots, v_n) = \lambda$.
5 $\binom{n}{r}$-decision on v: $GD(v_1, \ldots, v_n) = v$ if $|\{v_i | v_i = v\}| \geq r$, otherwise $GD(v_1, \ldots, v_n) = \lambda$.
6 Minimal-decision: $GD(v_1, \ldots, v_n) = \cup_*\langle v_1, \ldots, v_n \rangle$.
7 Super-vote: $GD(v_1, \ldots, v_n) = v_i$ if p_i has the highest priority.

GD can be defined based on the application semantics. For example, if D is a set of numbers, some value is computed from $v_1, \ldots, v_n$ in D. An average of $v_1, \ldots, v_n$ is computed by GD.

In the centralized protocol, one process makes the global decision GD while every process makes GD in the distributed protocol.

6 M-ary Commitment Protocol

We discuss the m-ary commitment protocol by extending the binary commitment protocol.

6.1 Binary commitment protocol

First, the 2PC protocol is described in terms of the domain D, the decision logics, and the dominant relation. In the commitment protocol, suppose that 1 means *commit* and 0 means *abort*. Hence, the domain includes two values, i.e. $D = \{0, 1\}$.

The protocol terminates at the 2nd round. At the 1st round, every process p_i takes an initial value $v_{i0} \in D$ and sends v_{i0} to the coordinator p_0. Then, p_0 takes v_{01} but p_i takes the same value v_{i1} as v_{i0} at the 2nd round. Here, $v_{00} \preceq v_{01}$ but v_{i0}

$= v_{i1} \preceq v_{01} (i \geq 1)$. That is, only p_0 can change the value but no participant can. In the centralized protocol, v_{01} is the global decision. All the processes taking 0 *abort* unilaterally, i.e. without waiting for the global decision. On the other hand, processes taking 1 may commit or abort up to the global decision. Hence, 0 dominates 1, i.e. $1 \preceq 0$. 0 is maximum and 1 is minimum.

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

The final local decision is $LD_i(v_0, v_1, \ldots, v_n, v) = v$ because the process taking 1 obeys the global decision. That is, the processes taking 1 are not autonomous.

6.2 Centralized protocol

We extend the binary commitment protocol so that each process p_i can take more than two values, i.e. $v_1, \ldots, v_m (m \geq 2)$, λ, and $\perp$. That is, $D = \{v_1, \cdots v_m, \lambda, \perp\}$. In the 2PC protocol, p_i may not be able to send the value even if p_i receives *Prepare* from the coordinator p_0, e.g. p_i is too heavy-loaded to take a value. In such a case, p_i can send λ instead of sending the proper value, or p_i can be considered to take λ if no reply of *Prepare* is received in some time units. The global decision GD is the m-ary commitment one. We assume that the group G is homogeneous, i.e. $\preceq_i = \preceq$ and $\cup_* = \cup$.

First, we present the centralized protocol where there is one coordinator process p_0 and participant processes $p_1, \ldots, p_n$.

[Basic centralized protocol]

1 First, p_0 takes a value v_0 and sends $\langle v_0, \lambda, \ldots, \lambda \rangle$ to all the processes $p_1, \ldots, p_n$. $j := 0$.

2 On receipt of $\langle v_0, v_1, \ldots, v_n \rangle$ from p_0, each p_i takes one value v_i in D and sends v_i to p_0. In addition, p_i may send λ to p_0 if p_i could not decide whatever to take. p_i may send $\perp$ to p_0 if p_i could take any value in D. $j := j + 1$.

3 p_0 obtains $\langle v_1, \ldots, v_n \rangle$ where each v_i is obtained from p_i at step 2.

- If $v_0 = v_1 = \cdots = v_n (= v)$, p_0 makes the global decision v and sends v to $p_1, \ldots, p_n$.
- If $v_i = \cup_* \langle v_0, v_1, \ldots, v_i, \ldots, v_n \rangle \neq \lambda$ for some i, p_0 makes the global decision v_i and sends v_i to $p_1, \ldots, p_n$
- If $v'_0 = \cup_* \langle v_0, v_1, \ldots, v_n \rangle \neq \lambda$, p_0 takes v'_0 where $v_0 \preceq v'_0$ and $v_i \preceq v'_0$ for every j. Then, p_0 sends $\langle v'_0, v_1, \ldots, v_n \rangle$ to $p_1, \ldots, p_n$ and go to 2.
- If $\cup_* \langle v_0, v_1, \ldots, v_n \rangle = \lambda$, p_0 terminates and sends λ to $p_1, \ldots, p_n$ at the global decision.

4 On receipt of the global decision v from p_0, p_i decides if p_i takes v by the final local decision LD_i. If v satisfies LD_i, p_i takes v. Otherwise, p_i takes v_i. □

From $\langle v_0, v_1, \ldots, v_n \rangle$ at step 3, $v_i \preceq v_0$ for every p_i. On receipt of $\langle v_0, v_1, \ldots, v_n \rangle$, p_i takes a value v'_i by the local decision function F_{ij}. Here, $v_i \preceq v'_i$ and $v_0 \preceq v'_i$. Here, if v'_i is maximum, p_i makes the final decision v'_i and terminates after sending v'_i. Otherwise, p_i has to wait for the next decision value from p_0. Here, p_i is *uncertain*. At step 3, p_0 makes a consensus decision v if $v = v'_0 = v_i$ for every p_i. p_0 sends v to all the processes and terminates. This decision corresponds to the *commit* in the 2PC. On receipt of v, each p_i takes v as the final consensus value. If $v_i \neq v_j$ for some p_i and p_j, p_0 takes a new value v'_0 after receiving $v_1, \ldots, v_n$. p_0 finds the least upper bound v of $\langle v'_0, v_1, \ldots, v_n \rangle$. If some p_i takes $v (\neq \lambda)$, i.e. $v_i = v$, p_0 makes a consensus decision v. p_0 sends v to $p_1, \ldots, p_n$. It corresponds to the *abort* in the 2PC. If $v \neq v_i$ for every p_i, p_0 sends $\langle v, v_1, \ldots, v_n \rangle$ to $p_1, \ldots, p_n$ and step 2 is iterated.

Suppose that p_0 is faulty at the jth round after each p_i sends v_i before sending the reply v'_0 to all the processes. If p_i receives no reply in some predetermined time units, p_i detects that p_0 is faulty and invokes the following termination protocol to make consensus among the operational processes. Here, suppose that p_i takes a new value v'_i after receiving $\langle v_0, v_1, \ldots, v_n \rangle$.

[Termination protocol]

1 p_i sends *StateReq* with v'_i to all the processes.

2 On receipt of *StateReq* from p_i, p_k sends the local state to p_i. If p_k receives no reply from p_0, p_k sends the value v'_k. If p_k had received some reply v'_0 from p_0, p_k sends the reply $\langle v_0, v_1, \ldots, v_n \rangle$ back to p_i.

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

[Termination rule]

1 If p_i receives a maximal value v from some process, p_i takes v as the global decision.

2 If some operational p_i still takes λ, p_i makes a consensus decision of the maximal value. Otherwise, p_i waits for the recovery of p_0. □

Next, suppose that p_i recovers from the fault. Suppose that p_i records the local state in the log L_i. p_i invokes the following recovery protocol.

[Recovery protocol]

1 p_i restores the state from L_i.

2 If p_i is uncertain, p_i asks other processes in the same way as the termination protocol.

3 If p_i had taken λ or the maximal value, p_i makes the consensus decision on the maximum value. □

6.3 Distributed protocol

We discuss the distributed m-ary commitment protocol. Each process makes the global consensus decision on receipt of the value from the other processes.

[Basic distributed protocol]

1 Each process p_i takes a value v_i and sends v_i to all the processes $p_1, \ldots, p_n$. $j := 0$.

2 p_i receives $\langle v_1, \ldots, v_n \rangle$ from $p_1, \ldots, p_n$.

- If $v_1 = \ldots = v_n (= v)$, p_i makes the global consensus decision v and then terminates.
- If $v_i = \cup_* \langle v_1, \ldots, v_n \rangle (\neq \lambda)$ for some v_i, p_i makes the decision v_i and then terminates.

- If $v_i' = \cup_*\langle v_1, \ldots, v_n\rangle$ $(\neq \lambda)$, p_i sends v_i' to all the processes. $j := j + 1$ and go to 2.
- If $\cup_*\langle v_1, \ldots, v_n\rangle = \lambda$, p_i sends λ to all the processes and then terminates. □

Here, suppose that some process p_k is faulty after the jth round. Each operational process p_i has a tuple $\langle v_1, \ldots, v_n\rangle$ obtained at the jth round. If a global decision v from $\langle v_1, \ldots, v_n\rangle$ satisfying the termination condition M is obtained, every operational process takes v. Otherwise, every operational process blocks.

7 Concluding Remarks

This paper has discussed the general framework of consensus protocols where the values to be taken by the processes are partially ordered by the dominant relation. The dominant relation $\preceq_i$ is defined for each process p_i. p_i decides which value v_i' p_i takes after taking v_i so that $v_i \preceq_i v_i'$. We have presented kinds of the decision logics based on the dominant relation. We have also presented the m-ary commitment protocol obtained by extending the 2PC protocol, where each process takes m (≥ 2) kinds of values.

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*, 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 Byzantine Generals Problem," *ACM Trans. Programming Languages and Systems*, Vol.4, No3,1982, pp.382-401.

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

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

[11] 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.

[12] Tachikawa, T. and Takizawa, M., "Selective Total Ordering Broadcast Protocol," *Proc. of IEEE ICNP-94*, 1994, pp.212–219.

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

[14] Yahata, C., Sakai, J., and Takizawa, M., "Generalization of Consensus Protocols," *Proc. of the 9th IEEE Int'l Conf. on Information Networking (ICOIN-9)*, 1994, pp.419-424.

[15] Yahata, C. and Takizawa, M., "General Protocol for Consensus in Distributed Systems", *Proc. of DEXA(Lecture Notes in Computer Science*, No. 978, Springer-verlag), 1995, pp.227-236.