An Asynchronous [(n-1)/3]-Resilient Consensus Protocol*

Gabriel Bracha

TR 84-590
January 1984

Department of Computer Science
Cornell University
Ithaca, New York 14853

*Partial support for this work was provided by the National Science Foundation under grant No. 83-03135.

An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient Consensus protocol

Gabriel Bracha

Cornell University
Ithaca, New York 14853

ABSTRACT

A consensus protocol enables a system of n asynchronous processes, some of them malicious, to reach agreement. No assumptions are made on the behaviour of the processes and the message system; both are capable of colluding to prevent the correct processes from reaching decision. A protocol is t-resilient if in the presence of up to t malicious processes it reaches agreement with probability 1. In a recent paper, t-resilient consensus protocols were presented for $t < n/5$. We improve this to $t < n/3$, thus matching the lower bound on the number of correct processes necessary for consensus. The protocol restricts the behaviour of the malicious processes to that of merely fail-stop processes, which makes it interesting in other contexts.

1. Introduction

A consensus protocol enables processes, some of them faulty, to agree on a common value. In [Fisc82] it was shown that in asynchronous systems there is no finite agreement protocol that can tolerate even one fail-stop process. However, probabilistic [Brac83] and randomized [BenO83] solutions are possible for both the fail-stop and the malicious cases. In [BenO83] it was left as an open question whether there is an asynchronous consensus algorithm that can tolerate t malicious processes, for $n/5 \leq t < n/3$. In this paper we present a random algorithm for $t < n/3$, thus matching the lower bound on the number of correct processes necessary for any consensus protocol.

The expected number of steps required to reach decision is 2^{n-t}. However, if $t \leq c \cdot \sqrt{n}$, then the expected number of steps is constant (though exponential in c).

For the protocol, we develop schemes that render the malicious processes virtually fail-stop. Thus, these schemes enable us to use malicious processes in other protocols designed for fail-stop processes.

2. The consensus agreement problem

We consider an asynchronous system of n fully interconnected processes. Processes communicate by sending *messages* via the *message system*. The message system supports the *Send* and *Receive* primitives. Messages sent can take arbitrarily long time until received, and they are not necessarily received in the order sent.

In an *atomic step* of the system, a process can try to receive a message, perform some local computation, and then send a finite set of messages. The computation and the messages sent are prescribed by the *protocol*, i.e., a function of the messages received and of the local state. The protocol may be random; at a given state a process can choose with certain probabilities what step to take next.

A *correct* process always follows the protocol. A *fail-stop* process may fail and stop participating in the protocol; other processes cannot detect its failure. *Malicious* processes, besides failing to send messages, can send wrong and conflicting messages. The protocol has to withstand any behaviour of the malicious processes, so we assume that the malicious processes can collude against the correct ones according to some malevolent design. However, processes can identify the origin of every message that they receive. Otherwise, one malicious process can impersonate the whole system and no solution ispossible.

Each process p has an initial value i_p in $\{0,1\}$. The protocol terminates when each correct process has made an irreversible *decision* on some value.

A *configuration* of the system is the collection of processes' states and the messages in transit. The *initial configuration* of the system is the collection of the processes' i_p's at the beginning of the protocol.

A *schedule* is an alternating sequence $C_1 e_1, \ldots, C_k e_k C_{k+1}$ of steps and configurations such that C_{i+1} is obtained by applying e_i to C_i. The schedule determines which message is received by which process, and which process takes the next step. We make no assumption about the schedule. Thus, we have to consider the schedule as capable of arranging the steps of the protocol to the worst possible effect. The random steps taken by the processes introduce probabilities on the schedule space.

A *t-resilient consensus protocol* is a protocol that satisfies the following properties, provided that no more than t processes are faulty:

bivalence: if all the correct processes start with the same value v then they all decide on v.

consistency: all the correct processes decide on the same value.

convergence: for any initial configuration,

$$\lim_{k \to \infty} \text{Probability [a correct process has not decided within } k \text{ steps]} = 0.$$

3. Reliable broadcast - Asynchronous Byzantine agreement

In this section we present a broadcasting protocol which will be used in the consensus protocol as a primitive. In a broadcast protocol some process p sends a message containing its value to all the other processes. The protocol achieves Asynchronous Byzantine agreement [Brac83] if the following conditions hold:

1. If p is correct, then all the correct processes accept the value of its message.

2. If p is malicious, then either all the correct processes accept the same value, or none of them will accept any value from p.

3.1. The protocol

All the messages that are sent in the k'th broadcast by p are tagged with (p,k), thus eliminating possible interference between broadcasts. There are three types of messages in the protocol: *initial*, *echo* and *ready*. The algorithm starts with the transmitter p sending $(initial, v)$ messages, where v is the value of p. Then processes report to each other the value they received via $(echo, v)$ messages. If more than $(n+t)/2$ $(echo, v)$ messages are received by a process, then it sends $(ready, v)$ messages to all the other processes. Also, if a process receives $t+1$ $(ready, v)$ messages then it sends its own $(ready, v)$ message. If it receives $2t+1$ $(ready, v)$ messages with the same value v, then it accepts v.

Broadcast(v)

step 0. (Only by the transmitter)

  Send $(initial, v)$ to all the processes.

step 1. Wait till *Receive* for some v,

    one $(initial, v)$ message

    or $(n+t)/2$ $(echo, v)$ messages

    or $(t+1)$ $(ready, v)$ messages.

  Send $(echo, v)$ to all the processes.

step 2. Wait till *Receive* for some v,

    $(n+t)/2$ $(echo, v)$ messages

    or $t+1$ $(ready, v)$ messages

  Send $(ready, v)$ to all the processes.

step 3. Wait till *Receive* for some v, $2t+1$ $(ready, v)$.

  Accept v.

Figure 1. An Asynchronous Byzantine agreement protocol for $t < n/3$

3.2. Correctness proof

In this section we show that the protocol in Fig 1. achieves Asynchronous Byzantine agreement for $0 \leq t < n/3$.

Lemma 1. If two correct processes r and s send $(ready, v)$ and $(ready, u)$ messages respectively, then $u = v$.

Proof: Suppose not; r can send a $(ready, v)$ message if it receives more than $(n + t)/2$ $(echo, v)$ messages, or if it receives more than $t + 1$ $(ready, v)$ messages, i.e., it receives a $(ready, v)$ message from some other correct process. Therefore, there is some correct process p (which may be r) that received more than $(n+t)/2$ $(echo, v)$ messages. Similarlly, there is a correct process q that received more than $(n+t)/2$ $(echo, u)$ messages, where $u \neq v$. Therefore, some correct process r must have sent both $(echo, u)$ and $(echo, v)$ messages. But correct processes can send only one message of each type, during a broadcast, and hence a contradiction. □

Lemma 2. If two correct processes p and q accept the values v and u respectively, then $u = v$.

Proof: In order for p to accept v it must have seen $2t+1$ $(ready, v)$ messages, and therefore at least $t + 1$ $(ready, v)$ messages from correct processes. Similarlly, q must have seen at least $t + 1$ $(ready, u)$ messages from correct processes. By lemma 1, $u = v$. □

Lemma 3. If a correct process p accepts the value v then every other correct process will eventually accept v.

Proof: If p accepts v, then p received $2t+1$ $(ready, v)$ messages. At least $t+1$ of these messages were sent by correct processes. Therefore, every other correct process receives at least $t + 1$ $(ready, v)$ messages, and sends its own $(ready, v)$ message. Note that, by lemma 1, it is impossible for a correct process to send a different *ready* message. Thus, at least $n-t$ processes will send $(ready, v)$ messages. Every correct process will eventually receive at least $2t + 1 \leq n-t$ $(ready, v)$ messages, and will accept v. □

Lemma 4. If the transmitter p is correct and it sends v, then all the correct processes will accept v.

Proof: Suppose p is correct and sends v; every other correct process will receive an $(initial, v)$ message and will send an $(echo, v)$ messages. Consider a correct process q, q will receive $n-t > (n+t)/2$ $(echo, v)$ messages from the correct processes, and possibly

$t < (n+t)/2$ different messages from the malicious ones. Therefore, every correct process will send a $(ready, v)$ messages. In step 3, q will receive $n-t \geq 2t+1$ $(ready, v)$ messages, and possibly t different *ready* messages from the malicious ones. Therefore q will accept v. □

4. Correctness enforcement

In the previous section we restricted the behaviour of the malicious processes by forcing them to "virtually" send the same message to all the processes or no message at all. However, we could not control the content of the message. In this section we present a scheme that forces the malicious processes to conform with the underlying protocol.

We call any message that gets accepted in the k'th broadcast of some process a *k-message*. In particular, the k-message from p is denoted as m_p^k. By lemma 2, m_p^k is well defined. Consider the following general form of the k'th *step of the protocol*.

step(k)

Send v to all the processes.

wait till *Receive* a set S of $n-t$ k-messages

$v := N(k, S)$

N is the *protocol function* that determines the new value of the process according to the step number and S.

The following scheme will allow correct processes to consider only messages that are valid, i.e, messages that could have been sent by correct processes at that step. Thus, malicious processes have to behave correctly, otherwise they are ignored.

Each process p maintains the following set of messages $VALID_p$:

$VALID_p^1 =$ { accepted 1-messages }

For $k > 1$, $m_q^k \in VALID_p^k$, if there exist $n-t$ $(k-1)$-messages $m_1, \ldots, m_{n-t} \in VALID_p^{k-1}$ such that $m_q^k = N(k, \{m_1, \ldots, m_{n-t}\})$.

Process p *Validates* a message m if $m \in VALID_p^k$.

The basic step form is modified to the following:

step(k)

Broadcast(v)

wait till *Validate* a set S of $n-t$ k-messages

$v := N(k,S)$

In the consensus protocol we use *Validate* and *Broadcast* as the communication primitive. We now show that they have the same properties as the *Accept* and *Broadcast* primitives.

lemma 5. Let p and q be correct processes. If p validates a k-message from r with value v and q validates a k-message from r with value u, then $u = v$.

Proof: In order for p (q) to validate a k-message with value v (u), it must accept it. By lemma 2, $u = v$. □

Lemma 6. Let p and q be correct processes. If $m \in VALID_p^k$, then eventually $m \in VALID_q^k$ will hold.

Proof: The proof is by induction on k. If $k = 1$ then by lemma 3, if m is accepted by p then m will be accepted by q, and we are done. Let assume that the statement of the lemma holds for some $k \geq 1$. Let $m \in VALID_p^{k+1}$. Therefore, there are $n-t$ messages $m_1, \ldots, m_{n-t} \in VALID_p^k$ such that $m = N(k+1,\{ m_1, \ldots, m_{n-t} \})$. By our induction hypothesis, each of these messages joins $VALID_q^k$. By lemma 3, m will be eventually accepted by q. Therefore m joins $VALID_q^{k+1}$. □

Lemma 7. If p is a correct process, then eventually $m_p^k \in VALID_q^k$ for every correct process q.

Proof: The proof is by induction on k. If $k = 1$, then by lemma 4 we are done. Suppose the statement of the lemma holds for step k. Since p is correct it can send m_p^{k+1} only if it validated $n-t$ k-messages, $m_1, \ldots, m_{n-t}$, such that $m_p^{k+1} = N(k+1,\{m_1, \ldots, m_{n-t}\})$. By lemma 6, for every correct process q and for each m_i, $1 \leq i \leq n-t$, eventually $m_i \in VALID_q^k$. Also, since p is correct, by lemma 4, every other correct process q will accept m_p^{k+1}. Therefore, eventually $m_p^{k+1} \in VALID_q^{k+1}$ for every correct process q. □

Through the validation mechanism the malicious processes are reduced to mere fail-stop processes. We will not provide a formal proof of that in this paper. The only remaining aspect of their malice is that, whenever there is a random step to perform, their choice is not subject to probability.

5. The consensus algorithm

The protocol is conducted in rounds which are repeatedly executed by all the processes. Each round consists of three steps. For notational convenience, the protocol in Figure 2 does not terminate once decision is made. However, this can be easily accomplished by making the processes that have decided send some special halting message.

Round(k): (by process p)

1. *Broadcast*(i_p), and wait till *Validate* † $n-t$ messages.

   $i_p :=$ majority value of the validated messages.

2. *Broadcast*(i_p), and wait till *Validate* $n-t$ messages.

   If more than $n/2$ of the messages have the same value v, then $i_p :=(d,v)$

3. *Broadcast*(i_p), and wait till *Validate* $n-t$ messages.

   If validated more than $2t+1$ (d,v) messages then *Decide* v.

   If validated more than $t+1$ (d,v) messages then $i_p := v$.

   Otherwise, $i_p :=$ 1 or 0 with probability ½ .

   Go to step 1 of round $i+1$

figure 2. The consensus protocol.

6. Correctness proof

We prove in this section that the protocol in Figure 2 is a t-resilient consensus protocol, for $t < n/3$.

Lemma 8. If in some round r, at the beginning of step 1, all the correct processes have the same value v, then they all decide v at step 3 of round r.

Proof: Since all $n-t$ correct processes v as their value, every correct process will validate at least $n-2t$ messages with value v at step 1. Since $n-2t > (n-t)/2$ for $t < n/3$, each correct process retains v as its value at step 1. At step 2, in order to validate a message with value $u \neq v$, a correct process must also validate more than $(n-t)/2$ u messages from step 1. Since $(n-t)/2 > t$, this is clearly impossible. Therefore, the only possible value validated in step 2 is v, and all the correct processes change their value to (d,v). At step 3, all the correct processes validate more than $2t+1$ (d,v) messages, and

† The validation is done with respect to N, the protocol function described in Figure 2.

they decide v. □

Since the protocol requires processes to wait for each other, we must show that it does not deadlock.

Lemma 9. If a correct process p is at step k, then p will eventually progress to step $k + 1$.

Proof: Suppose not; then some correct processes are forever blocked. Let k be the smallest step number at which some process p is blocked, waiting for validated messages. Since k is minimal, all the correct processes have already broadcasted messages at step k. By lemma 7, all these messages get eventually validated. Therefore, p is not blocked at step k, a contradiction. □

Define D_r to be v, if $t + 1$ (d,v) messages were validated by some correct process at round r. Otherwise, $D_r = \phi$.

Lemma 10. D_r is well defined.

Proof: Suppose not; then some correct processes p and q have validated $(d,1)$ and $(d,0)$ messages respectively. If p validates a $(d,1)$ message it must also validate more than $n/2$ messages with value 1 that were sent at the step 2 of round r. Similarlly, q must have validated more than $n/2$ messages with value 0 that were sent at step 2 of round r. By lemma 5, this is impossible, and hence a contradiction. □

Theorem 1. The protocol described in Figure 2 is a t-resilient consensus protocol, for $t < n/3$.

Proof:

If all the processes start with value 1 (0) then by lemma 8, they all decide on 1 (0).

Suppose, without loss of generality, that a correct process p decides 1 at step 3 of round r. p must have validated at least $2t + 1$ $(d,1)$ messages. By lemma 6, every other correct process q validates at least $t + 1$ of these messages. Since, by lemma 10, D_r is well defined, every correct process sets its value to 1. At the beginning of round $r + 1$ all the correct processes have value 1. By lemma 8, at the end of round $r + 1$, all the correct processes decide 1.

Consider the time the first process p reaches the end of round r and sets i_p. The only thing the schedule can do to affect the processes' values in the next round is to establish a value for D_r, and then try to force the processes initial values at round $r+1$ to D_r. There are two cases:

1. p has validated a (d,v) message. Therefore, by lemma 10, the only potential value of D_r is v. With probability $\rho \geq 2^{-(n-t)}$, all the correct processes will set their value to D_r.

2. p has not validated a (d,v) message. Every other correct process can validate only up to t (d,v) messages, for any v. Therefore, D_r cannot have any value. The schedule has no way of forcing the processes' values. Again, with probability $\rho \geq 2^{-(n-t)}$, all the correct processes will set their values to the same value.

In either case, by lemma 8, they all decide in the next round. The probability of not terminating is $\lim_{k \to \infty} (1-\rho)^k = 0$. □

For $t \geq n/3$, t-resilient consensus protocol are impossible [Brac83]. Thus, the protocol is optimal in the number of malicious processes it can tolerate.

7. Performance

Since at each phase we have a probability $\rho \geq 2^{-(n-t)}$ of deciding in the next phase, the expected number of phases to decision is 2^{n-t}. However, if $t \leq c \cdot \sqrt{n}$, then the expected number of phases is only a constant [BenO83] (though exponential in c).

8. Reference

BenO83 M. Ben-Or, Another advantage of free choice: Completely asynchronous agreement protocol, *Proc. 2nd Symposium on the Principles of Distributed Systems, pp. 27-30.*

Brac83 G. Bracha and S. Toueg Asynchronous consensus and Byzantine protocol in faulty environment *TR-83-559, CS Dept., Cornell University, Ithaca, NY 14853.*

Fisc82 M. J. Fischer, N. A. Lynch, and M. S. Paterson, Impossibility of distributed consensus with one faulty process, *Proc. 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database systems.*