Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols

(Extended Abstract)

Michael Ben-Or †

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

1. Introduction

Recently, Fischer, Lynch and Paterson [3] proved that no completely asynchronous consensus protocol can tolerate even a single unannounced process death. We exhibit here a probabilistic solution for this problem, which guarantees that as long as a majority of the processes continues to operate, a decision will be made (Theorem 1). Our solution is completely asynchronous and is rather strong: As in [4], it is guaranteed to work with probability 1 even against an adversary scheduler who knows all about the system.

We apply the same ideas to the "Byzantine" type of failure. Here, if the number of faulty processes, t, satisfies $5t < N$, where N is the total number of the processes, then completely asynchronous agreement is possible (Theorem 2).

Our protocols provide the first example of a synchronization problem that has a probabilistic solution which is always guaranteed to work, but cannot be solved at all by any deterministic protocol. Previous examples required the processes to be symmetric.

The protocols presented here are not necessarily efficient. However, if the number of faulty processes, t, is $O(\sqrt{N})$, then when running the processes synchronously, the expected time to reach agreement is constant (Theorem 3). This result shows another advantage of probabilistic protocols, since any deterministic solution to the "Byzantine Generals" problem cannot reach agreement in less than $t+1$ rounds, (see [1,2]).

2. The Consensus Problem

A set of N asynchronous processes wish to agree about a binary value. Each process P starts with a binary input x_P, and they all must decide on a common value. The trivial solution, say, 0 is always chosen, is ruled out by the following correctness criterion:

(C1) If for all P, $x_P = v$, then the decision must be v.

A process "decides" by setting a "write-once" output register to be 0 or 1. Thus after deciding, a process may no longer change its decision.

To reach agreement processes communicate by means of messages. A *message* is a pair (P, m), where P is the name of the destination of the message and m is its content. The message system maintains a *message buffer* M that contains all the messages send but not yet delivered.

A process P can send the message m to process Q by performing *send*(Q, m). This operation adds the message (Q, m) to the message buffer. Process

† Research supported by a Weismann Postdoctoral fellowship and by NSF grant MCS-8006938.

P can attempt to receive a message by performing *receive*(P). This operation can delete some $(P, m) \in M$, in which case we say that (P, m) was delivered, or returns a special null message ϕ and leaves the buffer M unchanged.

Thus the message space acts nondeterministically, subject only to the condition that if *receive*(P) is performed infinitely many times, then every message (P, m) in the message buffer is eventually delivered.

A *configuration* of the system consists of the internal state of each process together with the contents of the message buffer. An *initial configuration* is one in which each process starts at an initial state and the message buffer is empty.

A *step* of a single process takes one configuration to another. In this primitive step process P first performs *receive*(P). This may be either a message m from the message buffer that was addressed to P, or the null message ϕ. Then, depending on P's internal state and on the value received, P performs some computation (including perhaps some probabilistic choices) ending in a new internal state, and sends a finite number of messages to other processes.

The processes are completely asynchronous, that is, we make no assumptions about their relative speed nor about the delay time in delivering a message. Thus a solution for this consensus problem must work correctly even against an adversary schedule. We allow such schedule to choose the next process P to make a step, and to control the message system. The schedule choice may depend on the current configuration as well as on all the past history of the computation.

Thus starting from an initial configuration, the schedule chooses the first process to make a step. This step may end in many different configurations. Once P made its step, some possible configuration has been reached. Knowing this, the schedule now chooses the next process to step and what his *receive* operation will return. This process completes his step leaving the system in some configuration, and so on, producing an infinite *run* of the system.

A schedule is *t-correct* if on any infinite run, at most t processes make a finite number of steps, and any message is eventually delivered if the receiving process makes an infinite number of steps. Thus the only failure allowed is a process death. It is clear, however, that other processes cannot determine whether a process has died or is just operating very slowly.

3. A Consensus Protocol

In this section we present a simple probabilistic consensus protocol. In this protocol the processes perform "rounds" of exchange of information. On each round, if some process decides v, then by the next round all the other operating processes will decide the same value v. If no process decides then with some bounded positive probability all the operating processes will reach agreement on the next round. The round number r is attached to the messages of round r, so the processes can distinguish between messages from different rounds.

A — Consensus Protocol

Process P: Initial value x_P.

step 0: set $r := 1$.

step 1: Send the message $(1, r, x_P)$ to all the processes.

step 2: Wait till $N - t$ messages of type $(1, r, *)$ are received. If more than $N/2$ messages have the same value v, then send the message $(2, r, v, D)$ to all processes. Else send the message $(2, r, ?)$ to all processes.

step 3: Wait till $N - t$ messages of type $(2, r, *)$ arrive.

(a) If there is one D-message $(2, r, v, D)$ then set $x_P := v$.

(b) If there are more than t D-messages, then decide v.

(c) Else set $x_P = 1$ or 0 each with probability $\frac{1}{2}$.

step 4: Set $r := r + 1$ and go to step 1.

Theorem 1: *Let $N > 2t$. For any t-correct schedule and any initial values of the processes, the above protocol guarantees, with probability 1, that:*
(i) all the processes will eventually decide on the same value v;
(ii) if all processes start with the value v, then within one round they will all decide v; and
(iii) if for some round r, some process decides v in step 3(b), then all other processes will decide v within the next round.

Remark: If $N \leq 2t$ then consensus is certainly impossible, since the schedule can then simulate a network partition.

4. Byzantine Agreement

Here faulty processes might go completely haywire, perhaps even sending messages according to some malevolent plan. The following completely distributed protocol can reach agreement even in the presence of such faults. We assume that a process can determine the originator of a message he has received. This is necessary since otherwise no solution is possible.

In this setting the schedule takes care for the message system, determines when each process will make a step, and determines what the faulty processes do. A schedule is t-correct if it allows at most t faulty process and eventually delivers all the messages to any correct process that makes an infinite number of steps.

B — Byzantine Protocol

Process P: Initial value x_P.

step 0: set $r := 1$.

step 1: Send the message $(1, r, x_P)$ to all the processes.

step 2: Wait till messages of type $(1, r, *)$ are received from $N - t$ processes. If more than $(N + t)/2$ messages have the same value v, then send the message $(2, r, v, D)$ to all processes. Else send the message $(2, r, ?)$ to all processes.

step 3: Wait till messages of type $(2, r, *)$ arrive from $N - t$ processes.

(a) If there are at least $t+1$ D-messages $(2, r, v, D)$, then set $x_P := v$.
(b) If there are more than $(N + t)/2$ D-messages then **decide** v.
(c) Else set $x_P = 1$ or 0 each with probability $\frac{1}{2}$.

step 4: Set $r := r + 1$ and go to step 1.

Theorem 2. *Let $N > 5t$. For any t-correct schedule and any initial values of the processes, the above protocol guarantees, with probability 1, that:*
(i) all the correct processes will eventually decide on the same value v;
(ii) if all correct processes start with the value v, then within one round they will all decide v; and
(iii) if for some round r, some correct process decides v in step 3(b), then all other correct processes will decide v within the next round.

Remark: We do not know whether $N > 5t$ is the best possible bound to reach distributed Byzantine agreement.

5. Efficiency

The protocols above are not very efficient, and in particular the expected number of rounds to reach agreement may be exponential. However if the number of faulty processes is $O(\sqrt{N})$ then the following theorem shows that the expected number of rounds to reach agreement is constant.

Theorem 3. *If $t = O(\sqrt{N})$ then the expected number of rounds to reach agreement in protocols* **A** *and* **B** *is constant, (i.e. does not depend on N).*

This last result is especially interesting since for deterministic protocols it is known that Byzantine agreement is impossible in less than $t + 1$ rounds of exchange of information [1,2].

Acknowledgment

The author would like to thank Nancy Lynch for many helpful discussions.

References

[1] Dolev, D. and Strong, R. Polynomial Algorithms for Byzantine Agreement. *Proc. 14th*

ACM Symp. on Theory of Computing (1982), 401–407.

[2] Fischer, M. and Lynch, N. A Lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters 14,* 4 (1982), 182–186.

[3] Fischer, M., Lynch, N. and Paterson, M. Impossibility of Distributed Consensus With One Faulty Process. MIT/LCS/TR-282.

[4] Lehman, D. and Rabin, M. On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem. *to appear.*