Randomized Protocols for Asynchronous Consensus

Alessandro Panconesi
DSI - La Sapienza
via Salaria 113, piano III
00198 Roma, Italy

One of the central problems in the Theory of (feasible) Computation is to ascertain to what extent, if any, randomization makes algorithms more powerful. This broad question is the motivation behind several very challenging open problems such as $\mathcal{P} \stackrel{?}{=} \mathcal{BPP}$– whether randomization expands the class of problems solvable in polynomial time– or $\mathcal{NC} \stackrel{?}{=} \mathcal{RNC}$– whether randomization helps to compute problems fast in parallel in the PRAM model. While these problems appear today to be as unfathomable as they were the first time that they were formulated a couple of decades ago, the question randomization is more powerful than determinism *can* be settled in the more restricted setting of distributed computation.

We saw in the last lecture that no 1-resilient protocol for consensus exists. If we use randomization however, consensus can be solved. In fact, we shall give an f-resilient protocol for $f < n/2$! This is an instance where randomization makes a computation model more powerful. As usual, we shall consider only crash failures.

1 A Randomized Protocol for Consensus

We describe a protocol which was originally proposed by Ben Or [3].[1] The protocol runs forever but every non-faulty process makes a decision in finitely many steps. With high probability this number of steps is huge. In spite of the fact that Ben Or's protocol is not practical the result is important because it indicates that by using randomization the impossibility of consensus can be circumvented. Indeed, after Ben Or's protocol the search for

[1]The protocol presented here is a simplified version of that given in the book of Nancy Lynch [4]. There it is claimed that the protocol needs the asumption $f < n/3$ but this is incorrect. As we shall see, $f < n/2$ suffices.

feasible randomized solutions for the consensus problem has been an active area of research. Recent solutions for the shared-memory model– where a set of n processes access asynchronously a common shared memory via read and write operations– require as few as $O(n \log^2 n)$ expected operations per process. Furthermore, these solutions are $(n-1)$–resilient (see, for instance, [1, 2]). In fact, by using randomization a protocol can cope even with Byzantine failures! (see, for example, [5]).

Let us now turn to Ben Or's protocol, which is illustrated in Figure 1. In the protocol v stands for any value different from $\perp$ and $a_p :=$ Ⓢ means that the value of a_p is set at random to be 0 or 1 with uniform probability. Although the protocol goes on forever the decision value of a process is unique– once a decision is made it cannot be changed. We can assume that the **decide** instruction has the property that once the first value is output, all subsequent invocations output the same value. The protocol is f-resilient as long as $f < n/2$ and assumes that channels are FIFO. It consists of an infinite repetition of "asynchronous rounds". During round r only messages which are timestamped with round number r are processed. The protocol consists of an endless broadcast of *a-values* and *b-values*. The initial a-value of a process is set to be equal to the input bit and can therefore be either 0 or 1. Thereafter, the a– and b–values can be in the set $\{0, 1, \perp\}$ and are determined as follows. During phase 1 of round r, each process waits for the first $n - f$ a-values with timestamp r and sets its own new b-value as a function of these. During the second phase of round r each process sends its own b-value timestamped with r to all, including oneself, and waits for the first $n - f$ b-values with timestamp r. Depending on these, it then sets its own new a-value and decides whether to commit to a final value. If the set of b-values does not satisfy a certain condition, the new a-value is set at random. Once the new a-value is determined it is sent to all (including oneself) with the new timestamp. And so on.

We now prove that the three requirements of Non Triviality, Agreement and Limited Dithering are satisfied.

We start with Non Triviality. Suppose that all input bits are 0. Then, all processes decide 0 at round 1. To see this just follow the protocol through round 1. Each process starts by sending 0 to all. Therefore every correct process will set the b-value to 0 and send it to all. Then, every process still running will receive 0's with timestamp 1 from $n - f$ distinct processes and hence will decide 0. The same reasoning holds for the case when all inputs are 1.

Let a_{pr} and b_{pr} denote the a-value and b-value of process p at round r, respectively. To establish Agreement and Limited Dithering we shall make

$a_p :=$ *input-bit*; $r := 1$;

repeat forever

phase 1
Send the message (a_p, r) to all. Let A be the multiset of the first $n-f$ a-values received with timestamp r. If A contains $n-f$ identical values equal to some v then set $b_p := v$. Otherwise set $b_p := \perp$.

phase 2
Send the message (b_p, r) to all. Let B be the multiset of the first $n-f$ b-values received with timestamp r.

1. If B contains $n-f$ identical values equal to some $v \neq \perp$ then decide v and set $a_p := v$.
2. Otherwise, if there exists $v \in B$, $v \neq \perp$, then set $a_p := v$.
3. Otherwise, set $a_p :=$ $.

Set $r := r+1$.

end-repeat

Figure 1: BenOr's protocol for process p

use of a simple but important observation.

Proposition 1.1. *For all r, either $b_{pr} \in \{1, \perp\}$ for all p or $b_{pr} \in \{0, \perp\}$ for all p.*

Proof. Suppose to the contrary that, at some round r, there are two processes p and q such that $b_{pr} = 0$ and $b_{qr} = 1$. From the b-value deciding rule of phase 1 it follows that p received 0's from $n-f$ distinct processes and q received 1's from $n-f$ distinct processes. This means that there are at least $2(n-f) \leq n$ processes in the system. But this is a contradiction since $n > 2f$.

To establish Agreement, suppose that at round r process p decides on some value v. Suppose also that round r is the first round where a decision is made. We want to show that every correct process decides v by round $r+1$. Suppose without loss of generality that p decides on 0. First, it is impossible that two processes p and q deciding at round r decide on different values. If this were not the case some b-values would be 0 and others would be 1, violating Proposition 1.1. Since p decides on 0 it has received 0's from $n-f$ distinct processes. This means that every other process at round r receives 0's from at least $n-2f \geq 1$ processes. By clause 2 of phase 2, every process still running will set its new a-value for round $r+1$ to 0 (including p). Therefore at the beginning of phase 1 of round $r+1$ every process sends 0 to all. At this point, the same reasoning used in the case of Non Triviality allows us to conclude that every process which has not already decided on 0 at round r will do so at round $r+1$.

It then remains to prove Limited Dithering. It is here that randomization enters into the picture.

In order to get an intuition of how randomization is used, suppose that the inputs are split in half; half of them being zeroes and half of them being ones. In this case, every process will set its b-value to $\perp$. Since all b-values are $\perp$, each process will set the next a-value at random. With high probability, about half of the new a-values will be zeroes and about half will be ones. Which again will force a random determination of all new a-values, and so on.

It is conceivable that this goes on indefinitely but this event has probability zero. How long can this go on? It follows from Proposition 1.1 that if $A_r = (a_{1r}, a_{2r}, \ldots, a_{nr})$ denotes the vector of the new a-values then, either A_r is of the kind

$$A_r = (v, \ldots, \text{\textcircled{\$}}, \ldots, \dagger, \ldots, \dagger, \ldots, v, \ldots, \dagger, \text{\textcircled{\$}})$$

or of the kind

$$A_r = (\bar{v}, \ldots, \dagger, \ldots, \dagger, \ldots, \text{\textcircled{\$}}, \ldots, \bar{v}, \ldots, \dagger, \text{\textcircled{\$}}).$$

where $a_{ir} = \text{\textcircled{\$}}$ means that the value is set at random and $a_{ir} = \dagger$ means that process i is dead (and hence it can be ignored). In other words, it is never the case that $a_{ir} = v$ and $a_{jr} = \bar{v}$. This implies that there is always a positive probability that all new proposal coincide, i.e. that either $a_{ir} = v$ for all i still running or $a_i = \bar{v}$ for all i still running. Since processes flip coins independently, this probability is at least 2^{-n}. When this happens we say that a *landslide* takes place. The argument for Non Triviality already seen shows that in case of a landslide every non faulty process will decide by the end of the next round. Now, for any round r

$$\Pr[\textit{no landslide at } r] \leq 1 - \frac{1}{2^n}.$$

Since the coin flips are independent, the probability of having no landslide for k consecutive rounds is then

$$\Pr[\textit{no landslide for first } k \textit{ rounds}] \leq \left(1 - \frac{1}{2^n}\right)^k.$$

This probability goes to zero as k goes to infinity. It takes however, exponentially many steps before the value is very small. The value is about $1/e$ when $k = 2^n$ which is exponential in n. So, if we run the protocol for $k = c2^n$ rounds then

$$\Pr[\textit{landslide within } k \textit{ rounds}] \geq 1 - \left(1 - \frac{1}{2^n}\right)^k \geq 1 - \frac{1}{e^c}.$$

This probability goes very quickly to 1 as c grows. A slightly more sophisticated analysis shows that an exponential number of rounds is, with high probability, both necessary and sufficient.

References

[1] J. Aspnes and M. Herlihy, Fast Randomized Consensus using Shared Memory, Journal of Algorithms 11 (3):441–461, Sep 1990

[2] J. Aspnes and O. Waarts, Randomized Consensus in Expected $O(N \log^2 N)$ Operations per Processors, SIAM J on Comp, 25(5):1024-1044, October 1996.

[3] M. Ben Or, Another advantage of free-choice: Completely asynchronous agreement protocols, Proceedings of the 2nd annual ACM symposium on Principles of Distributed Computing (PODC 1983) pp. 27-30, Montreal, Canada

[4] N. Lynch, Distributed Algorithms, Morgan Kaufmann, San Francisco

[5] R. Motwani and P. Raghavan, Randomized Algorithms, Cambridge University Press 1995