

RANDOMIZED CONSENSUS IN ASYNCHRONOUS SYSTEMS

These notes describe Ben-Or's algorithm for solving Consensus in asynchronous systems using randomization. This algorithm assumes that the majority of processes are correct (i.e., $n > 2t$, where t is the maximum number of processes that may fail), that failures are benign (crash, send/receive omissions), and that every process can toss a coin that gives values 0 or 1, each with probability $1/2$.

The proof given here also requires the following assumption. The *scheduler* decides which process takes the next step and which message is received during that step. We assume that the scheduler is *weak*, i.e., it cannot see the state of the processes, the content of messages, or the result of coin tosses.

$x_p :=$ initial value proposed by p

for $k := 1, 2, \dots$ **do**

send (R, k, x_p) **to** all processes

wait for messages of the form $(R, k, *)$ **from** $n - t$ processes (where $*$ can be 0 or 1)

if received more than $n/2$ (R, k, v) with the same v

then send (P, k, v) **to** all processes

else send $(P, k, ?)$ **to** all processes

wait for messages of the form $(P, k, *)$ **from** $n - t$ processes (where $*$ can be 0, 1 or ?)

if received at least $t + 1$ (P, k, v) with $v \neq ?$

then decide v

if received at least one (P, k, v) with $v \neq ?$

then $x_p := v$

else $x_p := 0$ or 1 , each with probability $1/2$

Each iteration of the repeat loop will be called a round (k is the round number). The messages sent are triples containing a tag (R or P), a round number, and a value which is either 0 or 1 (for messages tagged P , it could also be '?'). Messages tagged R will be called *reports*; those tagged with P will be called *proposals*. Accordingly when p sends (R, k, v) ((P, k, v)) we'll say p *reports* (*proposes*) v in round k . Each round contains two "phases"; one during which processes send their reports to each other ("report phase") and one during which they send their proposals to each other ("proposal phase"). Note that in each phase a process p waits for messages from $n - t$ processes. In the asynchronous model p shouldn't ever have to wait for messages from any more processes because then the (up to t) faulty processes could cause p to wait forever.

Lemma 2.1. *It is impossible for a process to propose 0 and another to propose 1 in the same round k .*

PROOF: If p proposes 0, then it received more than $n/2$ reports for 0; if q proposes 1, it received more than $n/2$ reports for 1. But then there is a process that reported 0 to p and 1 to q , contradicting that there are only benign failures. □

Lemma 2.2. *If, for some $v \in \{0, 1\}$, at the end of round $k - 1$ we have $x_p = v$ for every p which sends a report message in round k , then all correct processes will decide v in round k . (Where the values of the x_p 's in "round" 0 are the initial values of the processes.)*

PROOF: By assumption, every report message received in phase k will be for v . Since each process receives $n - t$ reports and $n - t > n/2$, it will propose v (if it proposes anything at all). Thus, all proposals sent in round k will be for v as well. Since at least all correct processes will send proposals in round k and there are at least $n - t > t$ of them, each correct process will receive at least $t + 1$ proposals for v in round k and will therefore decide v in round k . □

Lemma 2.3. *If any process decides v in round k then all correct processes will decide v in round $k + 1$.*

PROOF: If p decides v in round k , then it received at least $t + 1$ proposals for v in round k . Consider any correct process q . Process q receives $n - t$ proposals in round k , so it can "miss" at most t of the $t + 1$

proposals for v that p received. So q receives at least one proposal for v , and will therefore set $x_q := v$ in round k (note that, by Lemma 2.1, q couldn't have also received a proposal for \bar{v}). Thus, every q that sends a report in round $k + 1$ will have $x_q = v$ at the end of round k . By Lemma 2.2 then, every correct process will decide v in round $k + 1$. \square

Corollary. *If a correct process decides v in a round it will keep deciding v in all subsequent rounds.* \square

Theorem 2.1. *If $n > 2t$ Ben-Or's algorithm guarantees Uniform Agreement, Validity and, with probability 1, Termination[†] in the presence of up to t faulty processes that cause only benign failures.*

PROOF: Uniform Agreement follows from Lemma 2.3, and Validity from Lemma 2.2 (with $k = 1$). For Termination, consider the set S of processes that reset their estimate (variable x) at the end of round k . Partition S into two subsets S_d and S_r — those processes that reset their estimate x *deterministically* and those that do so *randomly*, i.e., by tossing a coin. More precisely, every process p in S_d sets x_p to some value v because it received a proposal for v (and by Lemma 2.1, this v is the same for all processes in S_d); on the other hand, every process p in S_r sets x_p by choosing 0 or 1, each with probability $1/2$.

Since the scheduler is weak, the value v of the processes in S_d is independent of the coin tosses of the processes in S_r in round k . So with probability at least 2^{-n} all the processes in S_r also set their estimate to v . Thus, with probability at least 2^{-n} all the processes in S set their estimate to the same value at the end of round k , and, by Lemma 2.2, all correct processes decide in round $k + 1$. Therefore, in each round k , the probability that all processes will decide in round $k + 1$ is at least 2^{-n} . So, Termination is achieved with probability 1. (But the expected number of rounds for this to happen is $\Omega(2^n)$.) \square

Remark. *The algorithm does not, in general, satisfy Strong Validity. It does, if $n > 4t$.* (The proof of these assertions is left as an exercise.) \square

Necessity of $n > 2t$ in Theorem 2.1

We shall show that the requirement $n > 2t$ in Theorem 2.1 is actually necessary.

Theorem 2.2. *If $n \leq 2t$, there is no algorithm that guarantees Agreement, Validity and Termination in an asynchronous system, even if only crash failures can occur.*

PROOF SKETCH: Suppose such an algorithm exists. Let P_0, P_1 be a partition of the set of n processes so that each of P_0, P_1 has at most t processes. We arrive at a contradiction by considering the following 3 scenarios (i.e., possible executions of the algorithm):

Scenario 1: All processes have initial values 0. All processes in P_0 are correct, and all processes in P_1 crash at the start (note that there are no more than t faulty processes). Since the algorithm satisfies Termination and Validity, it has a run r_0 such that all processes in P_0 decide, and they decide 0. Let t_0 be the time by which all processes in P_0 decide 0 (relative from the start of r_0).

Scenario 2: (Symmetric to Scenario 1) All processes have initial values 1. All processes in P_1 are correct, and all processes in P_0 crash at the start (note that there are no more than t faulty processes). Since the algorithm satisfies Termination and Validity, it has a run r_1 such that all processes p_1 in P_1 decide, and they decide 1. Let t_1 be the time by which all processes in P_1 decide 1 (relative from the start of r_1).

Scenario 3: All processes in P_0 have initial values 0, all processes in P_1 have initial values 1. All processes are correct. Messages sent from P_0 to P_1 and vice versa are delayed to take longer than $t = \max\{t_0, t_1\}$ (we can do this because the system is asynchronous). The algorithm has a run r such that up to time t : (a) all processes in P_0 act as they do in r_0 (they decide 0 at time t_0), and (b) all processes in P_1 act as they do in r_1 (they decide 1 at time t_1). This is because up to time t , processes in P_0 cannot distinguish between Scenario 1 (in which processes in P_1 crashed at the start) from Scenario 3 (in which all messages from processes in P_1 have been delayed); and, similarly, processes in P_1 cannot distinguish between Scenario 2 and Scenario 3. After time t , we extend r by delivering all the delayed messages and by continuing r into an infinite run where no process is faulty and each message sent is delivered. Run r of the algorithm does not satisfy Agreement — a contradiction. \square

[†] Termination means that all correct processes eventually decide. In this algorithm, each process executes an infinite loop, so processes never really halt. It is possible to modify the algorithm so that they do.