MADS

Emmanuelle Anceaume

Lesson 3: Randomized Consensus in Asynchronous Environments

http://people.irisa.fr/Emmanuelle.Anceaume/

- We consider distributed systems where processes can communicate and synchronize by exchanging messages (message-passing model).
- The system is composed of n processes usually denoted $\Pi = \{p_1, \ldots, p_n\}$.
- The system is asynchronous because there exists no bound :
  - neither on the relative speeds of processes
  - nor on the communications speed.

Why such a model ?

- It is extremely simple
- If a problem can be solved in asynchronous systems, it can be solved in more constrained model (like synchronous systems or partially synchronous systems)
- A solution to a problem P in this model can always be used directly in a more demanding model M
  - It will then benefit from the good properties exhibited by model M
  - While at the same time being robust enough to tolerate violations of the properties exhibited by model M

Informal specification

- In this problem processes are trying to reach a consensus.
- Each process initially proposes a value v taken from a given set of value V.
- At the end of the protocol, all processes agree on a single value, called the decided value, or decision.
- This value must have been proposed by one of the processes.

Specification

Each process has an initial value and at the end of the protocol, the following must hold :

- Termination : All correct processes must eventually decide a value.
- Integrity : At most one decision per process.
- Agreement : All processes that decide (correct or not) must decide the same value.
- Validity : The value decided by a process must have been initially proposed.

FLP impossibility result [FLP85]

Theorem (FLP impossibility result)

There exists no deterministic algorithm that solves the binary consensus problem in the presence of even if a single faulty process [a]

a. M. Fischer, N. Lynch, and M. Paterson. « Impossibility of distributed consensus with one faulty process ». Journal of the ACM, 32(2) : 374-382, 1985

Binary consensus : processes have solely two possible input values « 0 » and « 1 »

A randomized consensus algorithm

- Soon after the FLP impossibility results appeared, people try to find a way to circumvent it.
- Ben-Or gave the first the first randomized algorithm that solves consensus with probability 1
- Asynchronous message-passing system with $f \leq n/2$ crash failures (n number of processes and f max. number of processes that may crash

Model of the system

- Set of n processes
- At most $f < n/2$ processes may crash (may stop to take steps)
- Asynchronous environment
- Communication channel is reliable
- Each process has access to a coin : when a process tosses its coin, it obtains 0 or 1 with probability $1/2$.

Step

A step of execution is as follows :

- Receipt of a message
- Tosses a coin (optional)
- Changing its state
- Sending a message to all processes

Adversary

When designing fault-tolerant algorithms, we often assume the presence of an adversary

- It has some control on the behavior of the system
- It knows the content of all sent messages
- It knows the local state of each process
  - $\rightarrow$ it can select the next process to take a step
  - $\rightarrow$ It can select the message the process will receive
- However
  - It cannot prevent a message from being eventually received
  - It cannot make more than f processes crash

The randomized consensus problem

Every process has some initial value $v_p \in \{0, 1\}$, and must decide on a value such that the following properties hold :

- Agreement : No two processes decide differently
- Validity : If any process decides v, then v is the initial value of some process
- Termination : With probability 1, every correct process eventually decides

Note that Agreement and Validity are safety properties and Termination is a liveness property.

Ben Or's randomized consensus algorithm

- First algorithm [1] to achieve consensus with probabilistic termination in an asynchronous model
- The algorithm is correct if no more than f crash occur with $f < n/2$

1. M. Ben-Or. « Another advantage of free choice : Completely asynchronous agreement protocols (extended abstract) ». In Proc. of the 2nd annual ACM Symposium on Principles of Distributed Computing Systems (PODCâĂŹ83), pages 27âĂŞ30, 1983.

Ben-Or's randomized consensus algorithm

- Operates in asynchronous rounds
- In each round, processes exchange messages twice
- Each message exchange is called a phase
- Every message contains a tag (R or P), a round number, and a value 0 or 1, and « ? » for massages tagged P
- Messages tagged R are sent in the first phase (they are called reports)
- Messages tagged P are sent in the second phase (they are called proposals)

Ben-Or's randomized consensus algorithm

General idea :

- Report phase : each process transmits its value, and waits to hear from other processes
- Proposal phase :
  - If enough processes detect the majority, then decide
  - If I know that someone detected majority, then switch to the majority value
  - Otherwise, flip a coin ; eventually a majority of correct processes will flip in the same way

Ben-Or's randomized consensus algorithm

Every process p_i executes the following algorithm :

```
procedure consensus(v_i)
{
x ← v_i // p_i's current estimate of the decision value
k = 0
while true do
  k ← k + 1 // k is the current round number
  send (R, k, x) to all processes
%
wait for (R, k, *) msgs from n − f processes //"*" in {0,1}
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 (P, k, *) msgs from n − f processes //"*" in {0,1,?}
if received at least f + 1 (P, k, *) with the same v ≠? then
  decide v
if received at least one (P, k, v) with v ≠? then
  x ← v
else
  x ← 0 or 1 randomly // toss coin
}
```

Ben-Or's randomized consensus algorithm

```
procedure consensus(v_i)
{
 x ← v_i // p_i's current estimate of the decision value
 k = 0
 while true do
   k ← k+1 // k is the current round number
   send (R,k,x) to all processes

   wait for (R,k,*) msgs from n−f processes //"*" in {0,1}
   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 (P,k,*) msgs from n−f processes //"*" in {0,1,?}
   if received at least f+1 (P,k,*) with the same v ≠? then
     decide v
   if received at least one (P,k,v) with v ≠? then
      x ← v
   else
      x ← 0 or 1 randomly // toss coin
```

At the end of the first phase a process
- proposes v if received a strict majority of reports v
- proposes ? otherwise
if v=1 is proposed then v=0 cannot be proposed

Ben-Or's randomized consensus algorithm

```
procedure consensus(v_i)
{
 x ← v_i // p_i's current estimate of the decision value
 k = 0
 while true do
   k ← k + 1 // k is the current round number
   send (R, k, x) to all processes

   wait for (R, k, *) msgs from n − f processes //"*" in {0,1}
   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 (P, k, *) msgs from n − f processes //"*" in {0,1,?}
   if received at least f + 1 (P, k, *) with the same v ≠? then
     decide v
   if received at least one (P, k, v) with v ≠? then
     x ← v
   else
     x ← 0 or 1 randomly // toss coin
```

At the end of the second phase a process
- decides v if received f+1 proposals v (≠ ?)
- adopts v for x if received at least one v (≠ ?)
- chooses a random value for x otherwise

Proof of correctness

Let p_i and p_j be any two processes.

Lemma 1

It is impossible for p_i to propose 0 and for p_j to propose 1 in the same round $k \geq 1$

Proof : By contradiction.

- Suppose that p_i proposes 0 and p_j proposes 1 in round k.
- Thus p_i received more than $n/2$ reports $= 0$ and p_j received more $n/2$ reports $= 1$ in round k
- Thus it exists a process p_k that reports 0 to p_i and 1 to p_j in round k
- This is impossible

Proof of correctness

Lemma 2

If some process p_i decides v in round $k \geq 1$, then all the processes p_j that start round $k+1$ do so with $x_{p_j} = v$.

Proof :

- Suppose that some p_i decides v in round k (this occurs in L17 and $v \neq ?$)
- p_i must have received $f+1$ proposals for v in round k
- Let p_j be any process that starts round $k+1$.
- p_j received $n-f$ proposals in L15 of round k
- Since p_i received at least $f+1$ proposals for v in round k, p_j received at least one proposal for v in round k
- By Lemma 1, p_j did not receive any proposal for $1-v$ in round k
  - $\rightarrow$ So p_j sets $x_j = v$ in Line 19 in round k
  - $\rightarrow$ And p_j starts round $k+1$ with $x_j = v$

We say that v is $(k+1)$-locked

Proof of correctness

Lemma3

If a value v is k-locked, then every process that reaches L18 in round k decides v in round k

Proof :

- Suppose v is k-locked
- Then all reports received in L9 of round k are equal to v
- Since $n - f > n/2$, every process that proposes a value in round k proposes v in L11
- Consider some process p that reaches L18 in round k
- Clearly, p received n-f proposals in L15 for v in round k
- Since $n - f > f + 1$, every process that reaches Line 12 decides v in round k

Proof of correctness

Corollary 1

If some process decides v in round k, then every processes that reaches L18 in round $k+1$ decides v in round $k+1$

Proof :
By Lemma 2 and 3

Proof of correctness

Corollary 2 - Agreement

If some processes p_i and p_j decide v and v' in round k and k' then $v = v'$

Proof : Suppose that p_i and p_j decide v and v' in round k and k'. There are 2 cases :

1. $k = k'$. Then both v and v' were proposed in round k. By Lemma 1, $v = v'$
2. $k < k'$.
    - Since p_j decides in round k', p_j executed L18 in round $k+1, \ldots k'$
    - Since p_i decides v in round k, by repeated applications of Corollary 1, p_j decides v in rounds $k+1, \ldots, k'$.
    - So p_j decides both v' and v in the same round k', by case (1), $v = v'$

Proof of correctness

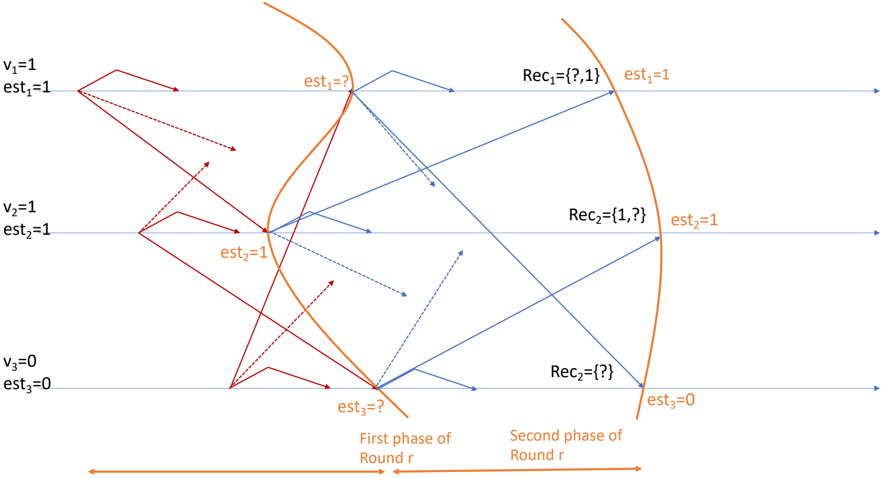
Validity

If any process p decides v, then v is the initial value of some process

Proof : by contradiction.

- Suppose that some process p decides a value v that has never been proposed.
- Then all the processes have the initial value $1 - v$
- So $1 - v$ is 1-locked (i.e., locked in round 1)
- From Lemma 3, p decides $1 - v$ in round 1
- So p decides both v and $\overline{v}$.
- This is a contradiction by Corollary 2.

Why the random oracle allows to break symmetry



Liveness property

We have seen that that both agreement and validity hold, i.e.

Validity

If any process p decides v, then v is the initial value of some process

Agreement

If some processes p_i and p_j decide v and v' in round k and k' then $v = v'$

We need to show that with probability 1, there is a round at which all the non faulty processes strat with the same estimate of the decision value

Liveness property

- By Lemma 3, if some value v is k-locked, then v is decided in round k
- At round k, the probability that some value v is k-locked is at least $(1/2)^n$
  - indeed some process p_i can set $x_i = v$ not necessarily by flipping a coin

Liveness property

- Hence, for any round k

$$Pr[\text{ no value is k-locked}] < 1 - (1/2)^n$$

- Since coin flips are independent,

$$Pr[\text{ no value is k-locked for the k first rounds }] < (1-(1/2)^n)^r$$

- Thus the proba that v is k-locked during the first r rounds is

$$Pr[\text{ v is k-locked during the k first rounds }] \geq 1-(1-(1/2)^n)^r$$

Any questions ?