2022 pastpaper

1.a

defmodule Consensus do

def start(rb, n, f, pval) do

next(rb, n, f, pval, [])

end

defp next(rb, n, f, pval, vals) do

send rb, {:rb\_broacast, {:rb\_pass1, self(), pval}} # broadcast to all

pass1(rb, n, f, pval, vals,%{},MapSet.new())

end

defp pass1(rb, n, f, pval, vals, svals\_stats, senders) do

receive do

{:rb\_deliver, {:rb\_pass1, sender, sval}=value} ->

MapSet.put(senders,sender)

Map.put(svals\_stats,sval,Map.get(svals\_stats,sval,0)+1)

vals = vals ++ [sval]

if Enum.count(senders) >= (n-f) do

passed\_1=false

for sval <- Map.keys(svals\_stats) do

if Map.get(svals\_stats,sval,0) > n/2 do

passed\_1=true

pass2(...,sval,%{},MapSet.new())

end

end

if not passed\_1 do

pass2(...,nil,%{},MapSet.new())

end

else

pass1(...)

end

end

end

defp pass2(rb, n, f, pval, vals,svals\_stats,senders) do

send rb, {:rb\_broacast, {:rb\_pass2, self(), pval}}

receive do

{:rb\_deliver, {:rb\_pass2, sender, sval}=value} ->

vals = vals ++ [sval]

MapSet.put(senders,sender)

Map.put(svals\_stats,sval,Map.get(svals\_stats,sval,0)+1)

if Enum.count(senders) >= (n-f) do

decided=false

for sval <- Map.keys(svals\_stats) do

if Map.get(svals\_stats,sval,0) > f and sval!=nil do

decided=true

decide(sval)

end

end

if not decided do

non\_nil\_vals = Enum.filter(Map.keys(svals\_stats), &(&1 != nil))

if Enum.count(non\_nil\_vals) > 0 do

next(rb, n, f, Enum.at(non\_nil\_vals, 0), vals)

else

random\_val = Enum.at(vals, :rand.uniform(length(vals)))

next(rb, n, f, random\_val, vals)

end

end

else

pass2(...)

end

end

end

End

(Might be wrong, are there any other thoughts?)

I basically have the same :)

1.b

Distinct Version 👇:D

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | N1 | N2 | N3 |
| Round 1 | Pass 1 broadcast | v1 | v2 | v3 |
|  | Pass 1 deliver | v1,v2 | v1,v2 | v1,v3 |
|  | Pass 2 broadcast | nil | nil | nil |
|  | Pass 2 deliver | nil,nil | nil,nil | nil,nil |
| Round 2 | Pass 1 broadcast | v1 | v2 | v1 |
|  | Pass 1 deliver | v1,v1 | v1,v2 | v1,v1 |
|  | Pass 2 broadcast | v1 | nil | v1 |
|  | Pass 2 deliver | v1,v1 | nil,v1 | v1,v1 |
| Halt | Decision | v1 | v1 | v1 |

(Might be wrong as well, any other solution?, is the final decision being broadcast to every node?)

1.c

Validity: If a process decides a value, then that value was proposed by some process. In this case, the decided value must be proposed by some process so that it can pass through all procedures.

Agreement: No two processes decide on different values. This holds because in pass1, we only broadcast nil or non-nil value if it is delivered more than N / 2 (there will be only one unique value can satisfy that) , which guarantees in pass2, all the value being processed should be the same value or just nil. Thus, in the final decision, no two processes decide on different values.

Termination: every correct process will eventually decide some value with probability converging to 1. As the consensus algorithm iterates, the number of possible proposed values in all processes will only decrease or remain the same after each round. So eventually every correct process will decide some value.

2. Uncovered content