Randomized consensus

SLR206, P2

So far…

- Consensus is impossible to solve *deterministically* using read-write registers if at least one process may crash

Next

- Circumventing consensus impossibility
  - ✓Commit adopt -> obstruction-free consensus
  - ✓Randomized consensus -> Ben-Or's algorithm

Circumventing consensus impossibility: commit-adopt

A variant of consensus with weaker safety (relaxed agreement)

Can be used for solving consensus with an oracle

A process p_i *proposes* an *input* value in V (|V|≥2) and *decides* on a tuple (c,v) where c is a boolean and v is in V

- ✓We say p_i *adopts* v
- ✓If c=true, we say p_i *commits* on v

Commit-adopt: properties

- *Validity:* Every adopted value is an input value of some process
- *Termination:* Every correct process decides (commits or adopts)
- *CA-Agreement:*
  - ✓If a process commits on a value v, then no process can adopt a value v’ ≠v
  - ✓If all inputs are the same, then no process decides on (false,*)

    (*every process that decides commits on a value)*

Commit-adopt : protocol

Shared objects:

N atomic registers A[0,…,N-1], initially T

N atomic registers B[0,…,N-1], initially T

Upon propose(v) by process p_i:

v_i := v

A[i] := v_i

U := read A[0,…,N-1]

if all non-T values in U are v then

    B[i] := (true,v_i)

else

    B[i] := (false,v_i)

U := read B[0,…,N-1]

if all non-T values in U are (true,*) then

    return (true,v_i)

else if U contains (true,v') then

    v_i := v'

return (false,v_i)

Commit-adopt: proof

Validity and Termination: immediate

CA-Agreement:

Claim 1 B[0,…,N-1] never contains (true,v) and (true,v') where v≠v'

Suppose not: p_i wrote (true,v) in B[i] and pj wrote (true,v') in B[j], v≠v'
Previously, p_i wrote v in A[i] and pj wrote v' in A[j] (let pi be the first to write)
But p_j should have seen A[i] ≠v' - a contradiction!

Commit-adopt: proof (contd.)

Claim 2 If p_i returns (true,v) then no process pj returns (c,v') where v≠v'

Suppose not: let p_j return (c,v') where v≠v'.

By Claim 1, p_j has previously written some (false,v'') in B[j]

Since p_j hasn't adopted v, it hasn't found (true,v) in B[1,…,N]

But then p_i should have read (false,v'') in B[j] – a contradiction!

Commit-adopt: proof (contd.)

Claim 3 If all inputs are the same then no process returns (false,*)

Immediate: both “if” conditions are true, i.e., the non-T values in A and B are the same

Ω: an oracle

- Eventual leader failure detector
- Outputs (at every process) a process identifier (a leader)
  - ✓‹Ω, p›
- Eventually, the same correct process is output at every correct process

Can be implemented in eventually synchronous system:

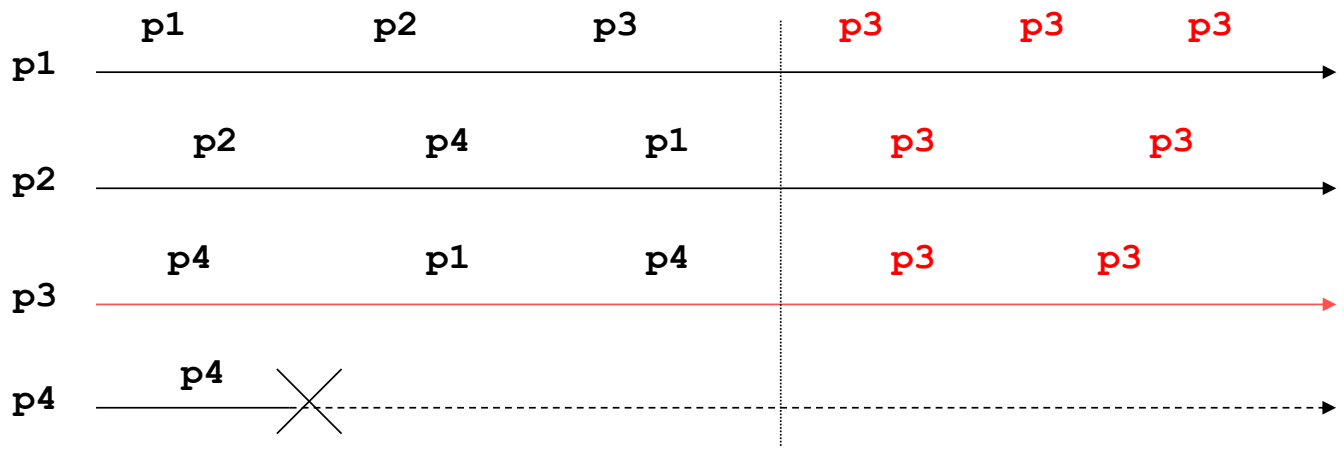
- ✓There is a bound on communication delays and processing that holds only eventually
- ✓There is an a priori unknown bound in every run

(Two computationally equivalent conditions)

Leader election Ω: example

There is a time after which the same correct process is considered leader by everyone.

(Sufficient to output a binary flag leader/not leader)



Consensus = Ω + CA

Shared:

D, a regular register, initially T

$CA_1, CA_2, \ldots$ a series of commit-adopt instances

Upon propose(v) by process p_i:

v_i := v
r := 0
repeat forever
    r++
    wait until Ω outputs p_i or D≠T
    if D=v' where v'≠T then
        return v'
    $(c,v_i):=CA_r(v_i)$ // r-th instance of commit-adopt
    if c=true then
        D:=v_i // let the others learn your decision
        return v_i
    v_i := v' //adopt the leader's value

Quiz 9.1: commit-adopt

- Would the CA algorithm is correct if regular registers were used?
- Show that Ω + CA indeed solve consensus
- Give an obstruction-free consensus algorithm using CA
  - ✓Obstruction-freedom: every process that runs solo from some point on eventually decides

每个进程独立地随机选择其决策

Randomized consensus = CA+local randomness

- Ω ensures termination: eventually a correct leader is elected
- Tossing a coin instead of consulting an oracle?
- Termination with probability 1
  - ✓Every correct processes almost surely eventually outputs

使用 commit-adopt 机制，每个进程提出一个值。
如果进程是当前轮次的领导者（由 Ω 确定），则其提议的值有更高的机会被提交，因为其他进程会采纳领导者的值。
如果进程不是领导者，它将采纳它看到的任何已提交的值，或者如果它没有看到任何已提交的值，它将随机投币来选择它的提案。
这个过程会不断重复，直到所有正确的进程都提交了同一个值。

由于随机性的引入，即使在异步网络和可能的进程失败下，算法也能够保证最终达成共识。这是因为随机选择在统计学意义上保证了算法最终会在某个点上收敛。每个正确的进程几乎肯定（概率为 1）最终会输出决策，这意味着虽然在理论上可能需要无限的时间，但在实践中很可能在可接受的时间范围内达成共识。

Adjusted commit-adopt: properties

Process p_i outputs (c,v) where v is a value (in V) and c is in {0,1,2}

- ✓If c= 1, we say p_i *adopts* v
- ✓If c=2, we say p_i *commits* on v
- ✓If c=0, we say p_i *sticks to its input*

- *Validity:* Every returned value is an input value of some process
- *Termination:* Every correct process decides
- *CA-Agreement:*
  - ✓If a process commits on a value v, then no process can stick to its input or adopt a value v' ≠v
  - ✓If all inputs are the same, then no process decides on (c,*) with c in {0,2}

    (*every process that decides commits on a value)*

Adjusted commit-adopt

Shared objects:

N atomic registers A[0,…,N-1], initially T

N atomic registers B[0,…,N-1], initially T

Upon propose(v) by process p_i:

v_i := v

A[i] := v_i

U := read A[0,…,N-1]

if all non-T values in U are v then

B[i] := (true,v_i)

else

B[i] := (false,v_i)

U := read B[0,…,N-1]

if all non-T values in U are (true,*) then

return (1,v_i)

else if U contains (true,v') then

return (2,v')

return (0,v_i)

Randomized (binary) consensus = CA+local randomness

Shared:

$CA_1, CA_2, \ldots$ a series of commit-adopt instances

Upon propose(v) by process p_i:

$v_i := v$
$r := 0$
repeat forever
    r++
    $(c, v_i) := CA_r(v_i)$ // r-th instance of commit-adopt
    if c=1 then
        return v_i
    else if c=2
        $v_i := v'$ //adopt the leader's value
    else
        v_i := Random() // Pick a random value in {0,1}

A variation of Bet-Or's consensus
("Another advantage of free choice", PODC'83)

Randomized consensus

- Prove Ben-Or's variation?
- Expected time complexity
- Improve time complexity?

Quiz 9.2: Randomized consensus

- Getting to multiple-value consensus?
  - ✓Hints: use one binary consensus instance per process or per possible input value