Obstruction-free Consensus and Paxos

Jiale KANG & Yuanjie ZHAO

Overview

Our simulation of Synod algorithm:

- Export one operation *propose(v)*, $v \in \{0,1\}$
- When a process invokes *propose(v)*, it will either *decide* or *abort*
- One process may invoke *propose(v)* multiple times
- f faulty processes, with probability α to crash for each operation
- Leader-election mechanism: non-leader process stops proposing after t_{le}

Proof of correctness

- Validity
  - Every decided value is a proposed value.
    - Obvious since the algorithm will not generate new values
- Agreement
  - No two processes decide differently.
    - Suppose two processes *i* and *j*: p_i decides *v* with ballot *b* which is the lowest among all processes, p_j send an impose message with ballot *b'* and value *v'* where *b'* is the lowest ballot number higher than *b* attached to an impose message
    - There exists two majorities of processes, one for the impose phase of p_i and one for the read phase of p_j
    - Let p_k be a process in both majorities. p_k must have received p_i's impose request before p_j's read request in order not to abort. Thus, p_i must have received a gather response from p_k containing *b* and *v*. Thus by previous assumption of the ballot number of p_j, we can prove that *v*=*v'*, which means no two processes decide differently

Proof of correctness

- Obstruction-free termination
  - If a correct process proposes, it eventually decides or aborts.
    - correct process receives either ABORT or GATHER
    - when receives GATHER from majority
      - sends IMPOSE and waits to receive ACKs
      - receives ACKs from majority, DECIDE
  - If a correct process decides, no correct process aborts infinitely often.
    - before decides, send DECIDE to all
    - others when receive DECIDE, decide
  - If there is a time after which exactly one correct process *p* proposes a value sufficiently many times, *p* eventually decides.
    - at some point, the proposal ballot *b* will be the greatest number in the system
    - cannot go to ABORT and eventually decides

Synod OFCons I

```
Code of every process pi:

Initially:
   ballot:=i-n; proposal:=nil; readballot:=0; imposeballot:=i-n;
   estimate:= nil; states:=[nil,0]^n

upon propose(v)
   proposal := v; ballot:=ballot + n; states:=[nil,0]^n
   send [READ, ballot] to all

upon receive [READ,ballot'] from pj
   if readballot > ballot' or  imposeballot > ballot' then
     send [ABORT, ballot'] to pj
   else
     readballot:=ballot'
    send [GATHER, ballot', imposeballot, estimate] to pj

upon receive [ABORT, ballot] from some process
   return abort
```

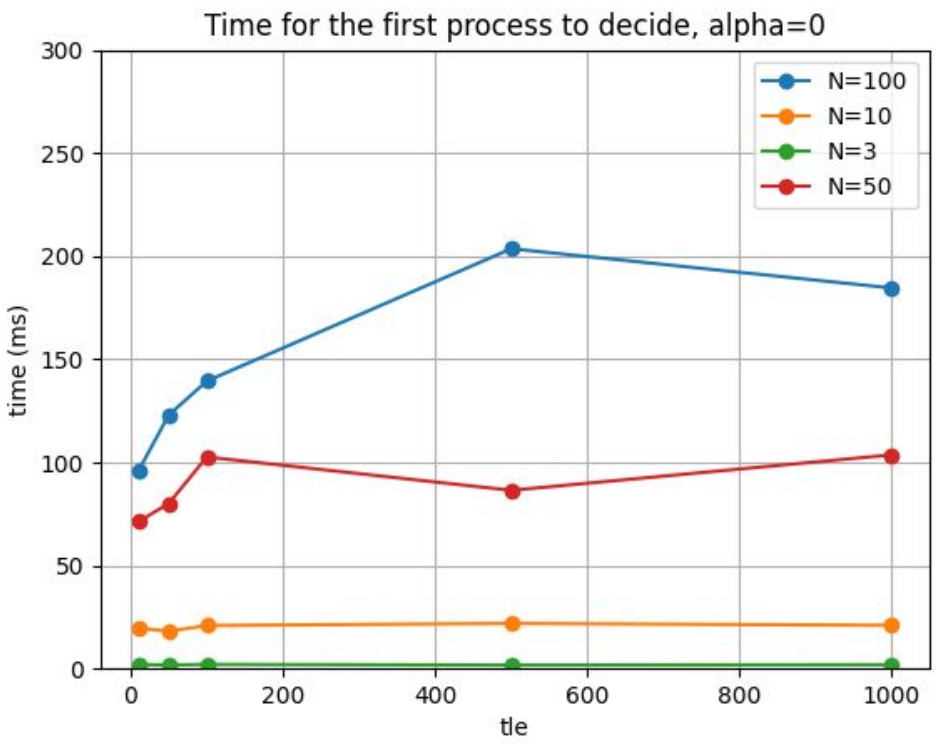
Synod OFCons II

```
upon receive [GATHER, ballot, estballot, est] from pj
  states[pj]:=[est,estballot]

upon  #states ≥ majority //collected a majority of responses
  if ∃ states[pk]=[est,estballot] with estballot>0  then
     select states[pk]=[est,estballot] with highest estballot
     proposal:= est //choose a potentially decided value
  states:=[nil,0]^n
  send [IMPOSE, ballot, proposal] to all

upon receive [IMPOSE,ballot',v] from p_j
   if readballot > ballot' or  imposeballot > ballot' then
     send [ABORT, ballot'] to p_j
  else
    estimate := v; imposeballot:=ballot'
        send [ACK, ballot'] to p_j
```

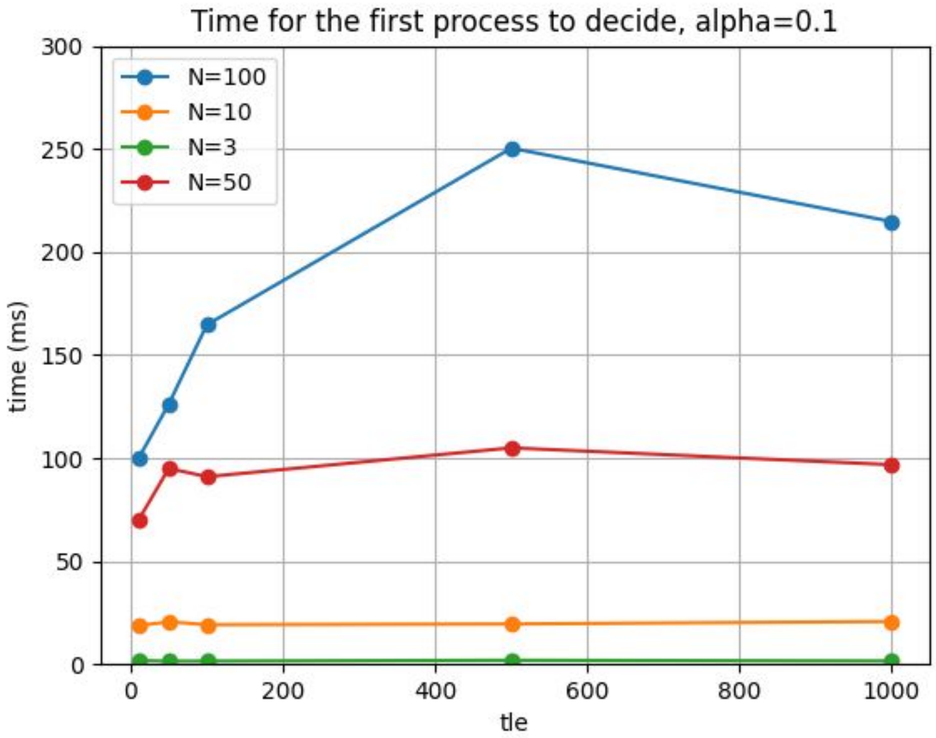
Synod OFCons III

```
upon received [ACK, ballot] from majority
  send [DECIDE, proposal] to all

upon receive [DECIDE, v]
   send [DECIDE, v] to all
   return [decide, v]
```

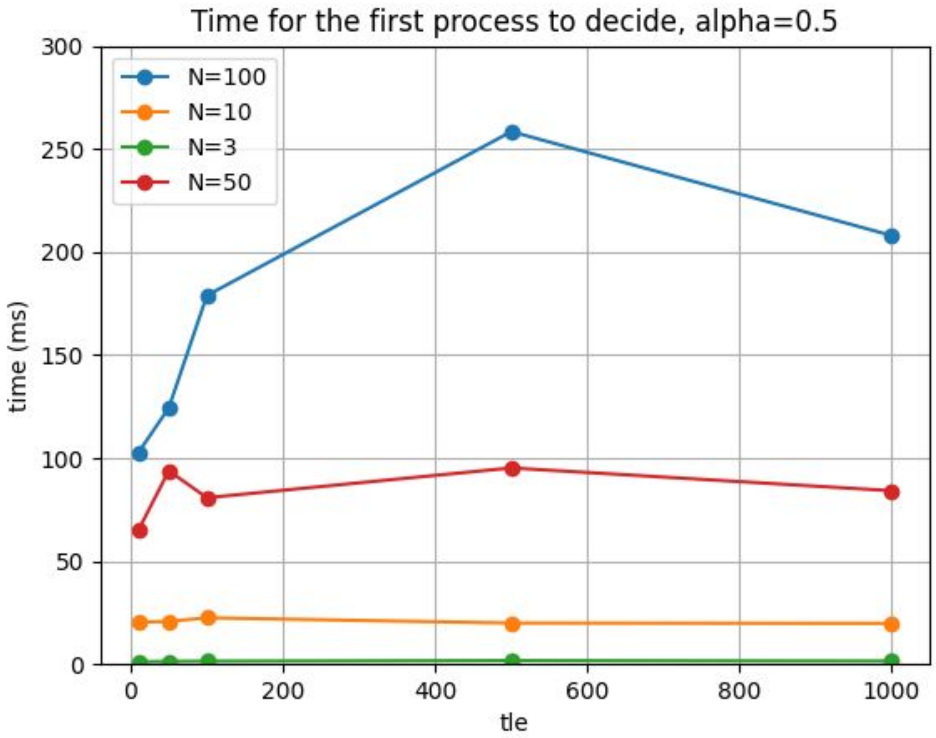
Performance analysis



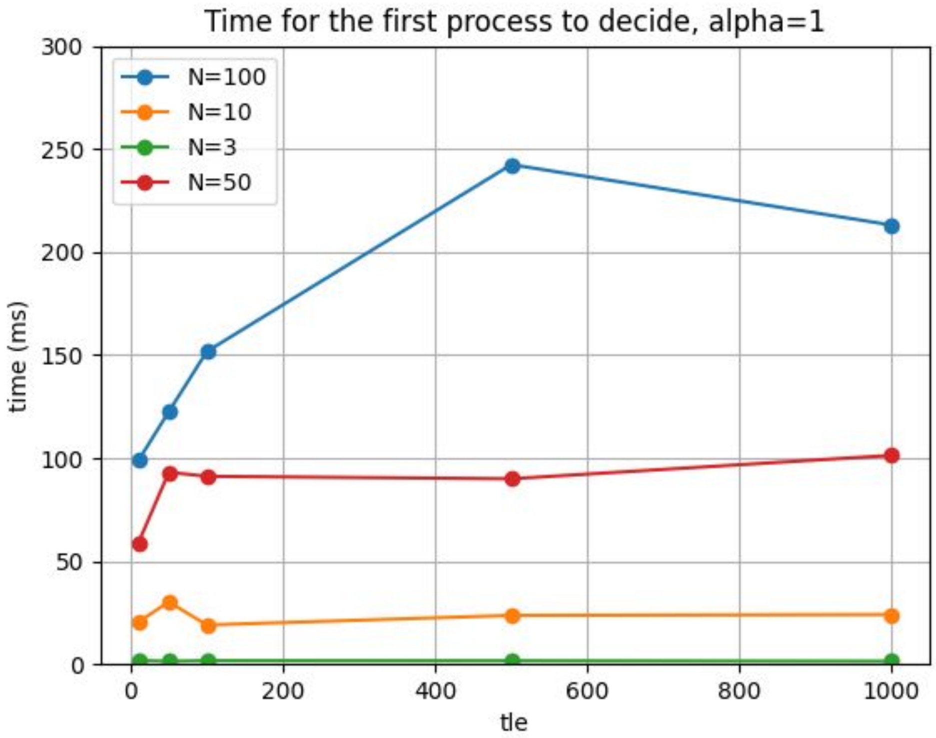
Performance analysis



Performance analysis



Performance analysis



Conclusion

- Larger systems use more time to complete consensus;
- Smaller systems show lower and stable consensus times;
    - less sensitive to the number of leader elections and crash probabilities
- Larger systems are more sensitive to t_{le} but still keeps stable for the variations of $\boldsymbol{\alpha}$.