Fault Tolerant Consensus – Wrapup I

Radu Nicolescu
Department of Computer Science
University of Auckland

16 Oct 2020

1. Synchronous network model
2. Stopping failures
3. EIGStop
4. Side by side
5. Byzantine agreement with authentication

Synchronous network model

- All these algorithms are still based on the synchronous network model
- Even the simple stopping failure cannot be deterministically solved in the asynchronous network model

- FLP – Fischer, Lynch and Patterson

  Impossibility of Asynchronous Distributed Consensus with a Single Faulty Process

- Solutions for the asynchronous model use randomisation, failure detectors (partially synchronous model)

Stopping failures model

- Much simplified version of the Byzantine agreement
- A failed process can only stop sending messages, forever
  (no intermittent failures, recovery not considered)

- No possibility to send confusing messages
  (i.e. different messages to different directions)

- The problem can be solved for any $F \leq N - 1$ ☺
  (not only when $3F \leq N - 1$)

The Stopping agreement conditions – vs ~~Byz~~

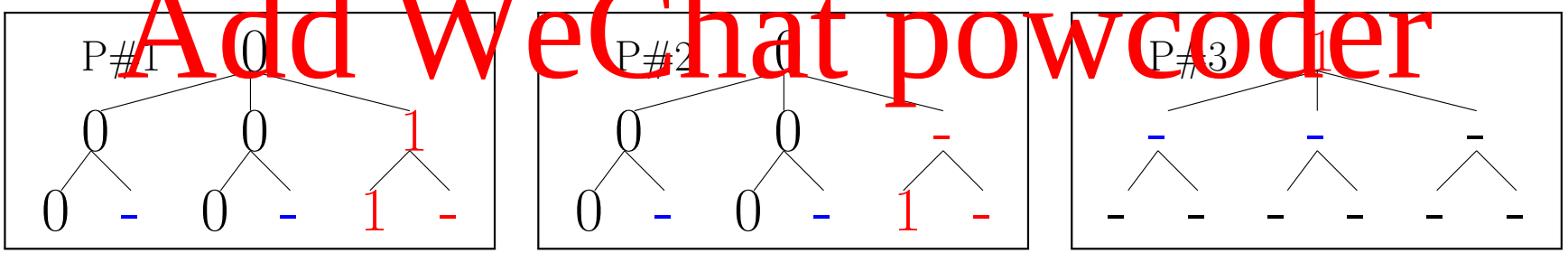
- Termination: all non-faulty processes eventually decide
- Agreement: no two ~~non-faulty~~ processes ever decide on different values
- Validity: if all ~~non-faulty~~ processes start with the same initial value $v \in V$, then v is the only one possible decision value
- If the processes start with different initial values, then the final decision could be any of these (as long as it is consistent)

EIGStop

- EIG tree as in the EIGByz, $F + 1$ messaging rounds
  - recall: F can be as high as $N - 1$ (not at most $(N - 1)/3$)
- Top-down val()'s as in the EIGByz, i.e. via messaging
- No bottom-up newval() attributes
- Final decision: set W of all non-null val()'s in EIG tree
  - all values at all levels! not just leaves
  - nulls discarded! not assumed v_0
- If W is singleton, $W = \{v\}$, then the decision is v
- Otherwise, if W is mixed, $W = \{0, 1\}$, then the decision is v_0
  - no voting! no tie breaking

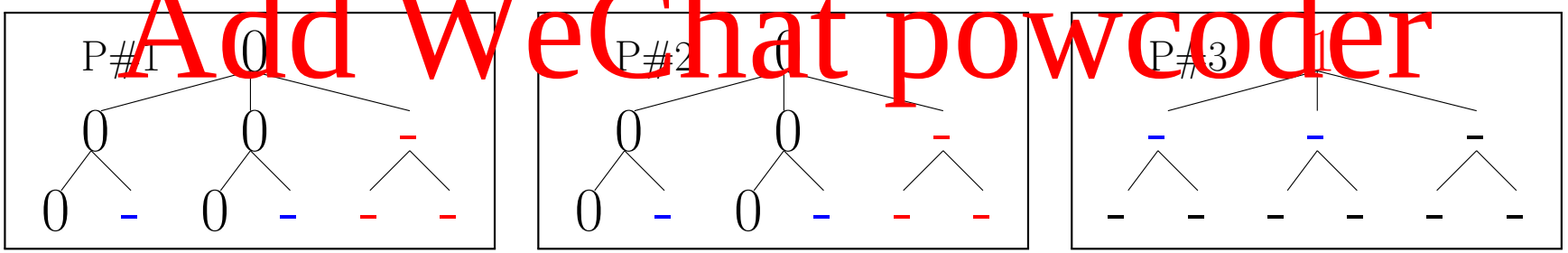
EIGStop example – assuming $v_0 = 1$; nulls as -

- Process #1 : init 0; decision $v_0 = 1$
- Process #2 : init 0; decision $v_0 = 1$
- Process #3 : init 1; no decision; fails after sending one 1st round message, to #1



EIGStop example – assuming $v_0 = 1$; nulls as -

- Process #1 : init 0; decision 0
- Process #2 : init 0; decision 0
- Process #3 : init 1; no decision; fails before sending any 1st round message



EIGStop example – assuming $v_0 = 1$; nulls as -

- WHAT IF scenario – NOT supported by this EIGStop protocol

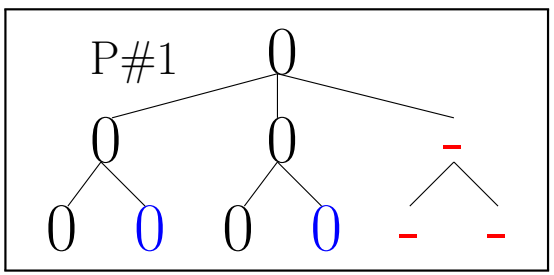
  What if P#3 fails before sending any 1st round out-message but would be allowed to recover and decide

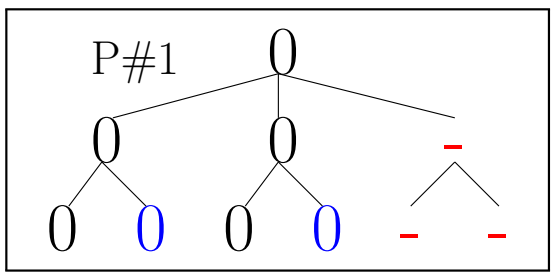
- NO agreement

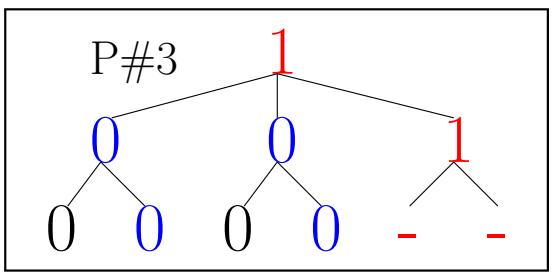
- Process #1 : init 0; decision 0

- Process #2 : init 0; decision 0

- Process #3 : init 1 ; decision $v_0 = 1$







OptEIGStop

- Each process sends out only two messages
- First: its initial choice, to all the processes
- Second: a different value, to all the processes
  - The first time it learns about a different value
  - Arbitrary choice, if there are more

EIGStop vs EIGByz vs 3PC – assuming $v_0 = 0$

- X indicates a faulty process, which fails from start, before sending any 1st round message

Initial	EIGStop	EIGByz	3PC
0 0 0 0	0	0	0
0 0 0 1	0	0	0
0 0 1 1	0	0	0
0 1 1 1	0	1	0
1 1 1 1	1	1	1
0 0 0 X	0	0	0
0 0 1 X	0	0	0
0 1 1 X	0	0	0
1 1 1 X	1	1	0

EIGStop vs EIGByz vs 3PC – assuming $v_0 = 1$

- X indicates a faulty process, which fails from start, before sending any 1st round message

Initial	EIGStop	EIGByz	3PC
0 0 0 0	0	0	0
0 0 0 1	1	0	0
0 0 1 1	1	1	0
0 1 1 1	1	1	0
1 1 1 1	1	1	1
0 0 0 X	0	0	0
0 0 1 X	1	1	0
0 1 1 X	1	1	0
1 1 1 X	1	1	0

Complexity

- EIGStop
  - Rounds: $f+1$
  - Messages: $\mathcal{O}((f+1)n^2)$ messages
  - OptEIGStop messages: $\mathcal{O}(2n^2)$ messages
- EIGByz – Additional requirement: $n > 3f$
  - Rounds: $f+1$
  - Messages: $\mathcal{O}((f+1)n^f)$ messages
- 3PC:
  - Rounds: $\mathcal{O}(f+1)$
  - Messages: $\mathcal{O}(fn)$ messages

Byzantine agreement with authentication

- Assume that each process digitally signs its messages in a total safe way, e.g. based on a reliable unbreakable PKI/DSA...
- Is this reasonable?
- Problem with certificate weaknesses: What if a powerful Byzantine faulty process is able to forge such signatures?
- Problem with authority: What if the certification authority itself is hacked or even turns into a Byzantine process?
- Anyway, assuming that such digital signatures are totally safe, Byzantine faulty nodes are not able to wreak much more havoc than a stopped process
- EIGStop can be adapted to solve the (slightly different) Byzantine agreement with authentication
- Faster/better/more general algorithms possible...