CS 3551 DISTRIBUTED COMPUTING

UNIT IV CONSENSUS AND RECOVERY **10**

Consensus and Agreement Algorithms: Problem Definition – Overview of Results – Agreement in a Failure-Free System(Synchronous and Asynchronous) – Agreement in Synchronous Systems with Failures; Checkpointing and Rollback Recovery: Introduction – Background and Definitions – Issues in Failure Recovery – Checkpoint-based Recovery – Coordinated Checkpointing Algorithm -- Algorithm for Asynchronous Checkpointing and Recovery

Consensus algorithms for Byzantine failures (synchronous system)

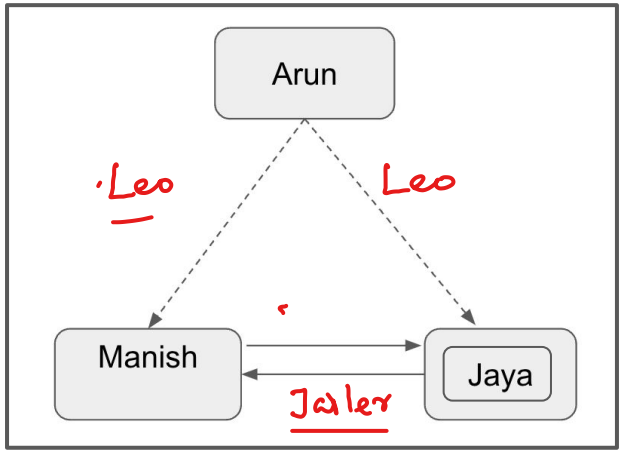
1. Upper bound on Byzantine processes
2. Byzantine agreement tree algorithm: exponential (synchronous system) Recursive formulation
3. Byzantine agreement tree algorithm: exponential (synchronous system) Iterative formulation
4. Phase King Algorithm

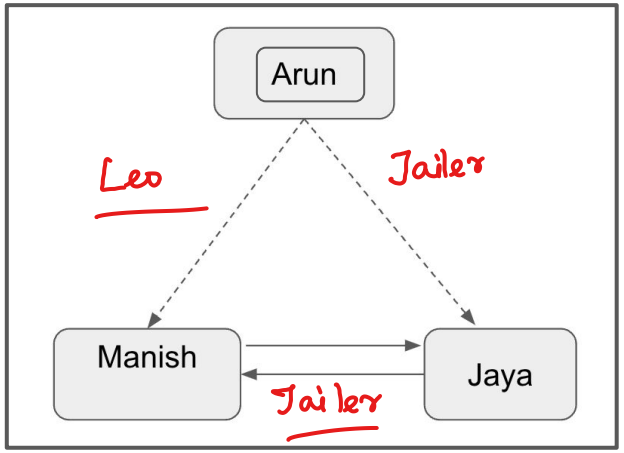
① Upper bound on Byzantine processes

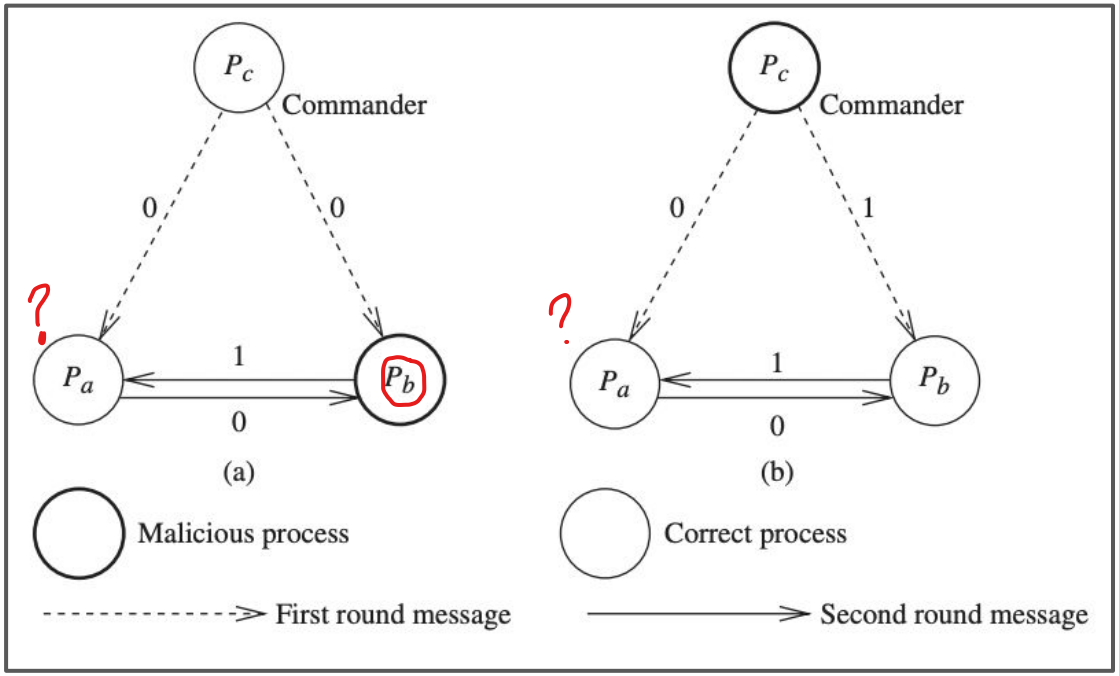
$\Rightarrow$ iff $f \leq \lfloor \frac{n-1}{3} \rfloor$

N = 3 ; f = 1 $\quad 1 \leq \lfloor 2/3 \rfloor$

$1 \leq 0$ X






P_c
Commander
0
0
P_a
1
P_b
0
(a)
Malicious process
First round message
P_c
Commander
0
1
P_a
1
P_b
0
(b)
Correct process
Second round message

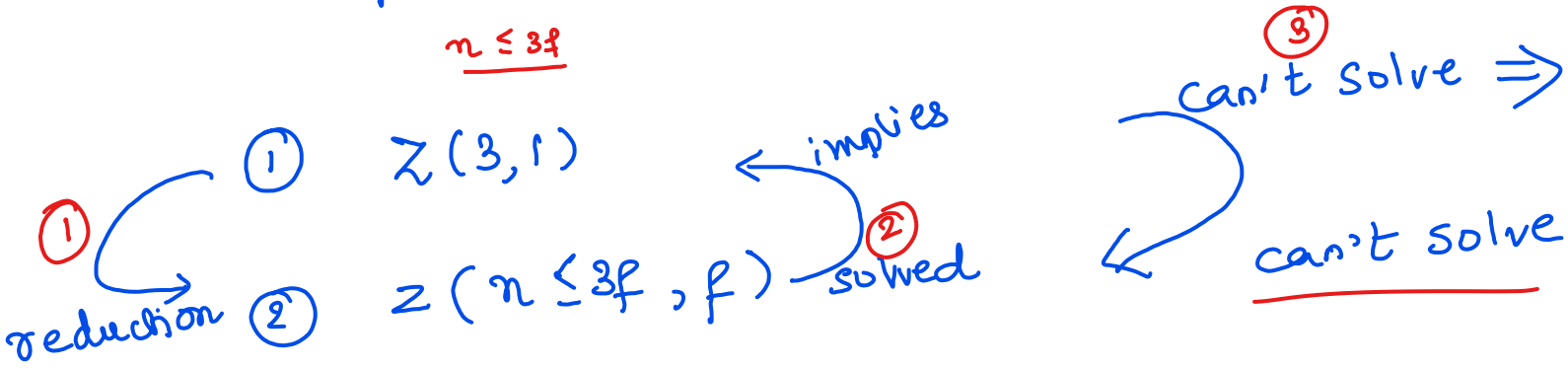
1. **Context: Byzantine Agreement Problem with n=3 processes and f=1 Byzantine process.**
2. **Scenario 1 (Figure 14.3(a)):**
   - Processes: Commander Pc, Lieutenants Pa and Pb.
   - Malicious Process: Pb (disloyal lieutenant).
   - Objective: Pa should agree on the value of the loyal commander Pc, which is 0.
   - Challenge: Pa cannot distinguish between scenarios, making it impossible to make a correct decision.
3. **Scenario 2 (Figure 14.3(b)):**
   - Processes: Commander Pc, Lieutenants Pa and Pb.
   - Malicious Process: Pc (disloyal commander), Pb (loyal lieutenant).
   - Situation: Pa receives identical values from Pb and Pc.
   - Dilemma: Pa needs to agree with Pb, but distinguishing between scenarios is not possible.
   - Message Exchange: Further communication doesn't help as each process has already shared its information.
4. **In both scenarios:**
   - Pa receives different values from the other two processes.
   - Default values (0 or 1) lead to incorrect decisions in at least one of the scenarios.
5. **Conclusion:**
   - Agreement is impossible when n=3 processes and f=1 Byzantine process.

Proof:

$n=3, f=1$

$f \geq n/3 \Rightarrow$ cannot be solved

$n \leq 3f$

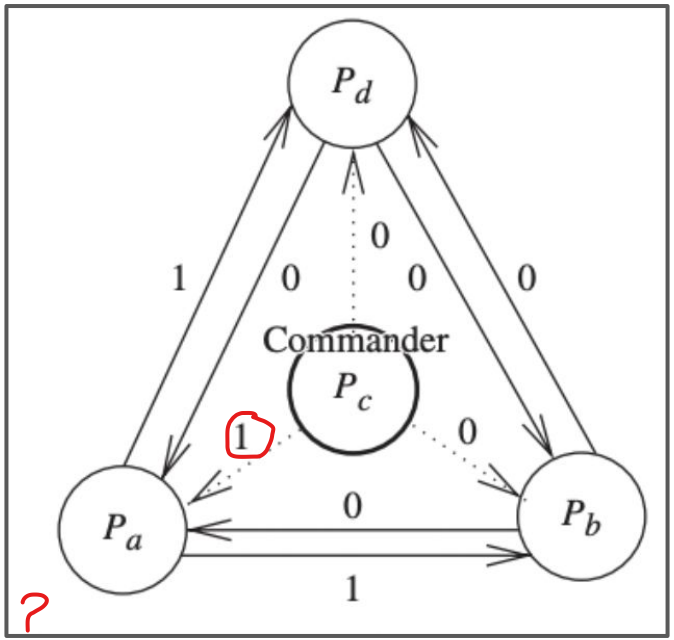


Proof Key Points:

- With n processes and $f \geq n/3$ processes, the Byzantine agreement problem cannot be solved. The correctness argument of this result can be shown using reduction. Let $Z(3, 1)$ denote the Byzantine agreement problem for parameters $n = 3$ and $f = 1$. Let $Z(n \leq 3f, f)$ denote the Byzantine agreement problem for parameters $n(\leq 3f)$ and f. A reduction from $Z(3, 1)$ to $Z(n \leq 3f, f)$ needs to be shown, i.e., if $Z(n \leq 3f, f)$ is solvable, then $Z(3, 1)$ is also solvable. After showing this reduction, we can argue that as $Z(3, 1)$ is not solvable, $Z(n \leq 3f, f)$ is also not solvable.

Byzantine agreement tree algorithm: exponential (synchronous system) - Recursive

$n = 4 ; f = 1$



@ P_a maj (P_c P_d P_b) (1 0 0) $\Rightarrow$ 0 //

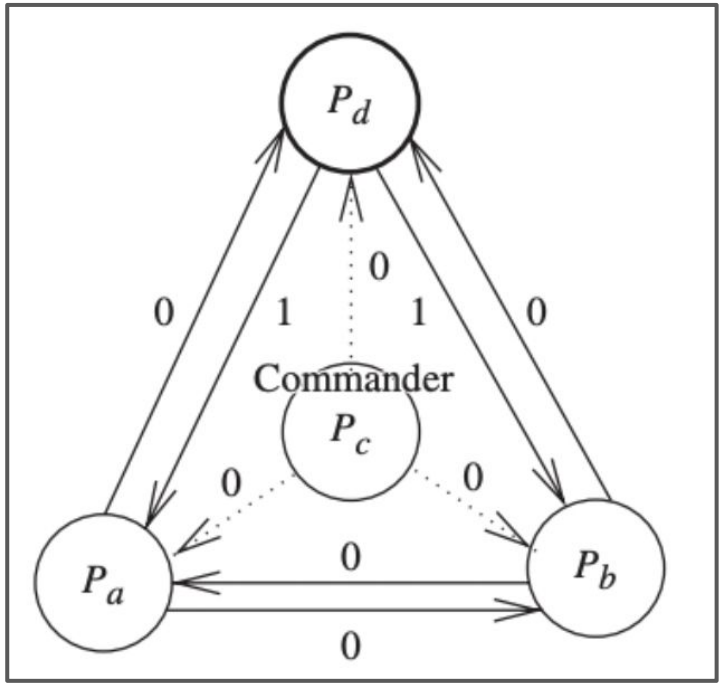
@ P_d maj (P_c P_a P_b) (0 1 0) $\Rightarrow$ 0

@ P_b :- maj (P_c P_d P_a) (0 0 1) $\Rightarrow$ 0 //

① commander sends own value

②

Byzantine agreement tree algorithm: exponential (synchronous system) - Recursive



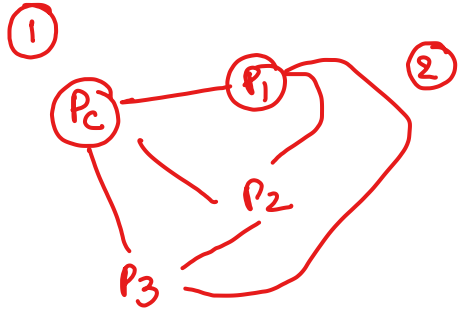
P_c P_d P_b

@ P_a maj(0 1 0) = 0

@ P_d → 0,0,0 ⇒ 0

@ P_b ⇒ (0,1,0) ⇒ 0

Simple Algorithm



1. Every process believes in majority.
2. Round 1:
   a. Sends out value to everyone. (commander)
   b. Every process receiving a value from commander sends it to processes it knows.(Unfolding)
3. Round 2:
   a. Each process takes majority of the values it received. (Folding)
   b. If none received => Uses default value.

(variables)

boolean: $v \longleftarrow$ initial value;

integer: $f \longleftarrow$ maximum number of malicious processes, $\leq \lfloor (n-1)/3 \rfloor$;

(message type)

$OM(v, Dests, List, faulty)$, where

v is a boolean,

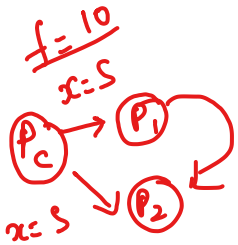
Dests is a set of destination process i.d.s to which the message is sent,

List is a list of process i.d.s traversed by this message, ordered from most recent to earliest,

faulty is an integer indicating the number of malicious processes to be tolerated.

Oral_Msg (f), where $f > 0$:

(1) The algorithm is initiated by the commander, who sends his source value v to all other processes using a $OM(v, N, \langle i \rangle, f)$ message. The commander returns his own value v and terminates.



(2) **[Recursion unfolding:]** For each message of the form $OM(v_j, Dests, List, f')$ received in this round from some process j, the process i uses the value v_j it receives from the source j, and using that value, acts as a *new* source. (If no value is received, a default value is assumed.)

To act as a new source, the process i initiates $Oral\_Msg\,(f'-1)$, wherein it sends
$OM(v_j, Dests - \{i\}, concat(\langle i \rangle, L), (f'-1))$
to destinations not in $concat(\langle i \rangle, L)$
in the next round.

(3) **[Recursion folding:]** For each message of the form $OM(v_j, Dests, List, f')$ received in step 2, each process i has computed the agreement value v_k, for each k not in $List$ and $k \neq i$, corresponding to the value received from P_k after traversing the nodes in $List$, at one level lower in the recursion. If it receives no value in this round, it uses a default value. Process i then uses the value $majority_{k \notin List, k \neq i}(v_j, v_k)$ as the agreement value and returns it to the next higher level in the recursive invocation.

$Oral\_Msg(0)$:

(1) **[Recursion unfolding:]** Process acts as a source and sends its value to each other process.

(2) **[Recursion folding:]** Each process uses the value it receives from the other sources, and uses that value as the agreement value. If no value is received, a default value is assumed.

Algorithm 14.2 Byzantine generals algorithm – exponential number of unsigned messages, $n > 3f$. Recursive formulation.

Table 14.3 Relationships between messages and rounds in the oral messages algorithm for the Byzantine agreement.

Round number	A message has already visited	Aims to tolerate these many failures	Each message gets sent to	Total number of messages in round
1	1	f	$n-1$	$n-1$
2	2	$f-1$	$n-2$	$(n-1)\cdot(n-2)$
...	...	...	...	...
x	x	$(f+1)-x$	$n-x$	$(n-1)(n-2)\ldots(n-x)$
$x+1$	$x+1$	$(f+1)-x-1$	$n-x-1$	$(n-1)(n-2)\ldots(n-x-1)$
...	...	...	...	...
$f+1$	$f+1$	0	$n-f-1$	$(n-1)(n-2)\ldots(n-f-1)$

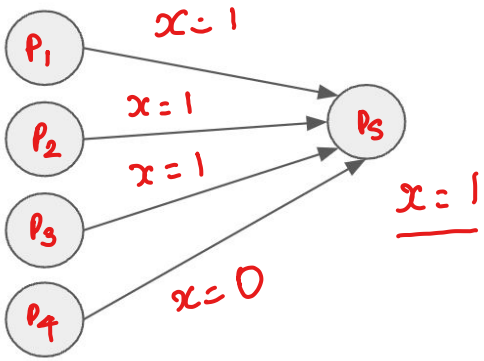
The recursive version of the algorithm is given in Algorithm 14.2. Each message has the following parameters: a consensus estimate value (v); a set of destinations ($Dests$); a list of nodes traversed by the message, from most recent to least recent ($List$); and the number of Byzantine processes that the algorithm still needs to tolerate ($faulty$). The list $L = \langle P_i, P_{k_1} \ldots P_{k_{f+1-faulty}} \rangle$ represents the sequence of processes (subscripts) in the knowledge expression $K_i(K_{k_1}(K_{k_2} \ldots K_{k_{f+1-faulty}}(v_0)\ldots))$. This knowledge is what $P_{k_{f+1-faulty}}$ conveyed to $P_{k_{f-faulty}}$ conveyed to $\ldots$ P_{k_1} conveyed to P_i who is conveying to the receiver of this message, the value of the commander ($P_{k_{f+1-faulty}}$)'s initial value.

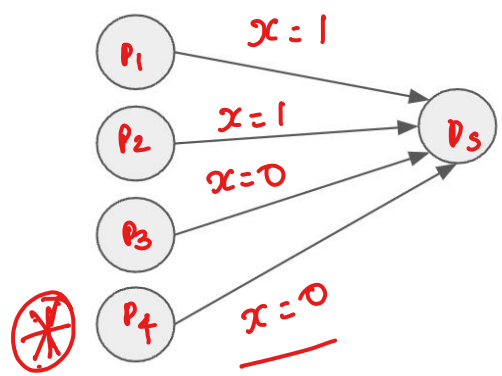
The commander invokes the algorithm with parameter $faulty$ set to f, the maximum number of malicious processes to be tolerated. The algorithm uses $f+1$ synchronous rounds. Each message (having this parameter $faulty = k$) received by a process invokes several other instances of the algorithm with parameter $faulty = k-1$. The terminating case of the recursion is when the parameter $faulty$ is 0. As the recursion folds, each process progressively computes the majority function over the values it used as a source for that level of invocation in the unfolding, and the values it has just computed as consensus values using the majority function for the lower level of invocations.

There are an exponential number of messages $O(n^f)$ used by this algorithm. Table 14.3 shows the number of messages used in each round of the algorithm, and relates that number to the number of processes already visited by any message as well as the number of destinations of that message.

Phase-king algorithm for consensus: polynomial (synchronous system)

$f < \lceil n/4 \rceil$





$\Rightarrow$ $f+1$ phases

$\Rightarrow$ 2 round

(variables)
boolean: $v \longleftarrow$ initial value;
integer: $f \longleftarrow$ maximum number of malicious processes, $f < \lceil n/4 \rceil$;
(1) Each process executes the following $f+1$ phases, where $f < n/4$:
(1a) **for** *phase* = 1 **to** $f+1$ **do**
(1b) Execute the following round 1 actions:
(1c) **broadcast** v to all processes;
(1d) **await** value v_j from each process P_j;
(1e) *majority* $\longleftarrow$ the value among the v_j that occurs $> n/2$ time
(default value if no majority);
(1f) *mult* $\longleftarrow$ number of times that *majority* occurs;
(1g) Execute the following round 2 actions:
(1h) **if** $i =$ *phase* **then**
(1i) **broadcast** *majority* to all processes;
(1j) **receive** *tiebreaker* from P_{phase} (default value if nothing is
received);
(1k) **if** *mult* $> n/2 + f$ **then**
(1l) $v \longleftarrow$ *majority*;
(1m) **else** $v \longleftarrow$ *tiebreaker*;
(1n) **if** *phase* $= f+1$ **then**
(1o) output decision value v.

Algorithm 14.4 Phase-king algorithm [4] – polynomial number of unsigned messages, $n > 4f$. Code is for process P_i, $1 \le i \le n$.

1. **Round 1:**
   - Processes broadcast their consensus value estimates and await values from others (lines 1b–1f).
   - At the round's end, each process counts "1" and "0" votes.
   - If either count > n/2, set majority variable to that consensus value, and set mult to the number of votes for the majority.
   - If neither count > n/2 (possible when malicious processes don't respond), use a default value for the majority variable.
2. **Round 2:**
   - Initiated by the phase king (process with identifier Pk for phase k, where $k \in [1, n]$).
   - Phase king broadcasts its majority value, acting as a tie-breaker for processes with mult < n/2 + f.
   - When a process receives the tie-breaker and its own mult < n/2 + f, update its estimate of the decision variable v to the value sent by the phase king.
   - Reason: Among its majority votes, f could be bogus, so it doesn't have a clear non-malicious majority (> n/2), thus adopts the value of the phase king.
   - If mult > n/2 + f (lines 1k–1l), update the estimate of the consensus variable v to its own majority value, regardless of the tie-breaker value sent by the phase king in the second round.

Correctness: ✓

f=3 f+1=4

1. Among the f +1 phases, the phase king of some phase k is non-malicious because there are at most f malicious processes.
2. As the phase king of phase k is non-malicious, all non-malicious processes can be seen to have the same estimate value v at the end of phase k.
    a. Specifically, observe that any two non-malicious processes Pi and Pj can set their estimate v in three ways:
        i. Both Pi and Pj use their own majority values.
        ii. Both Pi and Pj use the phase king's tie-breaker value.
        iii. Pi uses its majority value as the new estimate and Pj uses the phase king's tie-breaker as the new estimate
3. All non-malicious processes have the same consensus estimate x at the start of phase k+1 and they continue to have the same estimate at the end of phase k+1. This is self-evident because we have that $n > 4f$ and each non-malicious process receives at least $n-f > n/2+f$ votes for x from the other non-malicious processes in the first round of phase k+1.

LIKE

COMMENT

SHARE

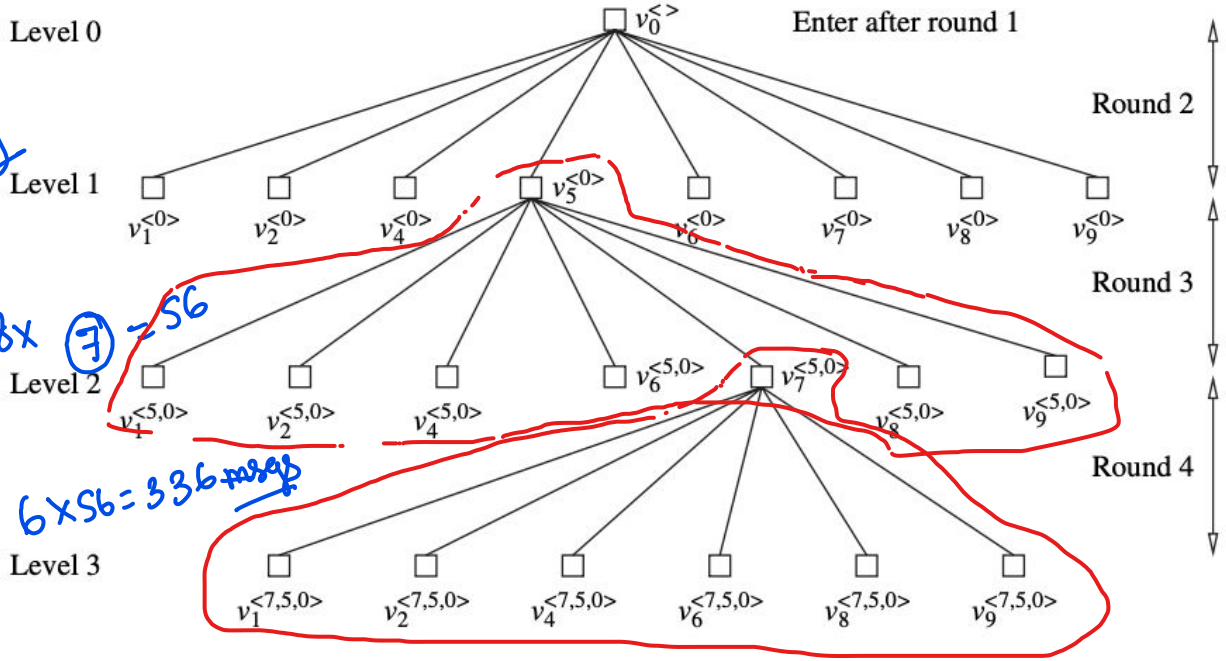
SUBSCRIBE

Consensus algorithms for Byzantine failures (synchronous system)

1. Upper bound on Byzantine processes
2. Byzantine agreement tree algorithm: exponential (synchronous system) Recursive formulation
3. **Byzantine agreement tree algorithm: exponential (synchronous system) Iterative formulation**
4. Phase King Algorithm

Example

Example Figure 14.5 shows the tree at a lieutenant node P_3, for $n = 10$ processes P_0 through P_9 and $f = 3$ processes. The commander is P_0. Only one branch of the tree is shown for simplicity. The reader is urged to work through all the steps to ensure a thorough understanding. Some key steps from P_3's perspective are outlined next, with respect to the iterative formulation of the algorithm.



An example of the majority computation is as follows:

- P_3 revises its estimate of $v_7^{\langle 5,0\rangle}$ by taking *majority* $(v_7^{\langle 5,0\rangle}, v_1^{\langle 7,5,0\rangle}, v_2^{\langle 7,5,0\rangle}, v_4^{\langle 7,5,0\rangle}, v_6^{\langle 7,5,0\rangle}, v_8^{\langle 7,5,0\rangle}, v_9^{\langle 7,5,0\rangle})$. Similarly for the other nodes at level 2 of the tree.
- P_3 revises its estimate of $v_5^{\langle 0\rangle}$ by taking *majority* $(v_5^{\langle 0\rangle}, v_1^{\langle 5,0\rangle}, v_2^{\langle 5,0\rangle}, v_4^{\langle 5,0\rangle}, v_6^{\langle 5,0\rangle}, v_7^{\langle 5,0\rangle}, v_8^{\langle 5,0\rangle}, v_9^{\langle 5,0\rangle})$. Similarly for the other nodes at level 1 of the tree.
- P_3 revises its estimate of $v_0^{\langle\rangle}$ by taking *majority*$(v_0^{\langle\rangle}, v_1^{\langle 0\rangle}, v_2^{\langle 0\rangle}, v_4^{\langle 0\rangle}, v_5^{\langle 0\rangle}, v_6^{\langle 0\rangle}, v_7^{\langle 0\rangle}, v_8^{\langle 0\rangle}, v_9^{\langle 0\rangle})$. This is the consensus value.

Explanation

- **Round 1** P_0 sends its value to all other nodes. This corresponds to invoking *Oral_Msg* (3) in the recursive formulation. At the end of the round, P_3 stores the received value in $v_0^{\langle\rangle}$.
- **Round 2** P_3 acts as a source for this value and sends this value to all nodes except itself and P_0. This corresponds to invoking *Oral_Msg* (2) in the recursive formulation. Thus, P_3 sends 8 messages. It will receive a similar message from all other nodes except P_0 and itself; the value received from P_k is stored in $v_k^{\langle 0\rangle}$.
- **Round 3** For each of the 8 values received in round 2, P_3 acts as a source and sends the values to all nodes except (i) itself, (ii) nodes visited previously by the corresponding value, as remembered in the superscript list, and (iii) the direct sender of the received message, as indicated by the subscript. This corresponds to invoking *Oral_Msg* (1) in the recursive formulation. Thus, P_3 sends 7 messages for each of these 8 values, giving a total of 56 messages it sends in this round. Likewise it receives 56 messages from other nodes; the values are stored in level 2 of the tree.
- **Round 4** For each of the 56 messages received in round 3, P_3 acts a source and sends the values to all nodes except (i) itself, (ii) nodes visited previously by the corresponding value, as remembered in the superscript list, and (iii) the direct sender of the received message, as indicated by the subscript. This corresponds to invoking *Oral_Msg* (0) in the recursive formulation. Thus, P_3 sends 6 messages for each of these 56 values, giving a total of 336 messages it sends in this round. Likewise, it receives 336 messages, and the values are stored at level 3 of the tree. As this round is *Oral_Msg* (0), the received values are used as estimates for computing the majority function in the folding of the recursion.

Iterative Algorithm

$v_0^{L<>}$

Boolean : v ← initial value

integer : f ← no. of malicious process ; $f \leq \lfloor (n-1)/3 \rfloor$

tree of boolean:

* level 0 root is v_{init}^{L} ; $L = <>$
* level h nodes

(message type)

OM (v, Dests, List, Faulty)

(1) Commander sends message to all.

Send $OM(v, N-\{i\}, <P_i>, f)$ to $N-\{i\}$;

return (v)

(2) @ each lieutenant

(2a) for rnd = 0 to f do

(2b) for each arriving msg OM, do

(2c) receive OM_1 from P_{k_1}

(2d) $v_{head(L)}^{tail(L)} \leftarrow v$

(2e) send OM_2 to Dests - {i} if rnd < f;

(2f) for level = f-1 down to 0 do

$\forall$ node $v_x^L \in$ level

v_x^L = majority $(v_x^L, v_y^{concat(<x>, L)})$;

$[x \neq i ; x \notin L ; y \neq i ;$

$y \notin concat(<x>, L)]$

$OM_1 \Rightarrow OM(v, Dests, L = \{P_{k_1}, \ldots P_{k_{f+1-faulty}}, faulty)$

$OM_2 \Rightarrow OM(v, Dests - \{i\}, <P_i, \ldots, P_{k_{f+1-faulty}}, faulty-1)$

unfolding

folding

Tree

level = 2 to 0

Iterative Algorithm

(variables)
boolean: $v \longleftarrow$ initial value;
integer: $f \longleftarrow$ maximum number of malicious processes, $\leq \lfloor (n-1)/3 \rfloor$;
tree of boolean:

- level 0 root is v_{init}^{L}, where $L = \langle\rangle$;
- level h $(f \geq h > 0)$ nodes: for each v_j^L at level $h - 1 = sizeof(L)$, its $n - 2 - sizeof(L)$ descendants at level h are $v_k^{concat(\langle j \rangle, L)}$, $\forall k$ such that $k \neq j, i$ and k is not a member of list L.

(message type)
$OM(v, Dests, List, faulty)$, where the parameters are as in the recursive formulation.

(1) Initiator (i.e., commander) initiates the oral Byzantine agreement:
(1a) **send** $OM(v, N - \{i\}, \langle P_i \rangle, f)$ to $N - \{i\}$;
(1b) **return**(v).

(2) (Non-initiator, i.e., lieutenant) receives the oral message (OM):

(2a) **for** $rnd = 0$ **to** f **do**

(2b) **for** each message OM that arrives in this round, **do**

(2c) **receive** $OM(v, Dests, L = \langle P_{k_1} \ldots P_{k_{f+1-faulty}} \rangle, faulty)$ from P_{k_1};
// $faulty + rnd = f$; $|Dests| + sizeof(L) = n$

(2d) $v_{head(L)}^{tail(L)} \longleftarrow v$; // $sizeof(L) + faulty = f + 1$. fill in estimate.

(2e) **send** $OM(v, Dests - \{i\}, \langle P_i, P_{k_1} \ldots P_{k_{f+1-faulty}} \rangle, faulty - 1)$
to $Dests - \{i\}$ **if** $rnd < f$;

(2f) **for** $level = f - 1$ **down to** 0 **do**

(2g) **for** each of the $1 \cdot (n-2) \cdot \ldots (n - (level + 1))$ nodes v_x^L in level $level$, **do**

(2h) $v_x^L (x \neq i, x \notin L) = majority_{y \notin concat(\langle x \rangle, L); y \neq i} (v_x^L, v_y^{concat(\langle x \rangle, L)})$;

Algorithm 14.3 Byzantine generals algorithm – exponential number of unsigned messages, $n > 3f$. Iterative formulation. Code for process P_i.

Working:

- There are $f+1$ levels from level 0 through level f.
- Level 0 has one root node, $v_{init}^{\langle\rangle}$, after round 1.

- Level h, $0 < h \leq f$ has $1 \cdot (n-2) \cdot (n-3) \cdots (n-h) \cdot (n-(h+1))$ nodes after round $h+1$. Each node at level $(h-1)$ has $(n-(h+1))$ child nodes.
- Node v_k^L denotes the command received from the node $head(L)$ by node k which forwards it to node i. The command was relayed to $head(L)$ by $head(tail(L))$, which received it from $head(tail(tail(L)))$, and so on. The very last element of L is the commander, denoted P_{init}.
- In the $f+1$ rounds of the algorithm (lines 2a–2e of the iterative version), each level k, $0 \leq k \leq f$, of the tree is successively filled to remember the values received at the end of round $k+1$, and with which the process sends the multiple instances of the *OM* message with the fourth parameter as $f-(k+1)$ for round $k+2$ (other than the final terminating round).
- For each message that arrives in a round (lines 2b–2c of the iterative version), a process sets $v_{head(L)}^{tail(L)}$ (line 2d). It then removes itself from *Dests*, prepends itself to L, decrements *faulty*, and forwards the value v to the updated *Dests* (line 2e).
- Once the entire tree is filled from root to leaves, the actions in the folding of the recursion are simulated in lines 2f–2h of the iterative version, proceeding from the leaves up to the root of the tree. These actions are crucial – they entail taking the majority of the values at each level of the tree. The final value of the root is the agreement value, which will be the same at all processes.

Correctness

Loyal commander

Given f and x, if the commander process is loyal, then *Oral_Msg* (x) is correct if there are at least $2f + x$ processes.

This can easily be seen by induction on x:

- For $x = 0$, *Oral_Msg* (0) is executed, and the processes simply use the (loyal) commander's value as the consensus value.
- Now assume the above induction hypothesis for any x.
- Then for *Oral_Msg* $(x+1)$, there are $2f + x + 1$ processes including the commander. Each loyal process invokes *Oral_Msg* (x) to broadcast the (loyal) commander's value v_0 – here it acts as a commander for this invocation it makes. As there are $2f + x$ processes for each such invocation, by the induction hypothesis, there is agreement on this value (at all the honest processes) – this would be at level 1 in the local tree in the folding of the recursion. In the last step, each loyal process takes the majority of the direct order received from the commander (level 0 entry of the tree), and its estimate of the commander's order conveyed to other processes as computed in the level 1 entries of the tree. Among the $2f + x$ values taken in the majority calculation (this includes the commanders's value but not its own), the majority is loyal because $x > 0$. Hence, taking the majority works.

No assumption about commander

Given f, *Oral_Msg* (x) is correct if $x \geq f$ and there are a total of $3x+1$ or more processes.

This case accounts for both possibilities – the commander being malicious or honest. An inductive argument is again useful.

- For $x = 0$, *Oral_Msg* (0) is executed, and as there are no malicious processes $(0 \geq f)$ the processes simply use the (loyal) commander's value as the consensus value. Hence the algorithm is correct.
- Now assume the above induction hypothesis for any x.
- Then for *Oral_Msg* $(x+1)$, there are at least $3x+4$ processes including the commander and at most $x+1$ are malicious.

    - (Loyal commander:) If the commander is loyal, then we can apply the argument used for the "loyal commander" case above, because there will be more than $(2(f+1)+(x+1))$ total processes.
    - (Malicious commander:) There are now at most x other malicious processes and $3x+3$ total processes (excluding the commander). From the induction hypothesis, each loyal process can compute the consensus value using the majority function in the protocol.

LIKE

COMMENT

SHARE

SUBSCRIBE