Fault Tolerant Distributed Consensus

There is no conversation more boring than the one where everybody agrees.

Marco Aiello, Eirini Kaldeli

University of Groningen

Distributed Systems, 2009

Chapter 14: Consensus and Agreement

Ajay Kshemkalyani and Mukesh Singhal

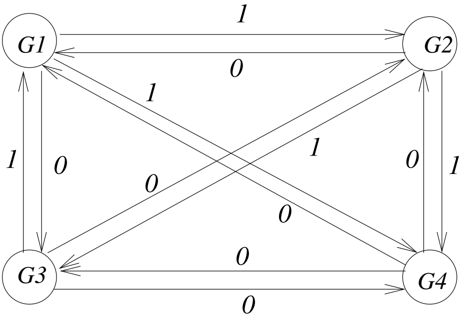
Distributed Computing: Principles, Algorithms, and Systems

Cambridge University Press

Assumptions

System assumptions

- Failure models
- Synchronous/ Asynchronous communication
- Network connectivity
- Sender identification
- Channel reliability
- Authenticated vs. non-authenticated messages
- Agreement variable



Problem Specifications

Byzantine Agreement (single source has an initial value)

- **Agreement:** All non-faulty processes must agree on the same value.
- **Validity:** If the source process is non-faulty, then the agreed upon value by all the non-faulty processes must be the same as the initial value of the source.
- **Termination:** Each non-faulty process must eventually decide on a value.

Consensus Problem (all processes have an initial value)

- **Agreement:** All non-faulty processes must agree on the same (single) value.
- **Validity:** If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value.
- **Termination:** Each non-faulty process must eventually decide on a value.

Interactive Consistency (all processes have an initial value)

- **Agreement:** All non-faulty processes must agree on the same array of values $A[v_1 \ldots v_n]$.
- **Validity:** If process i is non-faulty and its initial value is v_i, then all non-faulty processes agree on v_i as the ith element of the array A. If process j is faulty, then the non-faulty processes can agree on any value for $A[j]$.
- **Termination:** Each non-faulty process must eventually decide on the array A.

These problems are equivalent to one another! Show using reductions.

Overview of Results

Failure mode	Synchronous system (message-passing and shared memory)	Asynchronous system (message-passing and shared memory)
No failure	agreement attainable; common knowledge also attainable	agreement attainable; concurrent common knowledge attainable
Crash failure	agreement attainable $f < n$ ~~Byzantine~~ processes $\Omega(f+1)$ rounds	agreement not attainable
Byzantine failure	agreement attainable $f \leq \lfloor (n-1)/3 \rfloor$ Byzantine processes $\Omega(f+1)$ rounds	agreement not attainable

Table: Overview of results on agreement. f denotes number of failure-prone processes. n is the total number of processes.

In a failure-free system, consensus can be attained in a straightforward manner

Formal requirements for the consesus algorithm

An algorithm solves the consensus problem if it satisfies the following formal properties:

- ***Termination***: Eventually every non-faulty processor decides on a value y_i.
- ***Agreement***: The final decisions of all non-faulty processors are identical, i.e. if y_i, y_j are assigned then $\forall\ p_i, p_j \in Nonfaulty\ :\ (y_i = y_j)$.
- ***Validity***: If all non-faulty p_is have the same input then the decision of a non-faulty processor equals the common input, i.e. if $\forall\ p_i \in Nonfaulty\ :\ (x_i = v)$ then if y_i is assigned for some non-faulty p_j then $y_i = v$.

Systems with different levels of synchrony or different kinds of failures require different algorithms.

Impossibility of Distributed Consensus with one faulty process [Fisher, Lynch and Peterson, 1985]

Bad news

The design of a consensus protocol that tolerates failures is impossible in asynchronous distributed systems.

- Impossibility holds for both shared memory and message passing systems:
  - even if we assume reliable communication channels.
  - even if considering only benign failures (crashes).
  - even if at most one processor fails.
- The problem is that in totally asynchronous systems we cannot distinguish a dead process from a merely slow one.
- We have to assume some level of synchrony on communication, processes or message order for consensus to become possible.

Impossibility Result (MP, async)

FLP Impossibility result

Impossible to reach consensus in an async MP system even if a single process has a crash failure

- In a failure-free async MP system, initial state is *monovalent* $\Longrightarrow$ consensus can be reached.
- In the face of failures, initial state is necessarily bivalent
- Transforming the input assignments from the all-0 case to the all-1 case, there must exist input assignments $\vec{I_a}$ and $\vec{I_b}$ that are 0-valent and 1-valent, resp., and that differ in the input value of only one process, say P_i. If a 1-failure tolerant consensus protocol exists, then:
  - Starting from $\vec{I_a}$, if P_i fails immediately, the other processes must agree on 0 due to the termination condition.
  - Starting from $\vec{I_b}$, if P_i fails immediately, the other processes must agree on 1 due to the termination condition.

  However, execution (2) looks identical to execution (1), to all processes, and must end with a consensus value of 0, a contradiction. Hence, there must exist at least one bivalent initial state.
- Consensus requires some communication of initial values.

Impossibility Result (MP, async)

- To transition from bivalent to monovalent step, must exist a critical step which allows the transition by making a decision
- Critical step cannot be local (cannot tell apart between slow and failed process) nor can it be across multiple processes (it would not be well-defined)
- Hence, cannot transit from bivalent to univalent state.

Wider Significance of Impossibility Result

- By showing reduction from consensus to problem X, then X is also not solvable under same model (single crash failure)
- E.g., leader election, terminating reliable broadcast, atomic broadcast, computing a network-wide global function using BC-CC flows, transaction commit.

Consensus in synchronous systems

Fault-tolerant consensus can be reached in synchronous systems under certain assumptions on the number of faulty processors and the connectivity of the communication graph.
We will consider a model that can accommodate for process failures:

- The system includes at most f faulty processors: *f-resilient*.
- The subset F of the faulty processors (maybe different in each execution) is not known in advance.
- Communication channels are reliable (compare with the two generals problem).
- The graph topology is a complete graph.

A solely crash tolerant consensus algorithm

Code for each processor p_i, $1 \leq i \leq n$:

```
V={x_i}                          /*set V constains p_i's input*/
for (k:=1 to f+1)                /*round k*/
 broadcast(u ∈ V : p_i has not already sent u)
 receive set of msgs S_j from p_j, 1 ≤ j ≤ n, j ≠ i
 V:=V ∪ ⋃_{j=1}^{n} S_j          /*update V by joining it with the received sets*/

y_i=majority(V)                  /*decide at f+1 round*/
```

Correctness of the algorithm:

- *Termination*: The algorithm requires exactly $f+1$ rounds.
- *Validity*: The decision value is an input of some p_i, since no spurious messages are introduced: if all inputs have the same value, then that is the only one ever in circulation.

Agreement condition of the crash tolerant consensus algorithm

Agreement: At the end of round $f+1$,
$\forall\, p_i, p_j \notin F: \ (x \in V_i \ \Rightarrow \ x \in V_j)$: prove by contradiction.

Proof.

- Suppose $\exists x: \ (x \in V_i) \bigwedge (x \notin V_j)$, where p_i, p_j non-faulty.
- p_i must have received x for the first time at round $f+1$, otherwise it would have already sent it to p_j.
- There is a $p_{i_{f+1}}$ that sent x to p_i at round $f+1$. $p_{i_{f+1}}$ must have crashed in middle of this round, so x was not sent to p_j.
- Similarly, there is a p_{i_f} that sent x to $p_{i_{f+1}}$.
- So, there is a chain of $f+1$ distinct faulty processors $p_{i_1}, \ldots, p_{i_{f+1}}$ (remember that after a crash there is no resurrection), that transfered x to p_i ➠ Contradiction.

□

Complexity of crash-tolerant consensus

- Number of messages sent: $O(n^2)$. There are at most n different values and each of them is sent at most $n - 1$ times.
- Number of rounds: $f + 1$

➢ It can be proved that $f + 1$ is the lower bound on rounds for reaching fault-tolerant consensus (both for the benign and the severe case).

- Note that the algorithm is correct no matter how many the faulty processors are.

Traitors trying to spoil consensus

- There are n generals who head different divisions of the Byzantine army and have to agree whether to attack the enemy or not.
- Communication is reliable but f of the generals are traitors and try to bring confusion by feeding incorrect information.
- How many traitors can a byzantine consensus protocol tolerate?

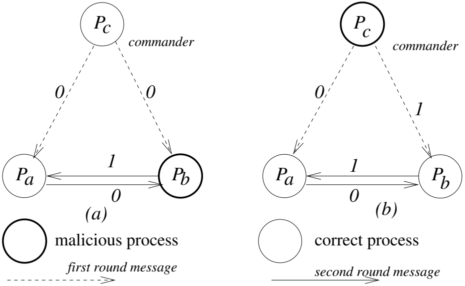
Theorem

In a system with three processors one of which are byzantine, there is no algorithm that solves the consensus problem.

☺ Let's see why.

Upper Bound on Byzantine Processes (sync)

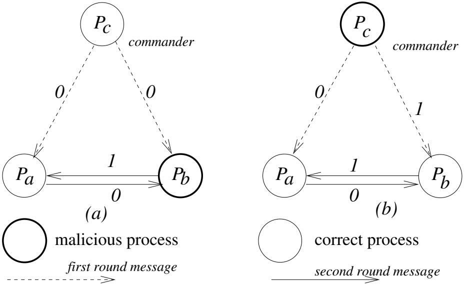
Agreement impossible when $f = 1, n = 3$.



- Taking simple majority decision does not help because loyal commander P_a cannot distinguish between the possible scenarios (a) and (b);
- hence does not know which action to take.
- Proof using induction that problem solvable if $f \leq \lfloor \frac{n-1}{3} \rfloor$. See text.

Upper Bound on Byzantine Processes (sync)

Agreement impossible when $f = 1, n = 3$.



- Taking simple majority decision does not help because loyal commander P_a cannot distinguish between the possible scenarios (a) and (b);
- hence does not know which action to take.
- Proof using induction that problem solvable if $f \leq \lfloor \frac{n-1}{3} \rfloor$. See text.

Lower bound on the ratio of faulty processors to achieve byzantine consensus

Theorem

In a system with n processors f of which are byzantine, there is no algorithm that solves the consensus problem if $n \leq 3f$ (even if the network is synchronous and complete).

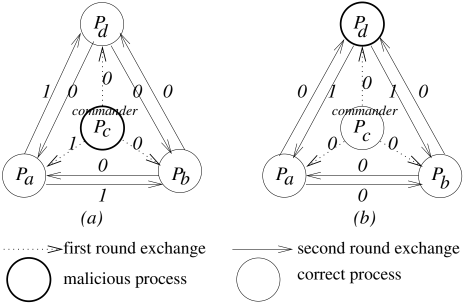
We can show that if we assume that there is such an algorithm, then we would be able to solve the problem for $n = 3$, $f = 1$ contradicting the previous theorem.

- What if the graph is not complete?
- It can be proved that for byzantine consensus to be possible the connectivity of the graph has to be at least $2f + 1$.

An algorithm with exponential message size

- ✓ We will present a byzantine consenus algorithm that is optimal in terms of resilience ($f < \frac{n}{3}$) and number of rounds (f+1).
- ✗ However, the size of the exchanged messages is exponential.
- Each p_i maintains a tree data structure of height $f + 1$ (levels 0 to $f + 1$).
- The algorithm consists of two phases:
  1. Information gathering: Values are filled in the tree level by level during the $f + 1$ rounds.
  2. Decision phase: Each p_i calculates its decision based on the values in its tree.

Consensus Solvable when $f = 1, n = 4$



- There is no ambiguity at any loyal commander, when taking majority decision
- Majority decision is over 2nd round messages, and 1st round message received directly from commander-in-chief process.

Byzantine Generals (recursive formulation), (sync, msg-passing)

(variables)
boolean: $v \longleftarrow$ initial value;
integer: $f \longleftarrow$ maximum number of malicious processes, $\leq \lfloor (n-1)/3 \rfloor$;
(message type)
Oral_Msg(v, *Dests*, *List*, *faulty*), where
v is a boolean,
Dests is a set of destination process ids to which the message is sent,
List is a list of process ids 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, 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.

Relationship between # Messages and Rounds

round number	a message has already visited	aims to tolerate these many failures	and 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	$(f+1)-x$	$n-x$	$(n-1)(n-2)\dots(n-x)$
$x+1$	$x+1$	$(f+1)-x-1$	$n-x-1$	$(n-1)(n-2)\dots(n-x-1)$
$f+1$	$f+1$	0	$n-f-1$	$(n-1)(n-2)\dots(n-f-1)$

Table: Relationships between messages and rounds in the Oral Messages algorithm for Byzantine agreement.

Complexity: $f+1$ rounds, exponential amount of space, and

$$(n-1)+(n-1)(n-2)+\ldots+(n-1)(n-2)..(n-f-1)\text{messages}$$

Bzantine Generals (iterative formulation), Sync, Msg-passing

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

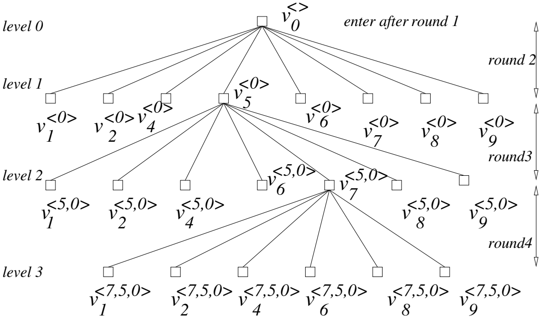
- level 0 root is v_{init}^{L}, where $L = \langle \rangle$;
- level $h(f \geq h > 0)$ nodes: for each v_j^L at level $h - 1 = sizeof(L)$, its $n - 2 - sizeof(L)$ descendants at level h are $v_k^{concat(\langle j \rangle, L)}$, $\forall k$ such that $k \neq j, i$ and k is not a member of list L.

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

(1) Initiator (i.e., Commander) initiates 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 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};
// $faulty + round = f$; $|Dests| + sizeof(L) = n$
(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) $v_x^L(x \neq i, x \notin L) = majority_{y \notin concat(\langle x \rangle, L); y \neq i}(v_x^L, v_y^{concat(\langle x \rangle, L)})$;

Tree Data Structure for Agreement Problem (Byzantine Generals)



Some branches of the tree at P_3. In this example, $n = 10, f = 3$, commander is P_0.

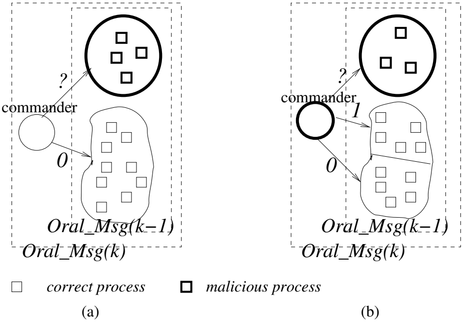
- (round 1) P_0 sends its value to all other processes using $Oral\_Msg(3)$, including to P_3.
- (round 2) P_3 sends 8 messages to others (excl. P_0 and P_3) using $Oral\_Msg(2)$. P_3 also receives 8 messages.
- (round 3) P_3 sends $8 \times 7 = 56$ messages to all others using $Oral\_Msg(1)$; P_3 also receives 56 messages.
- (round 4) P_3 sends $56 \times 6 = 336$ messages to all others using $Oral\_Msg(0)$; P_3 also receives 336 messages. The received values are used as estimates of the majority function at this level of recursion.

Exponential Algorithm: An example

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.

Impact of a Loyal and of a Disloyal Commander



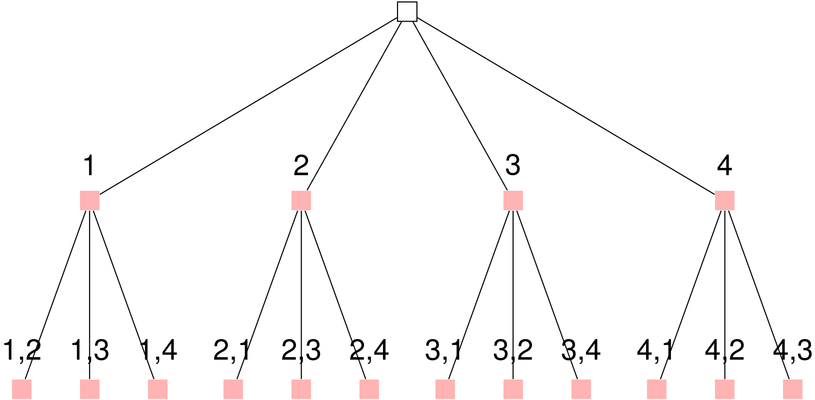
The effects of a loyal or a disloyal commander in a system with $n = 14$ and $f = 4$. The subsystems that need to tolerate k and $k-1$ traitors are shown for two cases. (a) Loyal commander. (b) No assumptions about commander.

(a) the commander who invokes *Oral_Msg(x)* is loyal, so all the loyal processes have the same estimate. Although the subsystem of $3x$ processes has x malicious processes, all the loyal processes have the same view to begin with. Even if this case repeats for each nested invocation of *Oral_Msg*, even after x rounds, among the processes, the loyal processes are in a simple majority, so the majority function works in having them maintain the same common view of the loyal commander's value.

(b) the commander who invokes *Oral_Msg(x)* may be malicious and can send conflicting values to the loyal processes. The subsystem of $3x$ processes has $x-1$ malicious processes, but all the loyal processes do not have the same view to begin with.

EIG algorithm: The tree structure

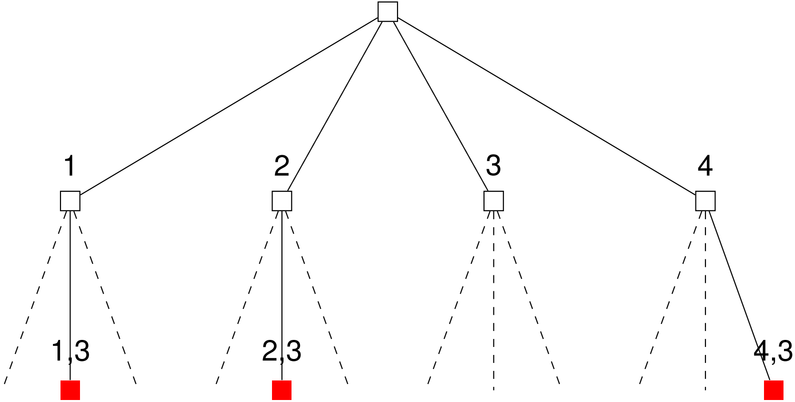
The EIG tree data structure with $n = 4$ and $f = 1$: the labels of nodes



- Initially each p_i stores its input in the root (level 0).
- Round $1 \leq r \leq f + 1$, each p_i:
  - broadcasts all nodes of the $r - 1$th level of its tree.
  - fills in level r: when it receives a message from p_j with the value of the node labeled $v = i_1, \ldots, i_k$, it stores it to the node labeled v,j of its tree (if a value for a node is not received, then default $u_\perp$ is stored).
- So, p_i stores in node $i_1, \ldots, i_k,j$ the value that "p_j says that p_{i_k} says that ... that p_{i_1} said".

The EIG algorithm: Information gathering

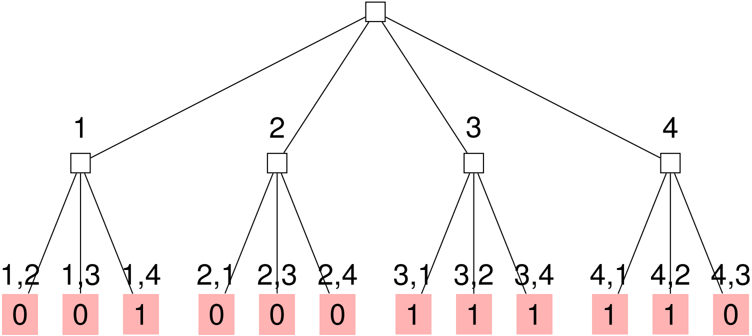
The tree is filled in from the root to the leaves, level by level.
Information received from p_3 in round 2 is stored at level 2:



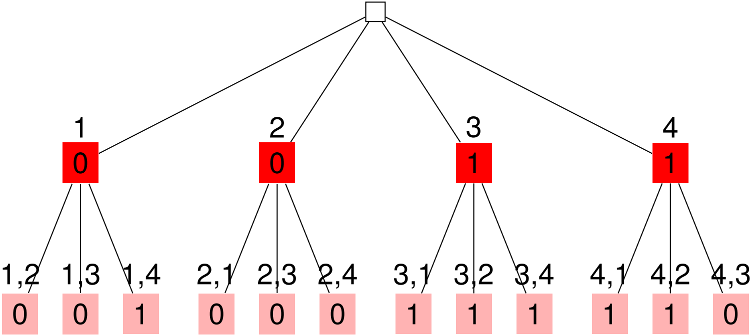
- At round $f + 1$ the entire tree has been filled in. Node labeled with sequence π has value $tree(\pi)$.
- Each p_i applies to each subtree with root π a recursive reduction function (usually majority vote) $resolve_i(\pi)$.
- The decision value is the resolved value of the root, $resolve()$, which is computed recursively based on the following definition:

$$resolve(\pi) = \begin{cases} tree(\pi) \text{ if } \pi \text{ is a leaf} \\ majority\{resolve(\pi'),\ \pi' : child\ of\ \pi\} \text{ otherwise} \\ (u_\perp \text{ if no majority exists}) \end{cases}$$

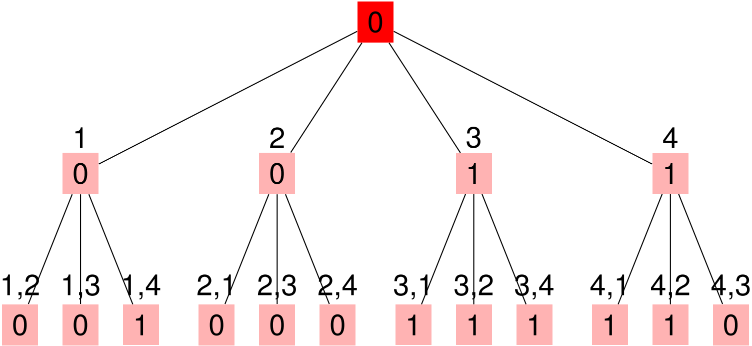
Start from the leaves, and compute the resolved value level by level till the root. Assume binary consensus (input $\in \{0, 1\}$). Default value is 0.



Start from the leaves, and compute the resolved value level by level till the root. Assume binary consensus (input $\in \{0, 1\}$). Default value is 0.



Start from the leaves, and compute the resolved value level by level till the root. Assume binary consensus (input $\in \{0, 1\}$). Default value is 0.



Correctness of the EIG algorithm: Validity

Validity can be proved based on the following lemma that says that the resolved values are consistent.

Lemma 1

For each non-faulty p_i, the resolved value of a node $\pi = \pi' j$ corresponding to a non-faulty p_j is the value that p_j had stored in its π' node: $resolve_i(\pi) = tree_j(\pi')$.

☺ Let's proove it (induction on the height of the tree).

- Suppose all non-faulty p_is start with initial value v.
- The decision value of p_i is $resolve_i() = majority\{resolve_i(j),\ j : child\ of\ root\}$
- By lemma 1 we have that $\forall\ non\text{-}faulty\ j:\ (resolve_i(j) = tree_j() = v)$.
- Since the majority of p_is are non-faulty $resolve_i() = v$.

To proove agreement we need to introduce some additional terms.

- A node π is ***common*** if
  $\forall p_i, p_j \in \mathit{Nonfaulty} : \ (\mathit{resolve}_i(\pi) = \mathit{resolve}_j(\pi))$.
- a node π has a ***common frontier*** if every path from π to a leaf contains a common node.

Lemma 2

If a node π has a common frontier then π is common.

☺ Let's proove it (induction on height of the tree, by contradiction).

Correctness of the EIG algorithm: Agreement

Agreement can be proven based on lemma 2.

- The nodes on each path from a node at level 1 to a leaf correspond to $f + 1$ different processors.
- So, at least one such node π corresponds to a non-faulty p_j, and by lemma 1 its resolved values are consistent (equal to $tree_j(\pi')$, where $\pi = \pi' j$), and thus it is common.
- Thus, the root has a common frontier since every path from the root to the leaves includes a common node.
- By lemma 2 the root is common, meaning that all non-faulty processors resolve the same decision value.

Complexity of the EIG algorithm

The EIG algorithm uses:

- $f + 1$ rounds (optimal)
- $n \geq 3f + 1$ processors (optimal)
- exponential size messages (sub-optimal):
  - At each round r every process bradcasts the whole level r of its tree.
  - At $r = 1$ each p_i broadcasts one value, at $r = 2$ n values, at $r = 3$ $n(n-1)$ values and at $r = k$ $n(n-1)(n-2)\ldots(n-(r-2))$ values.
  - The largest message corresponds to $r = f + 1$: $n(n-1)\ldots(n-(f+1)) = O(n^f)$.
- $n^2(f + 1)$ number of messages (sub-optimal)

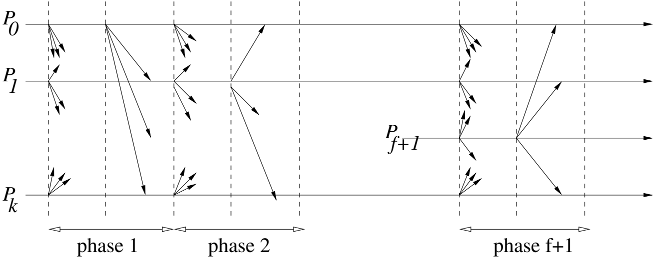
The Berman-Garray algorithm

- ✓ We will present an algorithm that uses message of $O(1)$ size.
- ✗ However, at the cost of $2(f+1)$ rounds and with the requirement that $n > 4f$.
- The algorithm contains $f+1$ phases, each taking 2 rounds.
- Each p_i has a *preference* at each phase, which is initially its input value and becomes its decision at phase $f+1$.
- At each phase k p_k is said to be the *king* of the phase.

The Phase King Algorithm

Operation

- Each round has a unique "phases king" derived, say, from PID.
- Each round has two phases:
- in 1st phase, each process sends its estimate to all other processes.
- in 2nd phase, the "Phase king" process arrives at an estimate based on the values it received in 1st phase, and broadcasts its new estimate to all others.



The Phase King Algorithm: Code

(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: // actions in round one of each phase
(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 $> n/2$ times (default if no maj.);
(1f) *mult* $\longleftarrow$ number of times that *majority* occurs;
(1g) Execute the following Round 2 actions: // actions in round two of each phase
(1h) **if** i = *phase* **then** // only the phase leader executes this send step
(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.

The Phase King Algorithm

- $(f+1)$ rounds, $(f+1)[(n-1)(n+1)]$ messages, and can tolerate up to $f < \lceil n/4 \rceil$ malicious processes

Correctness Argument

- Among $f+1$ rounds, at least one round k where phase-king is non-malicious.
- In round k, all non-malicious processes P_i and P_j will have same estimate of consensus value as P_k does.
  - P_i and P_j use their own majority values (Hint: $\Longrightarrow P_i$'s $mult > n/2 + f$)
  - P_i uses its majority value; P_j uses phase-king's tie-breaker value.
  - P_i and P_j use the phase-king's tie-breaker value. (Hint: In the round in which P_k is non-malicious, it sends same value to P_i and P_j)

  In all 3 cases, argue that P_i and P_j end up with same value as estimate
- If all non-malicious processes have the value x at the start of a round, they will continue to have x as the consensus value at the end of the round.

The Berman-Garray algorithm: Code for $p_i,\ 0 \leq i \leq n-1$

`pref[`i`]=`x_i`, pref[`$j = v_\perp$`] for any` $0 \leq j \leq n-1,\ j \neq i$

`Round` $2k-1,\ 1 \leq k \leq f+1$:

- `broadcast(pref[`i`])`
- `receive` v_j `from` p_j
- $\forall j : 0 \leq j \leq n-1, j \neq i$ `pref[`j`]:=`v_j
- maj`:=majority{pref[0],...,pref[`$n-1$`]}`
- $mult$`=multiplicity of` maj /*#procs that voted for maj*/

`Round` $2k,\ 1 \leq k \leq f+1$:

- `if` $i = k$ `then broadcast(`maj`)` //p_i is the king
- `receive(`$king\_maj$`) from` p_k
- `if (`$mult > \frac{n}{2} + f$`)`
  - `then pref[`i`]:=`maj
- `else pref[`i`]:=`$king\_maj$
- `if (`$k = f+1$`) then decision:=pref[`i`]`

Correctness of the Berman-Garray algorithm: Validity

Lemma 3

If all nonfaulty processes prefer v at the beginning of phase k, then they all prefer v at the end of phase k, for all $1 \leq k \leq f + 1$.

Proof sketch:

- Each p_i receives at least $n - f$ copies of v (including its own) in the first round of phase k.
- Because of the assumption $n > 4f \Rightarrow n/2 > 2f \Rightarrow n > n/2 + 2f \Rightarrow n - f > n/2 + f$.
- Thus, all nonfaulty p_is will prefer v at the end of phase k.

Validity follows by lemma 3: If all nonfaulty p_is start with v, they continue to prefer v throughout the phase.

- Observation: There are at most f faulty p_is, and $f + 1$ phases, so at least one phase has a nonfaulty king.

Lemma 4

Let g be a phase whose king p_g is nonfaulty. Then all nonfaulty p_is finish phase g with the same preference.

Proof sketch:

- Case 1: Suppose all nonfaulty p_is use king's majority for their preference. Since the king is nonfaulty it sends everyone the same value.

Proof sketch of lemma 4 continued:

- Case 2: Suppose that some nonfaulty p_j uses its own majority value v for its preference. This means that p_j has received more than $n/2 + f$ votes for v in the first round of g. Thus, every processor, including the king p_g, has received more than $n/2$ votes for v in the first round of g, and sets its majority value to v.

➢ Agreement follows by lemma 4: At phase $g + 1$ all nonfaulty p_is start with the same preference and by lemma 3 this agreement persists.