CSE535 Asynchronous Systems

Consensus and Agreement Algorithms

YoungMin Kwon

Agreement

- Processes in a distributed system needs to reach a common agreement before taking actions.
- E.g. Transaction
  - All processes either commit or abort a transaction

Failure Models

- Among n processes in the system at most f processes can be faulty
- Fail-stop
  - A process may fail in the middle of a step
  - It may send a message to only a subset of destination set before crashing
- Byzantine failure
  - A process may behave arbitrarily

Asynchronous Communication

- Suppose that a process p_i expects a message from a process p_j
  - p_i cannot tell whether a non-arrival of a message is due to a failure in p_j or a long message delay
- Impossibility of reaching an agreement in asynchronous system in any failure model

Other Assumptions

- Sender Identification
  - Receiver of a message always knows the identity of the sender
  - Even with Byzantine behavior
- Channel reliability
  - The channels are reliable and only the processes may fail

Byzantine War

- Multiple armies camp around the fort of Byzantine
- The attack is successful only when they attack together
- Generals send messengers to agree on the time of attack
  - Generals can be a traitor and may send a wrong time
  - Messengers can be caught

Byzantine Agreement Problem

- The problem
  - Designated process, called the source process, with an initial value
  - To reach agreement with other processes about its initial value

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

Consensus Problem

- The problem
  - Each process has an initial value
  - All the correct process must agree on a single value

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

Interactive Consistency Problem

- The problem
  - Each process has an initial value
  - All the correct processes must agree upon a set of values with one value for each process.

- Agreement
  - All non-faulty processes must agree on the same array of values A[1..n]
- Validity
  - If p_i is non-faulty and its initial value is v_i, then all non-faulty processes has v_i on A[i].
  - If p_j is faulty, all non-faulty processes agree on any value for A[j]
- Termination
  - Each non-faulty process must eventually decide on the array A

Agreement in a Failure-Free System

- As simple as broadcasting a message with the initial value

Consensus Algorithm for Crash Failures

(global constants)
integer: f; // maximum number of crash failures tolerated
(local variables)
integer: $x \longleftarrow$ local value;

(1) Process P_i $(1 \leq i \leq n)$ executes the consensus algorithm for up to f crash failures:
(1a) **for** *round* **from** 1 **to** $f+1$ **do**
(1b) **if** the current value of x has not been broadcast **then**
(1c) **broadcast**(x);
(1d) $y_j \longleftarrow$ value (if any) received from process j in this round;
(1e) $x \longleftarrow min_{\forall j}(x, y_j)$;
(1f) **output** x as the consensus value.

n processes, up to f process may fail

Consensus Algorithm for Crash Failures : Correctness

- Agreement
  - In the f+1 rounds, there is at least one round in which no process failed
- Validity
  - No process sends a fictitious value
- Termination
  - The algorithm runs for f+1 rounds.

Complexity

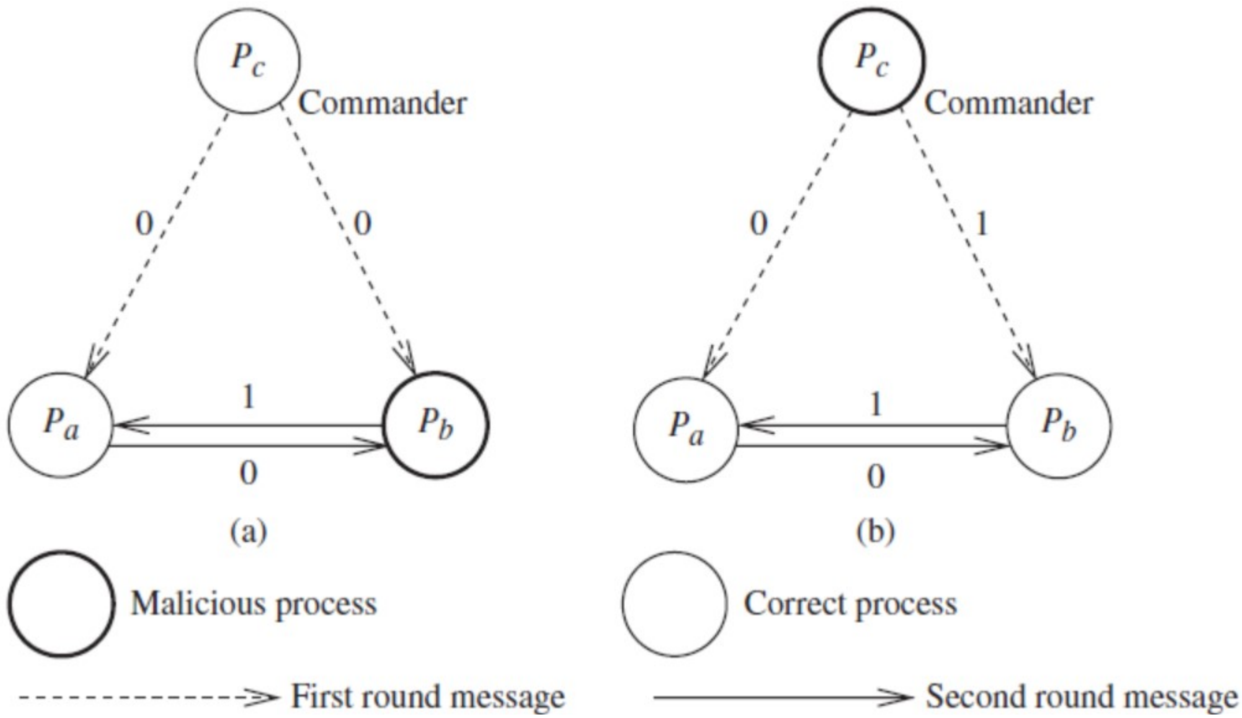
- The number of messages in each round is at most $O(n^2)$
- There are f+1 rounds.
- Total $O((f+1) \times n^2)$ messages

Consensus Algorithms for Byzantine Failures

- Upper bound on Byzantine processes
  - With n processes, the number of Byzantine processes f should satisfy $f \leq \lfloor \frac{n-1}{3} \rfloor$

Upper bound on Byzantine processes

- With n = 3 and f = 1, the Byzantine agreement problem cannot be solved



Upper bound on Byzantine processes

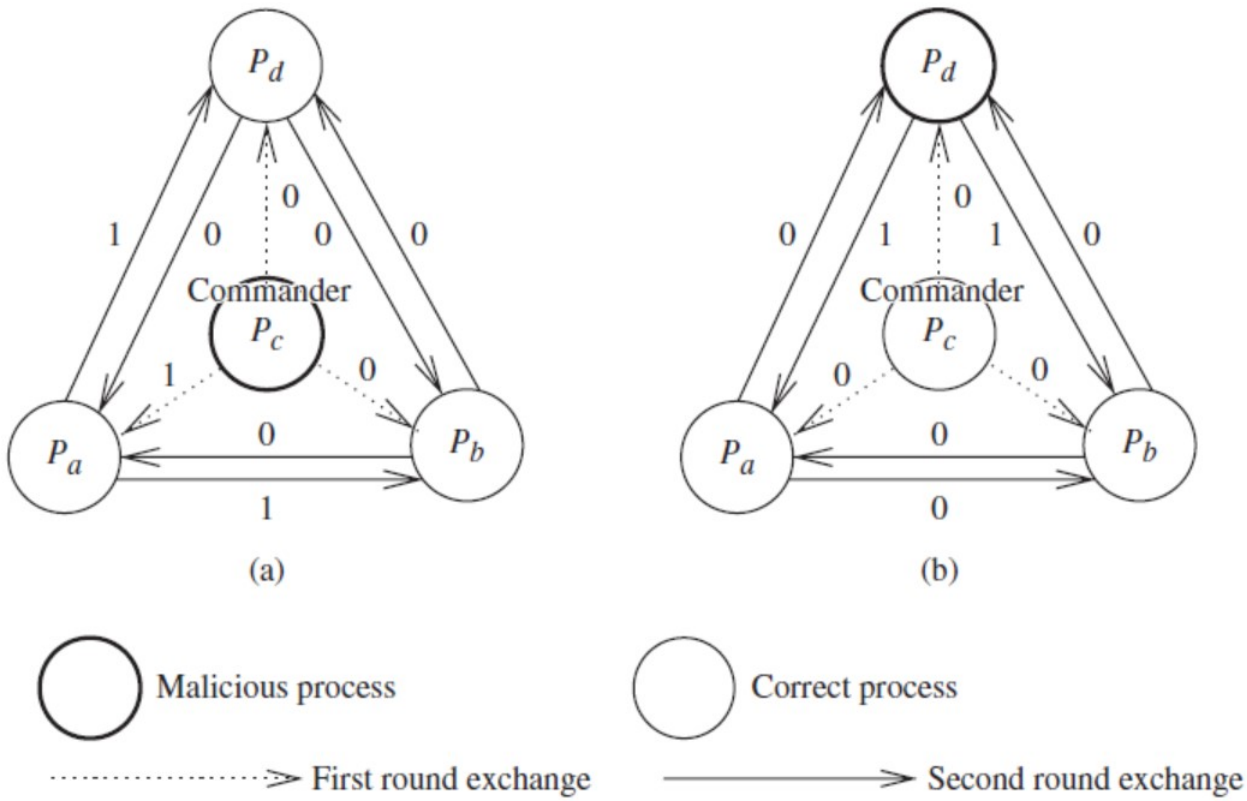
- Definitions
  - Let Z(3,1) denote the Byzantine agreement problem with n=3 and f=1
  - Let Z($n \leq 3f$,f) denote the problem with n ($\leq 3f$) and f.

- Proof overview
  - A reduction from Z(3,1) to Z($n \leq 3f$,f) will be shown.
  - Because Z(3,1) is not solvable, Z($n \leq 3f$,f) is not solvable either.

Upper bound on Byzantine processes

- In $Z(n \leq 3f, f)$, partition the n processes into three sets S_1, S_2, and S_3, each of size $\leq$ n/3
- In Z(3,1), P_1, P_2, and P_3 simulate the actions of the corresponding set S_1, S_2, and S_3
  - Simulate actions (send events, receive events, intra-set communication, inter-set communication)
- Because there is no algorithm for Z(3,1), no algorithm exists for $Z(n \leq 3f, f)$

Byzantine Agreement Tree Algorithm



Example with n=4, f=1

Byzantine Agreement Tree Algorithm

(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.

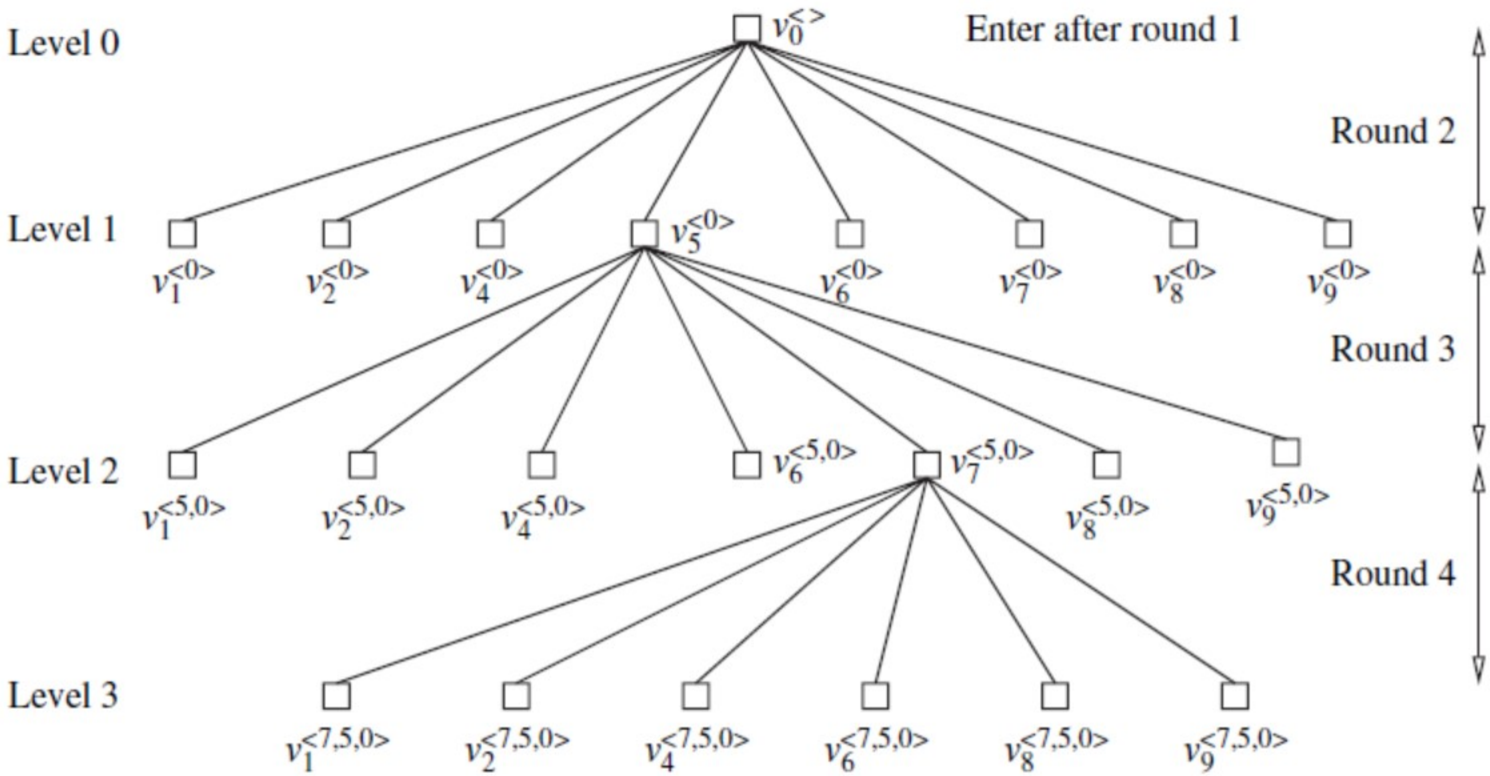
(3) **[Recursion folding:]** For each message of the form $OM(v_j, Dests, List, f')$ received in step 2, each process i has computed the agreement value v_k, for each k not in *List* and $k \neq i$, corresponding to the value received from P_k after traversing the nodes in *List*, at one level lower 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.

Byzantine Agreement Algorithm



Message tree from P_3's perspective (n=10, f=3)
Round 1: received 1 message from P_0,
Round 2: 8, Round 3: 56=7*8, Round 4:336=6*56

Byzantine Agreement Algorithm

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.

Correctness
(Loyal commander case)

- Oral_Msg(x) is correct if there are at least 2f+x processes
  - When x = 0, Oral_Msg(0) is executed and processes simply use the (loyal) commander's value as their consensus value
  - When x>0, let's assume the above as an induction hypothesis
  - For Oral_Msg(x+1), there are 2f+x+1 processes
    - Each loyal process invokes Oral_Msg(x); As there are 2f+x processes, by the induction, there is agreement (at loyal processes)
    - The majority taken on 2f+x values is loyal because x > 0

Correctness
(No assumption about the commander)

- Oral_Msg(x) is correct if $x \geq f$ and there are at least 3x+1 processes
- When x = 0, Oral_Msg(0) is executed with f=0.
- For Oral_Msg(x+1), there are at least 3x+4 processes
  - If the commander is loyal, because there will be more than 2(f+1) + (x+1) processes, we can apply the previous loyal commander case
  - If the commander is malicious, there are at most x traitors and 3x+3 total processes (excluding the commander). From the induction hypothesis, each loyal process can compute the consensus value using the majority function.

Asynchronous channels with failures

- Impossibility of reaching an agreement even with a single process crash failure (Fisher et al)

Asynchronous channels with failures

- v(GS), where GS is a global state:
  - The set of possible values that can be agreed upon in some global sate reachable from GS
- Valency: |v(GS)|
- A global state GS can be monovalent if |v(GS)|=1
  - 1-valent if v(GS) = {1}
  - 0-valent if v(GS) = {0}
- A global state GS can be bivalent if |v(GS)|=2
  - A 1-valent or 0-valent state can be reachable from a bivalent state

Asynchronous channels with failures

- Every correct consensus protocol has a bivalent initial state
  - Transforming the input assignment from the all 0 case to all 1 case,
    - there are input assignments I_a and I_b that are 0-valent and 1-valent, respectively, and they differ at only one process, say P_i
  - If a 1-crash-failure tolerant consensus protocol exists
    1) Staring from I_a, if P_i fails immediately, the other processes must agree on 0
    2) Staring from I_b, if P_i fails immediately, the other processes must agree on 1
  - Contradiction: execution 1) and 2) should be identical and they must agree on the same value

Asynchronous channels with failures

- Critical step
  - A step that moves from a bivalent state to a monovalent state
- In the face of a potential process crash, it is not possible to distinguish a crash or a long channel delay
  - It is not possible to take a critical step