Distributed Systems

Monsoon 2024

Lecture 10

International Institute of Information Technology
Hyderabad, India

Distributed Consensus

- Consider the following setting.
- There is one open position for a top coder for your start-up.
- Four of you interview a candidate independently.
- The four of you have a local decision on whether to hire this candidate or not.
- The four of you have to discuss and arrive at a common decision.
- This is the classical distributed consensus problem.

Distributed Consensus

- The problem of consensus is much more fundamental to distributed computing.
- Applications to
  - Distributed database transactions: Commit/Abort
  - Agreement on the reading reported by different sensors
  - Air traffic control system: all aircrafts must have the same view

Distributed Consensus

- Consider a distributed systems with n processes.
- Let us assume that there is logical connectivity between each pair of processes.
- These logical channels are assumed to be reliable.
- We will only consider the setting where messages are not authenticated.
  - So, message contents may be garbled, forged, etc.
  - Only the id of the sender is unforged through the network.
- The agreement variable is taken to be a Boolean value (0/1).

Distributed Consensus

- Two other significant assumptions on the system model:
- Asynchronous/Synchronous: We will consider both possibilities separately.
    - In the asynchronous case, if a message from a process P_i to P_j fails, then P_j cannot know the difference between the non-arrival of the message vs. a delayed arrival.
- Failure of processes: We consider that some of the processes may fail and in different ways:
    - Fail-stop
    - Byzantine
    - …

Distributed Consensus

- Reconsider the hiring problem again.
- If one of the four who interview the candidate have a reason to influence the decision, we model that as a faulty process.

Formal Definition

- The consensus problem is defined as follows.
- Consider a distributed system with n processes.
- Each process has an initial value. All the correct processes agree on a single value subject to:
- **Agreement**: All non-faulty processes must agree on a single value
- **Validity**: If all the non-faulty processes have the same initial value, then the eventual agreed value by all the non-faulty processes must be that same value.
- **Termination**: Each non-faulty process must eventually decide on a value.

Distributed Consensus

- Some observations
- The validity condition rules out trivial solutions such as agreeing on a default value in all cases.
- The faulty processes may or may not decide – these are not considered in the solution.

A Variant – Distributed Agreement

- The formal problem definition is as follows:
- In a distributed system with n processes, we have one process designated as the source process that has an initial value.
- The source process and other processes have to reach an agreement about the value subject to:
  - **Agreement**: All non-faulty processes must agree on the same value
  - **Validity**: If the source process is non-faulty, then the eventual agreed value by all the non-faulty processes must be the initial value of the source process.
  - **Termination**: Each non-faulty process must eventually decide on a value.

Distributed Agreement

- It will be seen later that both these problems are equivalent.
- One can be reduced to the other.

Quick Recap

- We identified two important problems.
- Distributed Consensus and Distributed Agreement
- Both reducible to each other.
- Main difference:
  - Consensus: Every node has an initial value.
  - Agreement: Only one node, the initiator, has an initial value.
- Properties of Solution: Agreement, Validity, Termination.

Consensus -- One Table to Fill

Fault/ Comm. Model	Fault-Free	Faulty	
		Fail-Stop	Byzantine
Synchronous			
Asynchronous			

The Simple Case – Failure Free System

- Consider a setting where there are no faulty processors.
- In a fault-free setting, each process can broadcast its value to other processes.
  - An All-to-All broadcast.
- The decision can be reached by having all the processes compute a common function such as min/max/average/majority on the n values.

The Simple Case – Failure Free System

- Consider a setting where there are no faulty processors.
- In a fault-free setting, each process can broadcast its value to other processes.
  - An All-to-All broadcast
- The decision can be reached by having all the processes compute a common function such as min/max/average/majority on the n values.
- In the case of a synchronous network, all this can be achieved in a constant number of rounds.
- In the case of an asynchronous network, there is a possibility. Defer details to later.

One Table to Fill

Fault/ Comm. Model	Fault-Free	Faulty	
		Fail-Stop	Byzantine
Synchronous	Easy, All-to-All communication		
Asynchronous	Possible, to do later		

The Simple Case

- Given the simple nature of the problem in fault-free systems, we will now move to faulty settings.

Consensus : The Case of Crash Failures

- Let us consider the next difficult case – that of crash failures.
- In the crash failure, or the fail-stop model, a process may crash at any time during execution.
    - Even during the middle of executing a step including send/receive.
- Let us consider a system of n processes with up to f of them being faulty.
- The value at each process is taken to be an integer.

Consensus in Fail-Stop Setting

- Process P_i has an initial value x_i.
- In each round, if P_i's value changed in the previous round, P_i sends its value x_i to all other processes.
- P_i takes the minimum of all the received values and updates x_i to this value.

- If there are f faulty processes in the system, then the algorithm runs for f+1 rounds.

Consensus in Fail-Stop Setting

The algorithm for Process P_i

Algorithm ConsensusFailStop

Begin

for round = 1 to f+1 do

begin

if the current value of x_i has not been sent yet

broadcast(x)

y_j = the value received from P_j

$x_i = \min \{ x_i, \min_j \{y_j\} \}$

end

output x_i as the consensus value

End

Example Run

- Consider a system of five nodes with initial values (1, 1, 0, 1, 1).
- The consensus value, if there are no failures, should be 0.
- Even if the value of 0 is sent to one node before P3 fails, then 0 is still the consensus value and is valid.
- Let us assume that there is one faulty node, say P3.
- Suppose that in round 1, P3 fails!
- Suppose P3 sends its value to any one of the other nodes before failing, say P2.

Example Run

- Suppose P3 sends its value to any one of the other nodes before failing, say P2.
- Now, P2 takes the minimum of 1 and 0, and resets its value to 0.
- All others do not know of the value of P3 and update their values to 1.
- In the second round, P3 no longer sends any messages.
- But P2 has a value that it has not sent yet!
- So, P2 sends 0 to every one.
- All nodes update their value to 0.

Example Run

- Suppose P3 crashes before sending its value to any other node.
- It is similar to a system with just 4 nodes!
- Then, the only value that others learn is 1. So, the result produced is 1.

Consensus in Fail-Stop Setting

- We will show that the algorithm satisfies all the three conditions required.
- Agreement: Need all non-faulty processors to agree on the same value.
- In each faulty round, at least one process may become faulty.
- So, in f+1 rounds, there is at least one round r that is fault-free, i.e., has no process failures.
- In that round r, all non-faulty processes broadcast their current value.
- This value is used to set the minimum by all other non-faulty processes.

Consensus in Fail-Stop Setting

- Agreement: Need all non-faulty processors to agree on the same value.
- In each round, at least one process may become faulty.
- So, in f+1 rounds, there is at least one round r that has no process failures.
- In round r, all non-faulty processes broadcast their current value.
- This value is used to set the minimum by all other processes. Say the minimum value is v.
- Post this, only v may be broadcast by processors at most one more time.

Consensus in Fail-Stop Setting

- Validity: Need all non-faulty processors to agree on the same initial value if all such processors indeed start with an identical value.
- Since faults are fail-stop, processors that fail in a round r do not propagate incorrect values.
- This also means that a process that crashes has only sent correct values till the round it crashed.
- If all processors start with an identical value v, then v is the only value that is ever sent by any process.

Consensus in Fail-Stop Setting

- Termination: Need all non-faulty processors to eventually decide.
- Since the algorithm runs only for f+1 rounds, this condition too is met.

Consensus in Fail-Stop Setting

- Complexity:
- Number of rounds = f+1.
- Number of message per round = $O(n^2)$.
- Number of messages = $O(f.n^2)$

- This bound is tight. There are examples where this bound is met.
- Consider round r, for r between 1 and f, having one process fail after sending a message to just one other process.

Consensus in Fail-Stop Setting

- Lower Bound: If there are f faults, indeed f+1 rounds are required in the worst-case scenario.
- Consider the situation where every round exactly one process fails.

One Table to Fill

Fault/ Comm. Model	Fault-Free	Faulty	
		Fail-Stop	Byzantine
Synchronous	Easy, All-to-All communication	f+1 rounds	
Asynchronous	Possible, to do later		

One Table to Fill

Fault/ Comm. Model	Fault-Free	Faulty	
		Fail-Stop	Byzantine
Synchronous	Easy, All-to-All communication	f+1 rounds	
Asynchronous	Possible, to do later		

The Byzantine Setting

- Let us now consider Byzantine faults and a synchronous setting.
- The name comes from the old Byzantine empire in history.
- Imagine that an army of invaders arranged in groups, each managed by a general, plans to attack from multiple directions.
- The attack is successful only if all the groups attack simultaneously.
- Requires the generals to reach an agreement on the time of attack.
- A general who is a traitor is modelled as a fault!

The Byzantine Setting

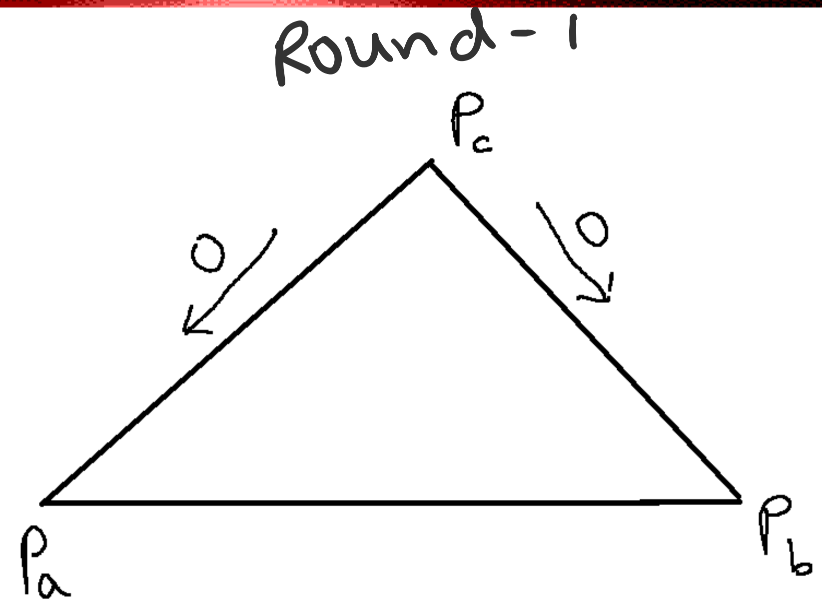
- The only way to communicate across the generals is by sending messages (via messengers)
- A faulty process, Byzantine general, may mislead one or more groups by sending different attacks times to different groups.
- Or, relay incorrect information.
- The problem of reaching an agreement in such settings is the agreement problem.

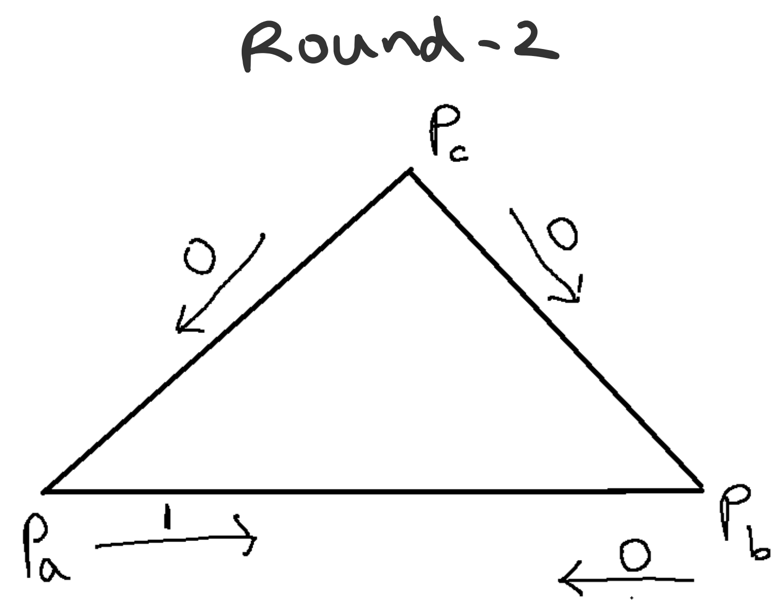
The Byzantine Setting

- Let us start a small example with 3 nodes and one of them being Byzantine faulty.
- Let us assume a synchronous fully-connected network topology.
- We use n to denote the number of nodes and f to denote the number of faulty nodes.
- In the example, $n = 3$ and $f = 1$.

The Byzantine Setting

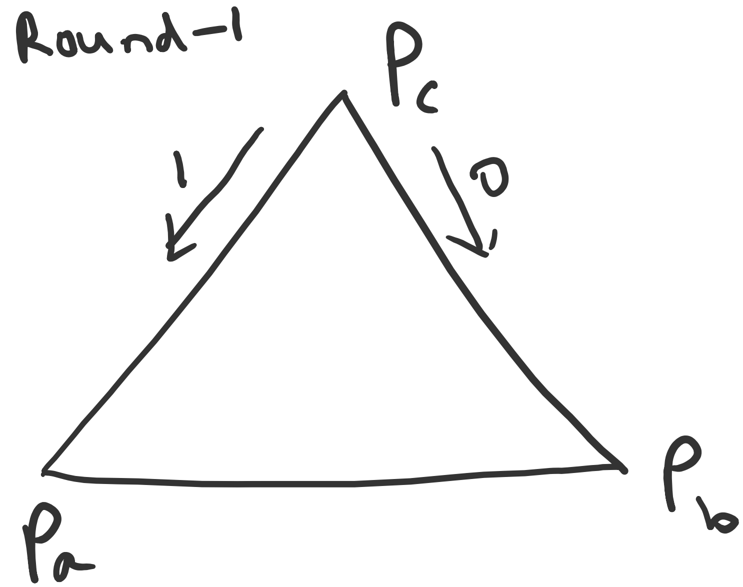
- Suppose that P_c is the initiator and P_a is the faulty node.
- The value at P_c is 0.
- In Round 1, P_c sends a value 0 to both P_a and P_b.
- In the Round 2, suppose that P_a sends 1 to P_b whereas P_b sends 0 to P_a.
- P_b has no clue as to which of P_a and P_c are at fault.
- Importantly, the same confusion exists if we assume that P_c and not P_a is faulty.

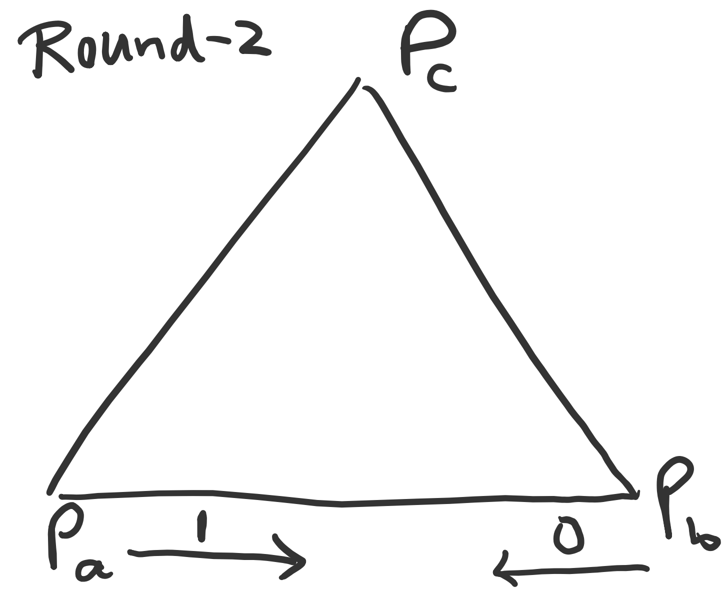




The Byzantine Setting

- Importantly, the same confusion exists if we assume that P_c and not P_a is faulty.
- Suppose that P_c is the initiator and P_c is the faulty node.
- The value at P_c is 0.
- In Round 1, P_c sends a value 1 to P_a and a value 0 to P_b.
- In the second round, P_a relays 1 to P_b whereas P_b sends 0 to P_a.
- P_b has no clue as to which of P_a and P_c are at fault.





The Byzantine Setting

- The trouble is that P_b gets identical inputs from P_c and P_a in two different scenarios with different output (validity) requirements.
- So, we conclude that one faulty node in a system of three nodes, agreement is not possible even in the synchronous setting.
- This result extends to also an n node system with f = n/3 or more.
- We establish a reduction as follows.

The Byzantine Setting

- Let S(n, f) be a synchronous system of n nodes with f of them being faulty.
  - Our earlier example is the S(3,1) system.
- Consider that f is at least n/3.
- Arrange the n nodes in S(n, f) into three subsets S_1, S_2, and S_3 each of size n/3.
  - Assume that n divides 3.
- We now map S(3,1) to these sets such that each node P_i in S(3,1) simulates the set S_i, for i = 1,2,3.