Distributed Systems

Monsoon 2024

Lecture 12

International Institute of Information Technology

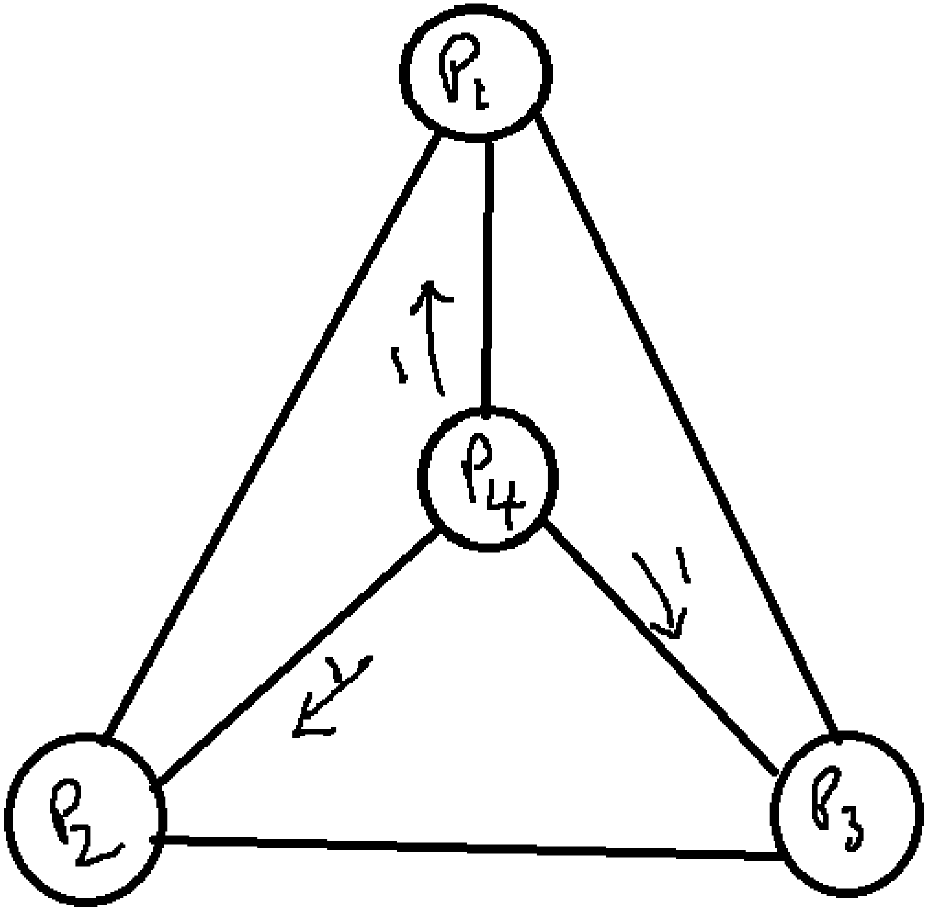
Hyderabad, India

An Algorithm

- Let us first see how agreement is possible in the S(4,1) system.
- Notice that S(4,1) has strictly less than n/3 faults.
- Consider that the initiator sends his value to all the other nodes.
  - A total of n-1 messages in this round
- In the second round, every other node sends the value it received from the initiator to all other nodes.
  - A total of (n – 1)(n – 2) messages.
- At the end of the second round, each process takes the majority of all the inputs it received.

An Algorithm

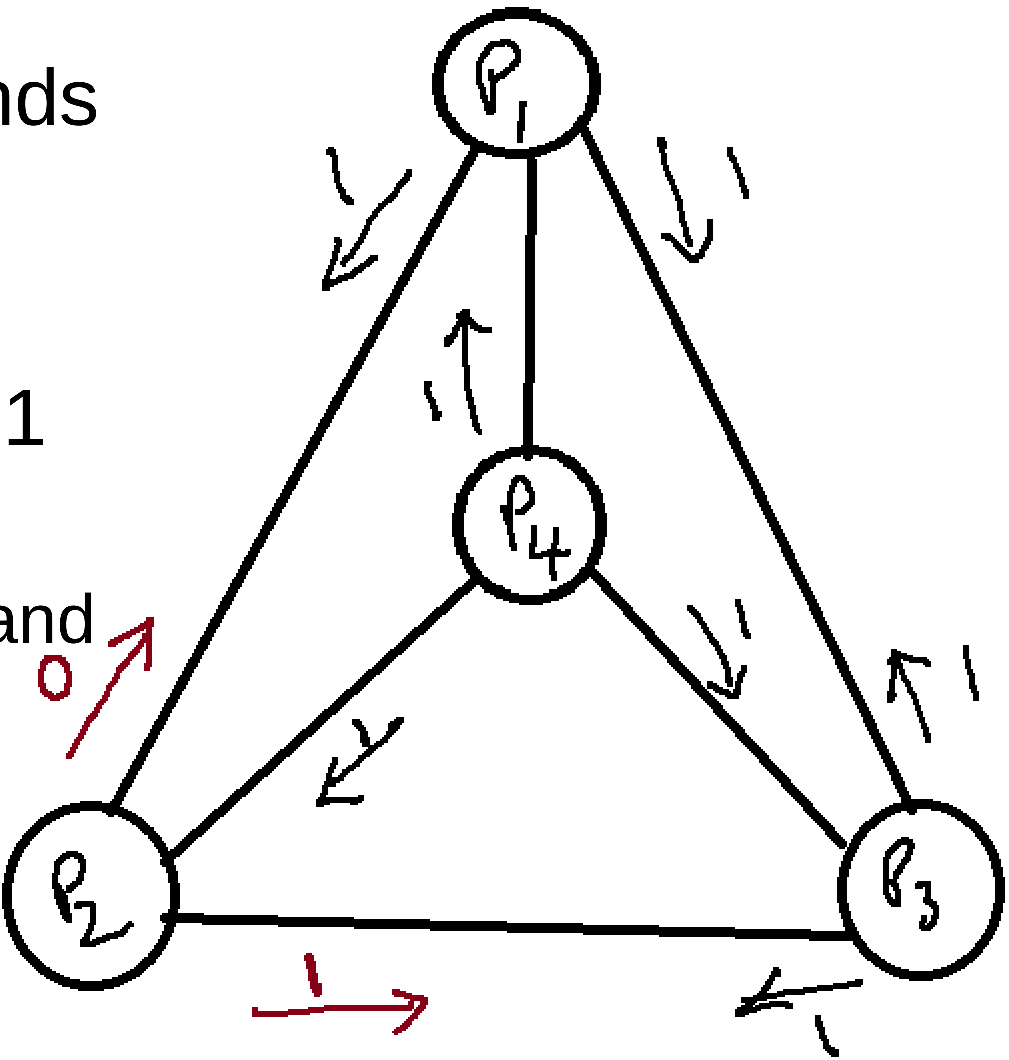
- S(4,1) in pictures.



S(4, 1)

- Round 2
- P2 is faulty, hence sends 1 to P3 and 0 to P1.

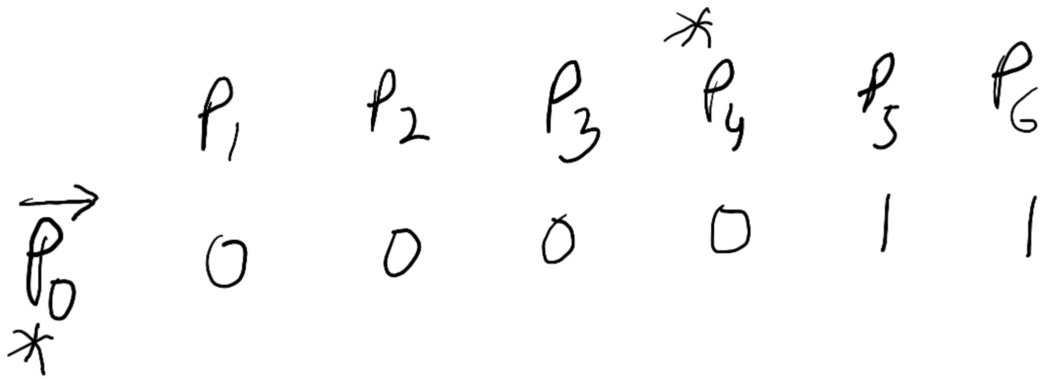
- Note the values that P1 receives.
  - 0 from P2, 1 from P3, and 1 from P4.



An Algorithm

- How is the algorithm working?
- Why did we need two rounds?
- The first round is clear – the initiator sends its value to everyone.
- Notice that there is at most one fault in the S(4,1) system.
- So, by having everyone relay the message that they hear from the initiator and then taking a majority, the effect of the faulty node is washed out.
- In other words, two rounds required to tolerate one fault.
- Can think of the actions of each node in the second round as a distinct subproblem of S(n-1, f-1).

An Algorithm



- Consider a seven node system with two faulty nodes.
- P_0 is the initiator and has initial value 0. Let us assume that P_0 is faulty.
- P_0 sends 0 to nodes P_1 through P_4 and 1 to others.

An Algorithm

$P_0 \downarrow$	P_2	P_3	P_4	P_5	P_6
$P_1 \rightarrow$	0	0	0	0	0

$P_0 \downarrow$	P_0	P_3	P_4	P_5	P_6
$P_2 \rightarrow$	0	0	0	0	0

$P_0 \downarrow$	P_1	P_2	P_4	P_5	P_6
$P_3 \rightarrow$	0	0	0	0	0

$P_0 \downarrow$	P_1	P_2	P_3	P_5	P_6
$P_4 \rightarrow$	0	0	1	0	0

*

$P_0 \downarrow$	P_1	P_2	P_3	P_4	P_6
$P_5 \rightarrow$	1	1	1	1	1

$P_0 \downarrow$	P_1	P_2	P_3	P_4	P_5
$P_6 \rightarrow$	1	1	1	1	1

- Consider the view of P_3, say. It receives 0 from P_0.
- In round 2, every one from P_1 to P_6 send the value they received from P_0 to everyone else.
  - But some could be faulty.
- Suppose that P_4 is faulty.
- In particular, P_4 sends 1 to P_3 and sends 0 to others.
- P_3 itself sends 5 messages with value 0 to everyone except P0 and itself.

An Algorithm

	P_0	P_1	P_2	P_4	P_5	P_6
@ P_3	0	0	0	1	1	1

	P_0	P_1	P_3	P_4	P_5	P_6
@ P_2	0	0	0	0	1	1

- At this stage, just taking majority might not help!
- Why?
- It is not clear who is at fault.
- P_3 cannot conclude that 0 is the "true value" and others sending 1 are faulty.
- Especially when the source is under fault.

- In particular, the output of different nodes at the nodes of this round can be different.
  - Not a valid consensus.

An Algorithm

- Let us see what the third round looks like.
- Each node (except P0) send (n-2)(n-3) messages conveying what they hear from each other in the previous round.

An Algorithm

$P_4 \downarrow$	P_2	P_3	P_5	P_6
$P_1 \rightarrow$	0	0	0	0

$P_4 \downarrow$	P_1	P_3	P_5	P_6
$P_2 \rightarrow$	0	0	0	0

$P_4 \downarrow$	P_1	P_2	P_5	P_6
$P_3 \rightarrow$	1	1	1	1

$P_4 \downarrow$	P_1	P_2	P_3	P_6
$P_5 \rightarrow$	0	0	0	0

$P_4 \downarrow$	P_1	P_2	P_3	P_5
$P_6 \rightarrow$	0	0	0	0

~~for~~ P_4	P_0	P_1	P_2	P_4	P_5	P_6
At P_3.	0	0	0	1	0	0

An Algorithm

- What if there are more nodes?
- What if there are more faulty nodes?
- How should the earlier algorithm be extended?
- Intuitively, need f+1 rounds to wash out the effect of all the (up to) f faulty nodes.
- In each round, each node acts as the initiator of a (distinct) $S(n_i, f_i)$ system with $n_i = n - i$ and $f_i = f - i$.

An Algorithm

- Intuitively, need f+1 rounds to wash out the effect of all the (up to) f+1 faulty nodes.
- In each round, each node acts as the initiator of a (distinct) $S(n_i, f_i)$ system with $n_i = n - i$ and $f_i = f - i$.
- Total number of messages across rounds i captured in the recurrence $M(n, f) = (n - 1)\, M(n-1, f-1)$, and $M(n, 1) = n-1$.
- Evaluates to $M(n, f) = (n - 1)(n - 2)(n - 3) \ldots (n - f - 1) = O(n^f)$.

An Algorithm

- In particular, if the initiator is non-faulty, then that node sends the same value to all others.
- So, all non-faulty ones will all have the same value.
- Even through multiple rounds, the non-faulty processors continue to have the same notion of majority.

An Algorithm

- The initiator could be malicious and sends conflicting values to the non-faulty processes.
- The remaining system has $f - 1$ malicious processes, but all the loyal processes do not have the same view to begin with.
- Need the extra rounds to allow the non-faulty ones to produce a valid output.

An Algorithm

- Can use the above observations to show that even if the status of the initiator is likely to be malicious, the algorithm can tolerate up to f faults and arrive at an agreement if $n > 3.f$

One Table to Fill

Fault/ Comm. Model	Fault-Free	Faulty	
		Fail-Stop	Byzantine
Synchronous	Easy, All-to-All communication	f+1 rounds	Possible if $n > 3f$ $O(n^f)$ messages.
Asynchronous	Possible, to do later		

Asynchronous Systems and Failures

- Let us consider a faulty asynchronous system and the consensus problem.
- The nature of fault really does not matter.
- Even under a single crash fault, it is known that in asynchronous systems, it is impossible to arrive at consensus.
- The result is due to Fischer, Lynch, and Paterson.
- Called the FLP result.

One Table to Fill

Fault/ Comm. Model	Fault-Free	Faulty	
		Fail-Stop	Byzantine
Synchronous	Easy, All-to-All communication	f+1 rounds	Possible if $n > 3f$ $O(n^f)$ messages.
Asynchronous	Possible, to do later	Impossible	Impossible

The FLP result

- The result is a bit devastating for practical systems.
- Most practical systems are asynchronous.
- The result says that many tasks including leader election are not possible in such systems.
- But, we know of several practical systems that do use consensus style mechanisms.
  - Various database protocols including UPI transactions
- So, there seems to be a middle–ground!

The Practical World – PAXOS

- A set of processes that can propose values
- Processes can crash and recover
- Processes have access to stable storage
- Asynchronous communication via messages
- Messages can be lost and duplicated, but not corrupted

The Practical World – PAXOS

- Two conditions to be satisfied:
- SAFETY: (also equivalent to Validity)
  - Only a value that has been proposed can be chosen
  - Only a single value is chosen
  - A process never learns that a value has been chosen unless it has been
- LIVENESS (also equivalent to Agreement+ Termination)
  - Some proposed value is eventually chosen
  - If a value is chosen, a process eventually learns it

PAXOS

- Paxos has three roles for nodes
  - Proposers
  - Acceptors, and
  - Learners

How to Choose a value?

- With a single acceptor.
- The acceptor looks at all proposals and chooses a value.
- Works fine until the acceptor fails.
  - No other process has knowledge of the choice
- More like a centralized solution. Not good enough for a distributed setting.

How to Choose a value?

- Use multiple acceptors.
- Choose only when a “large enough” or a “majority” set of acceptors accept
- Using a majority set guarantees that at most one value is chosen.

How to Accept a value?

- Use multiple acceptors.
- Suppose there is a single proposer.
- Suppose only one value is proposed by the single proposer.
- That value should be chosen!
- First Rule of Paxos – R1:
  - R1 : An acceptor must accept the first proposal that it receives.
  - This serves the situation when the acceptor just recovers from a fault also.
- But what if we have multiple proposers, each proposing a different value?

How to Accept a Value

- Acceptors must (be able to) accept more than one proposal
- To keep track of different proposals, assign a natural number to each proposal
  - A proposal is then a pair (psn, value)
  - Different proposals have different psn
  - A proposal is chosen when it has been accepted by a majority of acceptors
  - A value is chosen when a single proposal with that value has been chosen

Paxos

- We need to guarantee that all chosen proposals result in choosing the same value
- We introduce a second Rule, R2.
- R2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v.

which can be satisfied by:

- R2a. If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v

Paxos

- R2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v.

which can be satisfied by:

- R2a. If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v

Which can be tightened as

- R2b: If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v

Choosing a Value

- A proposer chooses a new n and sends to a majority of acceptors
- If an acceptor a receives <Prepare, n’, v’>, where n’ > n of any <Prepare, n, v> to which it has responded, then it responds to <Prepare, n’, v’>, with
  - a promise not to accept any more proposals numbered less than n’
  - the highest numbered proposal (if any) that it has accepted

The Acceptor's Protocol

- An acceptor receives prepare and accept requests from proposers. It can ignore these without affecting safety.
- It can always respond to a prepare request
- It can respond to an accept request with number n, accepting the proposal, iff it has not responded to a prepare request having number greater than n.
  - Lower numbered requests do not get a reply!

Towards Consensus

- If the proposer receives a response to <Prepare, n, v> from a majority of acceptors, then it sends <Accept, n, v> to each acceptor,
  - where v is either the value of the highest numbered proposal among the responses
  - any value if the responses reported no proposals
- If an acceptor receives <Accept, n, v> , it accepts the proposal unless it has in the meantime responded to <prepare, n'> , where n' > n

Liveness

- This scheme does not guarantee progress.
- Consider two proposes P1 and P2 who have to keep making proposals with increasing value, none of which are accepted.

- Key to progress is to have only proposer at a time.

Liveness

- Key to progress is to have only proposer at a time.
- To achieve that, in normal operation, elect a single server to be a leader.
- The leader acts as the distinguished proposer in all instances of the algorithm.

How about the FLP Result?

- What did Paxos do that FLP result of impossibility is bypassed?
- Paxos is always safe, but
- Paxos, for progress, needs a leader.
- But, leader election is difficult in the asynchronous setting.
- If there are enough periods of synchrony, then Paxos is both safe and live.