

# Paxos

## Distributed Consensus

Smruti R. Sarangi

Department of Computer Science  
Indian Institute of Technology  
New Delhi, India

# Outline

## 1 Problem of Consensus

## 2 Paxos

- Motivation
- Algorithm
- Analysis

# Consensus

- Each process might propose a value.
- Processes co-ordinate to agree upon one value,  $V$ .
- $V$  needs to be proposed by at least one process.
- All processes need to agree upon  $V$ .

## Uses

- 1 Solve the distributed commit problem
  - 1 Each process can propose two values – **commit** or **abort**
  - 2 All the processes agree to either commit or abort.
- 2 Run multiple copies of a computation, and use voting to decide the result.

# Consensus with Reliable Processes

## Simple Algorithm

- Elect a leader.
- Use the leader to collect proposals, decide on a consensus value, and broadcast the all processes.
- Issues:
  - Fault tolerance
  - Centralized processing

# Consensus with Reliable Processes

## Simple Algorithm

- Elect a leader.
- Use the leader to collect proposals, decide on a consensus value, and broadcast the all processes.
- Issues:
  - Fault tolerance
  - Centralized processing

The consensus problem is interesting in the presence of faults

# Basic Results

## Result by Fischer, Lynch, Patterson

It is impossible to achieve consensus with even one faulty process in an asynchronous system.

Reasons:

- We cannot distinguish between a failed and a slow process.
- Assume  $2n + 1$  processes.
  - Let  $n$  processes propose 1, and let  $n$  processes propose 0.
  - The faulty/slow process holds the key.
  - The algorithm can thus get stuck forever.

# Way Ahead

- **Safety Property** : Something wrong does not happen.
  - If the traffic light on one road is green, then the traffic light on the perpendicular road is red.
- **Liveness Property** : Something good always happens.
  - The red light will ultimately turn green.
- Let us propose protocols that never violate the safety constraint.
- Liveness is a secondary criteria.

# Types of Agents

- **Proposer** Proposes a value
- **Acceptor** A set of nodes that accept proposed values
- **Learner** Nodes that join the consensus protocol and learn the accepted value.
- Note that a node can be a proposer, and acceptor at the same time.



# Outline

## 1 Problem of Consensus

## 2 Paxos

- Motivation
- Algorithm
- Analysis

# Conditions

## First-Accept Condition: C1

An **acceptor** does not know how many proposals are there in the system. Hence, he must accept the first proposal that he gets.

- Several values could be proposed by different proposers.
- Different acceptors could accept different proposals.
- Hence, an **acceptor** needs to accept multiple proposals.
- Each **proposer** however needs to choose the consensus proposal.

# Proposal Numbering

- 1 Proposal  $\rightarrow$  (number, value)
- 2 All the proposal numbers are unique

## Condition – C2(Consensus Condition)

If a proposal  $(n, v)$  is chosen, then every proposal with a number greater than  $n$  that is chosen, has value  $v$ .

- By induction, we can prove that only one value is chosen.

Let us now see, how to satisfy condition C2 to achieve consensus.

# Condition – C2a

## Condition – C2a

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

- Assume that a process wakes up, and gets a proposal. By condition C1, it needs to accept it.
- This is not desirable.
- We need to strengthen C2a.

# Condition – C2b

## Condition – C2b

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

- C2b is a fairly strong condition.
- It ensures that after a proposal is chosen, all higher numbered proposals propose the chosen value.
- Any acceptor, which joins the protocol late, is proposed the consensus value, which it can readily accept.
- The main challenge is to ensure condition – C2b.

# Outline

## 1 Problem of Consensus

## 2 Paxos

- Motivation
- **Algorithm**
- Analysis

# Paxos Algorithm

## Algorithm 1: Paxos: Phase 1

- 1 **Propose:** *Proposer* sends  $\text{prepare}(n)$  to a majority of *acceptors* .
- 2 **Receive  $\text{prepare}(n)$ :**
- 3     **if**  $n > \text{maxPrep}$  **then**
  - 3     |      $\text{maxPrep} \leftarrow n$
  - 3     |     **return**  $\text{maxAccept}$
- 4 **end**

# Paxos Algorithm - II

## Algorithm 2: Paxos: Phase 2

1 Receive ( $maxAccept_i$ ) from a majority of acceptors:

$v \leftarrow \max(maxAccept_i)$

send accept( $n, v$ ) to all the **acceptors** in the quorum

2 Received accept( $n, v$ ) request:

if  $n \geq maxPrep$  then

3 |  $maxAccept \leftarrow v$

| accept the proposal ( $n, v$ )

| send response to **proposer**

4 end

5 Received all responses to accept messages:

Choose the value ( $v$ )



# Outline

## 1 Problem of Consensus

## 2 Paxos

- Motivation
- Algorithm
- **Analysis**

# Analysis of Paxos

## Definitions

Let us consider two proposals,  $P_1$ , and  $P_2$ .

- If an acceptor( $A$ ) receives  $P_2$ 's prepare message after  $P_1$ 's accept message, then  $P_1 < P_2$  at  $A$ .
- If an acceptor( $A$ ) receives  $P_2$ 's prepare message after  $P_1$ 's prepare message is ignored, then  $P_1 < P_2$  at  $A$ .
- $P_1$  is concurrent with  $P_2$  ( $P_1 \bowtie P_2$ ) at  $A$  if, if  $P_1 \not< P_2$ , and  $P_2 \not< P_1$ .

## Definitions

- $P_1 \bowtie P_2$ , if  $P_1 \bowtie P_2$  at any acceptor.
- Otherwise, either  $P_1 < P_2$ , or  $P_2 < P_1$ .

# Analysis - II

## Theorem

*If  $P_1 \bowtie P_2$ , and  $P_1.num < P_2.num$ ,  $P_1$  will not pass phase II at the common acceptor.*

## Proof.

- There must be some acceptor that gets messages for both proposals.
- Assume it first gets a prepare message from  $P_1$ .
  - $P_1$  will not succeed in phase II.
- Assume it first gets a prepare message from  $P_2$ .
  - $P_1$  will not succeed in phase I.



# Analysis - III

## Theorem

If  $P_1 \prec P_2$ , and  $P_1.num > P_2.num$ ,  $P_2$  will not pass phase I.

## Proof.

- There must be some acceptor that gets messages for both proposals.
- $P_2.num < maxPrep$  at that acceptor.
- Hence,  $P_2$  will not pass phase I.



# Analysis - IV

## Theorem

*If  $P_1 \prec P_2$ , and both of them succeed (pass phase II at all acceptors), then they choose the same value.*

## Proof.

- There must be some acceptor ( $A$ ) that gets messages for both proposals.
- For  $P_2$  to succeed  $P_2.num > P_1.num$ .
- Induction hypothesis: All the successful transactions with numbers between  $P_1.num$  and  $P_2.num$ , choose the value chosen by  $P_1$ .
- $A$  must have forwarded its value  $v$  at the end of phase 1 to the proposer of  $P_2$ .
- If  $v$  is the maximum value of the response, we are done.



## Analysis - V

## Proof.

- If it is not the case, then assume  $v' > v$  is the maximum value.
- Let  $P_3$  propose  $v'$ . Now, by the induction hypothesis  $P_3.num < P_1.num$  or  $P_3.num > P_2.num$ .
- If  $P_3.num < P_1.num$ , then  $P_1$  must have seen  $P_3$ 's value (cannot be concurrent). **Not Possible**
- If  $P_3.num > P_2.num$ , then  $P_2$  will not pass phase II.



# Progress

- Note that we have not said anything about progress.
- If  $P_1 \bowtie P_2$ , one of them will not succeed.
- We can possibly have a chain of failures, and theoretically never achieve consensus!
- **Solution** Eliminate concurrent proposals.
  - Use a distinguished proposer (leader), who can propose.
  - If the leader fails, choose a new leader.
  - Paxos provides safety, just in case the old leader wakes up.

-  The part-time parliament, Leslie Lamport. ACM TOCS, 1998.
-  Paxos made Simple, Leslie Lamport.