Distributed Systems

EECE 513: Design of Fault-tolerant Digital Systems

Learning Objectives

- List the issues in design of distributed systems
- Define the three agreement protocols and delineate their relationship with each other
- Understand the Byzantine agreement algorithm using oral messages and signed messages
- Define the Paxos problem and understand how to solve it – identify the common use-cases for Paxos

Agreement Protocols

- **It is often required that processes reach a mutual agreement.**
- **Faulty processes can send conflicting values to other processors preventing them from reaching an agreement**
- **Processes must exchange their values and relay the values received from other processes several times to isolate the effects of faulty processes.**

- **System model**
  - There are *n* processes in the system and at most *m* of them can be faulty.
  - Processes communicate with one another by message passing and the receiver process always knows the identity of the sender process
  - The communication network is reliable, i.e., only processes can fail

Synchronous vs. Asynchronous

- In ***synchronous computation***, processes in the system run in lockstep:
  - In each step, a process receives messages (sent to it in the previous step), performs computation, and sends messages to other processes (received in the next step).
  - A process knows all the messages it expects to receive in a step/round
- In ***asynchronous computation***, processes do not execute in lockstep:
  - A process can send and receive messages and perform computation at any time
  - Agreement is impossible with even a single, faulty process – FLP result by Lynch et al.

Model of Processor Failures

- **Crash fault**: Processor stops functioning and never resumes operation
- **Omission fault**: Processor "omits" to send messages to some processors
- **Malicious fault**: Processor behaves randomly and arbitrarily (Byzantine fault)
- In synchronous model, omission can be detected. We assume synchronous model.

Authenticated vs. Non-Authenticated Messages

- To reach an agreement, processes need to exchange their values and relay the received values to other processors

Two Types of Messages:

- ***Authenticated (signed)***
  - A faulty process cannot forge a message or change the contents of a received message (before it relays the message to other processes).
  - A process can verify the authenticity of the received message.
- ***Non-authenticated (oral)***
  - A faulty process can forge a message and claim to have received it from another processor or change the contents of the received message before it relays it to other processes.
  - A process has no way to verify the authenticity of the received message.

Learning Objectives

- List the issues in design of distributed systems
- Define the three agreement protocols and delineate their relationship with each other
- Understand the Byzantine agreement algorithm using oral messages and signed messages
- Define the Paxos problem and understand how to solve it – identify the common use-cases for Paxos

Agreement Problems - Classification

- ***The Byzantine Agreement Problem***
  - **A single value** is **initialized by any arbitrary process,** and all non-faulty processes have to agree on that value

- ***The Consensus Problem***
  - Every process has its **own initial value,** and all correct processes must agree on **a single, common value.**

- ***The Interactive Consistency Problem***
  - Every process has its **own initial value,** and all non-faulty process must agree on **a set of common values.**

The Byzantine Agreement Problem

- An **arbitrarily chosen process** - *the source* - broadcasts its value to all other processes.

- ***Agreement*** - All non-faulty processes agree on the same value

- ***Validity*** - If the source process is non-faulty then the common value agreed on by all non-faulty processes should be the value of the source

The Consensus Problem

- **Every process** broadcasts its initial value to all other processes
  - *Initial values of the processes may be different.*

- ***Agreement*** - All non-faulty processes agree on the same single value.

- ***Validity*** - if the initial value of every non-faulty process is υ, then the common value agreed upon by non-faulty processes must be υ.

The Interactive Consistency Problem

- **Every process** broadcasts its initial value to all other processes
  - *Initial values of the processes may be different.*

- ***Agreement*** - All non-faulty processes agree on the same vector:
  $(\upsilon_1, \upsilon_2, \ldots, \upsilon_n)$

- ***Validity*** - If the i_{th} process is non-faulty and its initial value is υ_i, then the i_{th} value to be agreed on by all non-faulty processes must be υ_i

Relations Among the Agreement Problems

1. Given an algorithm to solve Byzantine agreement, how would you solve Interactive Consistency?
2. Given an algorithm to solve Interactive Consistency, how would you solve Consensus?
3. Given an algorithm to solve Consensus, how would you solve Byzantine Agreement?

Learning Objectives

- List the issues in design of distributed systems
- Define the three agreement protocols and delineate their relationship with each other
- Understand the Byzantine agreement algorithm using oral messages and signed messages
- Define the Paxos problem and understand how to solve it – identify the common use-cases for Paxos

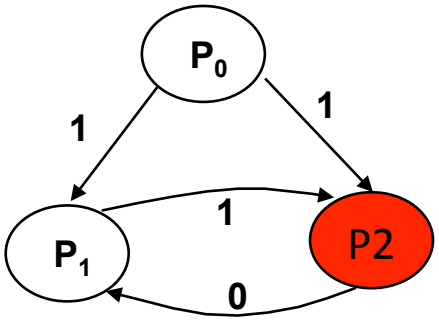
Byzantine Agreement Problem

- In a fully connected network it is impossible to reach a consensus if the number of faulty processes, *m* , exceeds $\lfloor (n-1)/3 \rfloor$,
  - For example, if $n = 3$, than $m = 0$, i.e., having three processes, we cannot solve the Byzantine agreement problem n the event of a single error.
  - The protocol requires *m+1 rounds of message exchange* (*m* is the maximum number of faulty processes)
  - This is also the lower bound on the number of rounds of message exchanged.
- Using authenticated messages, this bound is relaxed, and a consensus can be reached for any number of faulty processes.
  - We assume non-authenticated messages (oral messages)

Impossibility Results

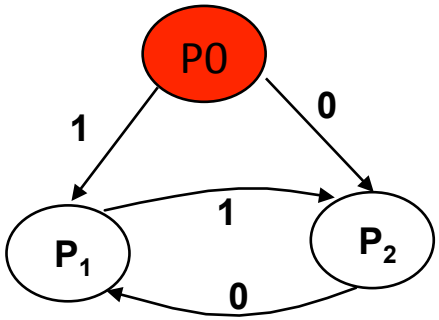
- Consider a system with three processes p_1, p_2, p_3
- There are two values, 0 and 1, on which processes agree.
- p_0 initiates the algorithm.

Case one - p_0 is not faulty



assume p_2 is faulty
suppose p_0 broadcast 1 to p_1 and p_2
p_2 acts maliciously and sends 0 to p_1
p_1 must agree on 1 if algorithm is to be satisfied
p_1 receives two conflicting values
no agreement is possible

Case one - p_0 is faulty



suppose p_0 sends 1 to p_1 and 0 to p_2
p_2 communicates 0 to p_1
p_1 receives two conflicting values
no agreement is possible

Oral Messages Algorithm OM(m)

- A recursive algorithm solves the Byzantine agreement problem for >= **3m+1** processes in the presence of at most ***m*** faulty processes.

Algorithm OM(0)

- 1. The source process sends its value to every process,
- 2. Each process uses the value it receives from the source (if it receives no value, then it uses a default value of 0).

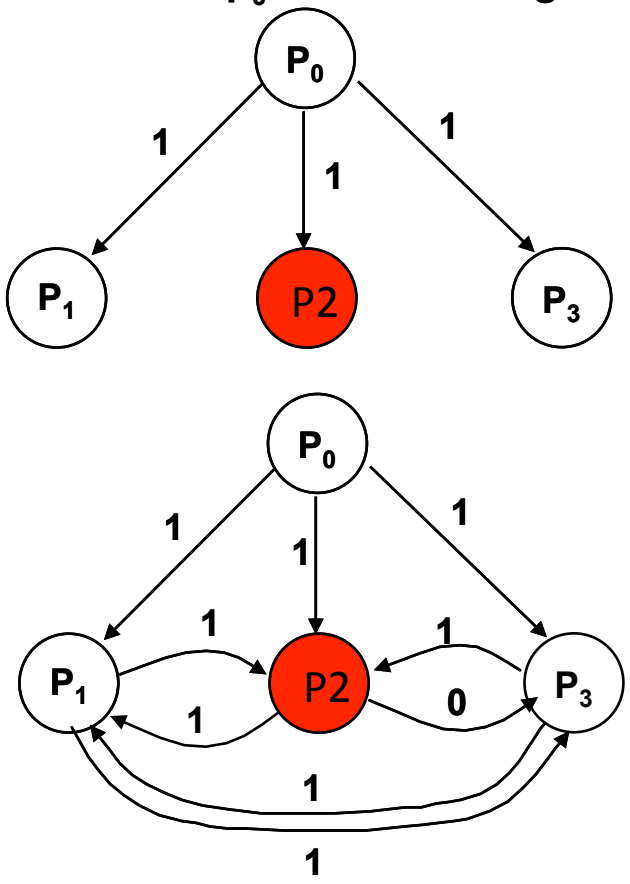
Oral Messages Algorithm OM(m)

Algorithm OM(m), m > 0

1. The source process sends its value to every process,
2. For each *i*, let υ_i be the value processor *i* receives from the source,
   - Process *i* acts as a new source and initiates ***Algorithm OM(m-1)*** wherein it sends the value υ_i to each of the n-2 other processes
3. For each *i* and each $j \neq i$ let υ_j be the value process *i* received from *j* in step (2) using *Algorithm OM(m-1).* (If no value is received then default value 0 is used). Process *i* uses the value *majority* $(\upsilon_1, \upsilon_2, \ldots, \upsilon_{n-1})$.

Oral Messages Algorithm OM(m)

Consider a system with four processes p_0, p_1, p_2, p_3
p_0 initiate the algorithm; p_2 is faulty



To initiate the agreement p_0 executes OM(1) wherein it sends 1 to all processes

At step 2 of the OM(1) algorithm, p_1, p_2, p_3 execute the algorithm OM(0)

p_1 and p_3 are non-faulty and
p_1 sends 1 to $\{p_2, p_3\}$
p_3 sends 1 to $\{p_1, p_2\}$
p_2 is faulty and sends 1 to p_1 and 0 to p_3

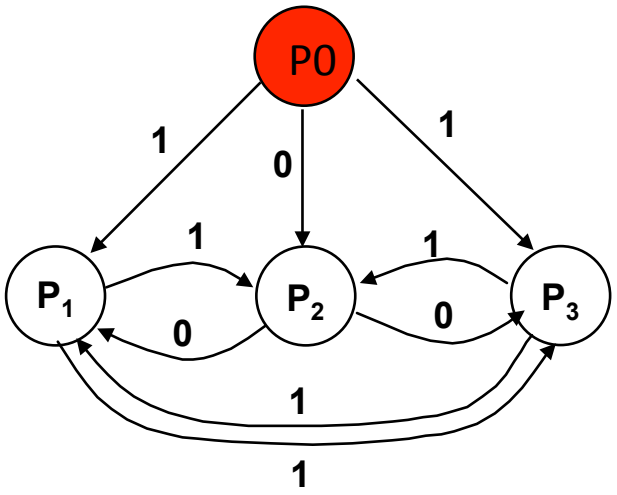
After receiving all messages p_1, p_2, p_3 execute step 3 of the OM(1) to decide the majority value

p_1 received $\{1, 1, 1\} \Rightarrow 1$
p_2 received $\{1, 1, 1\} \Rightarrow 1$
p_3 received $\{1, 1, 0\} \Rightarrow 1$

Both conditions of the Byzantine agreement are satisfied

Oral Messages Algorithm OM(m) (cont.)

Consider a system with four processes p_0, p_1, p_2, p_3
p_0 initiate the algorithm; p_0 is faulty



P_0 send conflicting values to p_1, p_2, p_3

Under step 1 of OM(0) p_1, p_2, p_3 send the received values to the other two processes

p_1, p_2, p_3 execute step 3 of OM(1) to decide on the majority value

p_1 received {1, 0, 1} ⇒ 1
p_2 received {0, 1, 1} ⇒ 1
p_3 received {1, 1, 0} ⇒ 1

Both conditions of the Byzantine agreement are satisfied

Protocol with Signed Messages

- Transmitter sends a “signed” message (use digital signature from asymmetric cryptography)
- If a node changes the content of message from transmitter before forwarding it, the receiver can detect the forgery
- With signed messages, agreement can be reached between $n=m+2$ processes, where m is the number of faulty processes
- Each process maintains a set V_i (for process i) that has all the unique values that it has received

Protocol with Signed Messages

Algorithm SM(m)

1. The transmitter (process 0) signs its value and sends to other nodes
2. For each process *i*:
   A. If process *i* received message *v*: 0 (i) it sets V_i to {*v*}; (ii) it sends *v*: 0: *i* to every other process
   B. If process *i* received message *v*: 0: j_1: … : j_k and $v \notin V_i$, then (i) it adds *v* to V_i; (ii) if $k < m$, it sends *v*: 0: j_1: …: j_k to every process other than j_1, …, j_k
3. For each process *i*, when it receives no more message, it considers the final value as *choice*(V_i)

Outline

- Issues in design of distributed systems
- Agreement protocols
- Byzantine agreement algorithm
- Paxos

Paxos: Problem

- Most failures are not Byzantine in the real world

- Consensus under non-Byzantine faults
  - Can be achieved with 2f + 1 processes
  - Assumes crash-stop-recovery failure semantics
  - Assumes network can lose or reorder messages
  - Requires at most 4f + 4 messages in fault-free case

Paxos: Principals

- **Proposer: Node that initiates the protocol**
  - Proposes an initial value to agree upon
  - May be more than one proposer to start with
  - **Leader:** A distinguished, trusted proposer

- **Acceptor: All other nodes that participate**
  - Can reject the proposal from the proposer
  - Can agree with the proposal, but is prevented from agreeing to proposals from other proposers

Paxos: Properties

- **Non-triviality**
  - The value learned is one of the proposed ones
- **Safety:**
  - At most of the proposed values is learned
- **Liveness:**
  - Eventually, all non-faulty acceptors will learn it

Paxos: Proposal Numbers

- Because multiple proposers can be active, we need a way to distinguish proposals
  - Assume that there is a global mechanism to sequence proposals from 1 .. N

- An acceptor accepts a proposal with value m if and only if it has not responded to a proposal with value higher than m (with a promise)

Paxos Algorithm: Phase 1

- **Proposer** selects a proposal number n and sends a prepare request with number n to a majority of acceptors (quorum)
- If an **acceptor** receives a prepare request with number ‘n’ greater than that of any prepare request to which it has already responded, then it responds with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal that it has accepted.

Paxos Algorithm: Phase 2

- If the proposer receives a response to its prepare requests numbered n from a majority of acceptors, then it sends an accept request to each of those acceptors numbered n with a value v, where v is the value of the highest-numbered proposal among the responses, or is any value.
- If an acceptor receives an accept request for a proposal numbered n, it accepts the proposal unless it has already responded to a prepare request having a number greater than n.

Termination

- How do we tell if a value has been learned by a majority of the processes ?
  - Solution: Have specially designated processes called Learners, which keep track of the accepted values. Acceptors send their responses to Learners, who may then communicate with other learners to spread the information
  - Message loss may prevent learners from ever finding out the value accepted by a quorum

Paxos: Failure modes

- More than one proposer
- Failures of proposer, acceptor (non-majority) or learners
- Network failures among any pairs of links
- Failure of leader (rare event)

Paxos: Message Complexity

- Failure-free operation:
  - Phase 1: Proposer sends f + 1 messages
  - Acceptors provide f + 1 responses
  - Phase 2: Proposer sends f + 1 messages and receives another f + 1 responses

- Total number of messages = 4f + 4

Compared to OM(n) protocol, this is much lower

Outline

- Issues in design of distributed systems
- Agreement protocols
- Byzantine agreement algorithm
- Paxos