1: Introduction and formal model(s)

Slides based on book ‘Distributed Computing:..’ By David Peleg
Some slides based on slides by Peter Robinson (RHUL)- Advanced Distributed Systems module.
1: Introduction and formal model(s)

Slides based on book ‘Distributed Computing:..’ By David Peleg
Some slides based on slides by Peter Robinson (RHUL)- Advanced Distributed Systems module.
Example: Some of my Algorithms

Centralised

Distributed: Static

Distributed: Dynamic/Fault-Tolerant

Leader Election

Byzantine Agreement

Self-Healing Compact Routing/ Internet of Things

Self-Healing Topology Maintenance

Self-Healing SDN

Self-Aware Analytics

Game Theory: EU Games

Biological Distributed Algorithms

Graph Theory

Analysis

Probability

Logic

Programming

Networking

concurrent thinking!
Let’s begin....

Welcome to the *Zoo* of distributed computing

....
Let’s begin….  

Welcome to the **Zoo** of distributed computing  

….  

Here you’ll find many kind of creatures - regular or strange, benign or dangerous, dumb or rational and cunning, healthy or weak, quick or slow, boring or interesting … :)
Let’s begin....

Welcome to the Zoo of distributed computing ....

Here you’ll find many kind of creatures - regular or strange, benign or dangerous, dumb or rational and cunning, healthy or weak, quick or slow, boring or interesting … :)

But let’s begin with some simple ideas...

...
Centralised: Who gets to print?

Centralised Algorithms:
Single computer with the whole problem instance/data available.

Q: which one of them will get the printer?
Distributed: Who gets to print?

DeCentralised/Distributed Algorithm: Multiple `computers' each with its own local view/data.

Q: which one of us will get the printer?
Saket’s class problem!
Problem: Inform everyone about class = \textit{Broadcast} M

Broadcast: Send a message from a source to every node in the network!
Saket’s class problem!
Saket’s class problem!

We have a ‘rough’ notion of broadcast in real life.

What are the questions we need to ask to mathematically formalise and analyse this process!

Questions?

Solution: Broadcast M
The Internet has (almost instantaneously) taken over global communications: from 1% in the year 1993, to 51% by 2000, and more than 97% in 2007

[Hilbert et al, The worlds technological capacity to store, communicate, and compute information. Science, 2011]
Message Passing model

- Suitable for Networks!
- Model the network as a Graph
- Each node/vertex is a processor and edges are links
- Communication is only by sending messages along edges (Point-to-point communication)
- Only 'local knowledge' initially
- Local computation 'free' (usually)
- Metrics?
Message Passing model

- Suitable for Networks!
- Model the network as a Graph
- Each node/vertex is a processor and edges are links
- Communication is only by sending messages along edges (Point-to-point communication)
- Only 'local knowledge' initially
- Local computation 'free' (usually)
- Metrics? - Time?
Message Passing model

- Suitable for Networks!
- Model the network as a Graph
- Each node/vertex is a processor and edges are links
- Communication is only by sending messages along edges (Point-to-point communication)
- Only `local knowledge' initially
- Local computation `free' (usually)
- Metrics?
  - Time:
  - Parallel time (count ‘time’ with different parts of the system functioning concurrently)
Saket’s class problem!

Broadcast $M$ on Graph $G$

‘Obvious’ Algorithm: Flooding
- Send message to every node!

*https://101clipart.com/ticking-clock-clipart/*
Saket’s class problem!

‘Obvious’ Algorithm: Flooding

Flooding: - At every node, On receiving M, send M on all other edges

Q: How much ‘time’ does flooding take?
Flooding M on G

Q: How much ‘time’ does this take?

Ans: 1 time unit

Analysis:
Message from originator reaches all others in parallel
Saket’s class problem!: Solution

Flooding M on G

Q: How much ‘time’ does this take?

Ans: 1 time unit

Analysis:
Message from originator reaches all others in parallel

Incorrect!*
Where’s the problem?

*Usually! Maybe correct in some models (later).
(Formally) What is a distributed algorithm and a protocol?

- Given a distributed system with n nodes:
  - **Protocol:** A local program executed by a vertex in the network. The protocol governs how the processor modifies local data and exchanges messages.
  - **Distributed Algorithm:** A distributed algorithm is the collection of n protocols, each dedicated to one of the vertices.
Saket’s class problem!: Problem with ‘flooding’ analysis

Flooding $M$ on $G$

Q: How much ‘time’ does this take?

Revisit the Algorithm:

Flooding: - **At every node:** On receiving $M$, send $M$ on all other edges

Intuition: Every node is:

- An independent processor (think as individual state machines/TMs)
- Has a *local view*: Only it’s own state and incoming/outgoing messages
- Q: When should the algorithm stop (at every machine!)?
Saket’s class problem!: Problem with ‘flooding’ analysis

**Flooding M on G**

Q: How much ‘time’ does this take?

Flooding: - **At every node**: On receiving M, send M on all other edges

- Q: When should the algorithm stop (at every machine!)?
Saket’s class problem!: Problem with ‘flooding’ analysis

Flooding M on G

Q: How much ‘time’ does this take?

Flooding: - **At every node**: On receiving M, send M on all other edges

• Q: When should the algorithm stop (at every machine!)?

**TERMINATION!**

The formal property showing end of a distributed algorithm!
Saket’s class problem!: Problem with ‘flooding’ analysis

Flooding $M$ on $G$

Q: How much ‘time’ does this take?

Flooding: - At every node: On receiving $M$, send $M$ on all other edges

• Q: When should the algorithm stop (at every machine!)?

TERMINATION!
Saket’s class problem!: Problem with ‘flooding’ analysis

Flooding $M$ on $G$  

Q: How much ‘time’ does this take?

Flooding: - At every node: On receiving $M$, send $M$ on all other edges

Termination: Consider the algorithm from Mr. Tom’s point of view.

Locally: Mr. Tom only knows it got a message $M$ in Round 1 (may not know topology etc) from one edge, so it faithfully sends the message on its other edge in Round 2! (Locally terminated)

Globally: Broadcast already achieved in Round 1 (in a centralised view)

However, each node only terminates in Round 2 (Global distributed termination)

Is $T = O(n)$?
# Centralised vs Distributed

<table>
<thead>
<tr>
<th>Centralised</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>All inputs centrally available</td>
<td>Inputs spread across the network</td>
</tr>
<tr>
<td>Single global view/state</td>
<td>Local views at processing units. Global state sum of local states</td>
</tr>
<tr>
<td>Simpler algorithmic model</td>
<td>More complex algorithmic model. <em>Hidden adversary</em> controlling network actions.</td>
</tr>
</tbody>
</table>
Flooding M on G
Q: How much ‘time’ does this take?

Ans: 1 time unit

Q. Given what assumptions, could this problem be solved in 1 round?
Saket’s class problem!: Solution revisited

Flooding M on G

Q: How much ‘time’ does this take?

Ans: 1 time unit

Q. Given what assumptions, could this problem be solved in 1 round?

• All nodes (at least both r1 receivers) know:
  • The complete topology
  • Know that messages are always delivered reliably
The Elephants in the Room!

Q: How much ‘time’ does this take?

Broadcast M
The Cheetahs in the Room!

Broadcast M

Q: How much ‘time’ does this take?
Elephants and Cheetahs and ? in the Room!

Broadcast M

Q: How much ‘time’ does this take?
The Elephants in the Room!

Broadcast M  Q: How much ‘time’ does this take?

- If only Elephants (or Cheetahs) and we know it, we use the **synchronous** model (formal definition to follow)
- If there is uncertainty of the speed of the links, we have to use the **asynchronous** model (formal definition to follow)
## Dimensions and Parameters!

<table>
<thead>
<tr>
<th>Architecture And Coupling</th>
<th>Message Passing (loose coupling)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timing</strong></td>
<td>Synchronous</td>
<td>Asynchronous</td>
</tr>
<tr>
<td><strong>Knowledge</strong></td>
<td>Global</td>
<td>Strictly local</td>
</tr>
</tbody>
</table>

- **Architecture And Coupling**
  - Message Passing (loose coupling)

- **Timing**
  - Synchronous
  - Asynchronous

- **Knowledge**
  - Global
  - Strictly local
Bergen class problem!

I won the Nobel Prize. Let’s pray!

Who you?

Who you?
Identifiers help!

- (Unique) Node Identifiers
- Reasonable assumption: node knows its ID and that of its immediate neighbours
## Dimensions and Parameters!

<table>
<thead>
<tr>
<th>Architecture And Coupling</th>
<th>Message Passing (loose coupling)</th>
<th>Timing</th>
<th>Identity</th>
<th>Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Synchronous</td>
<td>Anonymous</td>
<td>Local (Own + neighbour IDs), + size, did estimates?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ID</td>
<td>Global</td>
</tr>
</tbody>
</table>
# Red vs. Blue

## Shared Memory
- For tightly coupled architecture, which share memory
- Shared variables, read and written concurrently - implicit communication

## Message Passing
- Loosely coupled - networks
- Explicit communication - communication as computational resource
Models: the big picture

<table>
<thead>
<tr>
<th></th>
<th>Message Passing</th>
<th>Shared Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

- **Synchronous Shared Memory** is the **PRAM model** (popular with HPC and parallel computing community)
Intermediate models and architectures!

- Wireless networks
- Sensor networks
- Broadcast networks
- Overlay networks

(Can be modelled as variations of message passing model)
## Dimensions and Parameters!

<table>
<thead>
<tr>
<th>Architecture And Coupling</th>
<th>Message Passing (loose coupling)</th>
<th>Sensor, Wireless, Overlay …</th>
<th>Shared Memory (tight coupling)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timing</strong></td>
<td>Synchronous</td>
<td></td>
<td>Asynchronous</td>
</tr>
<tr>
<td><strong>Identity</strong></td>
<td>Anonymous</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td><strong>Knowledge</strong></td>
<td>Local (Own + neighbour IDs), + size, did estimates?</td>
<td></td>
<td>Global</td>
</tr>
</tbody>
</table>
Fully Synchronous model

- Bounded link delays
- Lockstep synchronisation: as if one global clock
- Message synchronisation property:
  - Message sent from processor $v$ to neighbour $u$ at pulse $p$ of $v$ must arrive at $u$ before pulse $p+1$ of $u$.

Computation Round Steps:
1. Send messages to (some of) the neighbours.
2. Receive messages from (some of) the neighbours.
3. Perform some local computation

Synchronous protocol: (Round based) Describes local computation and messages for every round.
Asynchronous model

- Messages arrive in finite but unpredictable time
- Clocks are practically useless!
- Delay vs Transmission:
  - Cannot differentiate between message delayed or not sent at all!

Computation Round Steps:
1. Send messages to (some of) the neighbours.
2. Receive messages from (some of) the neighbours.
3. Perform some local computation

Synchronous protocol: (Round based) Describes local computation and messages for every round.
Asynchrony leads to Nondeterminism!!

- Messages arrive in finite but unpredictable time
- Asynchronous model is inherently *nondeterministic*:
  1. Even when protocols are strictly deterministic and use no randomization!
  2. Arbitrary message delivery leads to non-determinism - modelled by *adversarial scheduling demons*

<table>
<thead>
<tr>
<th>P₁, P₂ protocol:</th>
<th>P₁</th>
<th>P₃ protocol:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send x to P₃</td>
<td>P₂</td>
<td>On getting message, print message &amp; halt</td>
</tr>
</tbody>
</table>

**Asynchronous protocol**: Has to be described as *event driven*  
"*On event X, do Y*"

- Processor idles waiting for an event X, executes Y and resumes wait
- Time complexity is trickier to describe and measure! (later)
# Dimensions and Parameters!

<table>
<thead>
<tr>
<th>Architecture And Coupling</th>
<th>Message Passing (loose coupling)</th>
<th>Sensor, Wireless, Overlay ...</th>
<th>Shared Memory (tight coupling)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timing</strong></td>
<td>Synchronous</td>
<td>Partial Asynchronous</td>
<td>Asynchronous</td>
</tr>
<tr>
<td><strong>Identity</strong></td>
<td>Anonymous</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td><strong>Knowledge</strong></td>
<td>Local (Own + neighbour IDs), + size, did estimates?</td>
<td>Global</td>
<td></td>
</tr>
</tbody>
</table>
Asynchronous Time complexity!

- How can we measure time in asynchronous executions?
  - Normalize by greatest message delay! - Assign non-decreasing real times to events s.t. sending and receiving any message takes \textit{at most 1 time unit}. (Still allows arbitrary delivery).
Complexity measures

- Time complexity: worst, average...
- Message complexity:
  - Number of messages: worst, average...
  - Message size etc...
- Space complexity
- Other measures occasionally.
References

**Distributed Computing: Fundamentals, Simulations and Advanced Topics**

**Distributed Computing: A Locality-Sensitive Approach**