Fundamentals of Distributed Computing

Dr. Rajendra Prasath

Spring 2019

About This Lecture

This Lecture covers the essential aspects (of Distributed Algorithms and Systems) that every serious programmer needs to know about **the Fundamentals of Distributed algorithms**, **design principles** and **their analysis**, with an emphasis on **scalable applications development**

What do we learn?

- Some Important aspects of DC:
  - Reliable network
  - Zero Latency
  - Infinite Bandwidth
  - Secure network
  - Fixed Topology,
  - Only one administrator
  - Zero Transport cost
  - Homogeneous Network
- Remember these points when you first develop a distributed application

Reliable Network

- Hardware may fail! Power failures; Switches have a mean time between failures

The implications:

- Hardware: weight the risks of failure versus the required investment to build redundancy
- Software: we need reliable messaging: be prepared to retry messages, acknowledge messages, reorder messages (do not depend on message order), verify message integrity, and so on.

Zero Latency

Latency (not bandwidth): how much time do the data take to move from one place to another? → measures in time

- The minimum round-trip time between two points on earth is determined by the maximum speed of information transmission: the speed of light.
- At 300,000 km/sec, it will take at least 30msec to send a ping from Europe to the USA and back.

The implications:

- Strive to make as few calls over the network (other than LANs) as possible

Infinite Bandwidth

Bandwidth: how much data you can transfer over a period of time (may be measured in bits/second)

- ➔ It constantly grows
- ➔ Bandwidth may be lowered by packet loss: we may want to use larger packet sizes

The implications:

- ➔ Compression: simulate the environment to get an estimate for your needs

Secure Network

How to secure the underlying network?

The implications:

- ➔ You may need to build security into your applications from the beginning
- ➔ As a result of security considerations, you might not be able to access networked resources, different user accounts may have different privileges, and so on
- ➔ How to solve these issues efficiently?

Fixed Topology

Topologies do not change as long as you are in a closed environment

- ➔ In reality, servers may be added and removed often, clients (laptops, wireless ad hoc networks) are coming and going: the topology is changing constantly

The implications:

- ➔ Do not rely on specific endpoints or routes
- ➔ Abstract the physical structure of the network: the most obvious example is DNS names as opposed to IP addresses

Who is the Administrator?

Different Administrators may be associated with the network with different degrees of expertise

- ➔ Might make it difficult to locate problems
- ➔ Coordination of upgrades: will the new version of MySQL work as before with Ruby on Rails?
- ➔ Never underestimate the 'human' factor!

Zero Transport Cost

- Going from the application layer to the transport layer (2nd highest in the five layer TCP/IP reference model) is not free:
- Information needs to be serialized (marshalling) to get data onto the wire
- The cost (in terms of money) from setting and running the network is not zero.
- Have we leased the necessary bandwidth

Everything costs "Money" !!

Homogeneous Network

Homogeneous = of the same kind; uniform

- Even a home network may connect a Linux PC and a Windows PC. A homogeneous network today is the exception, not the rule!

The implications:

- Interoperability will be needed
- Use standard technologies such as XML

FUNDAMENTALS OF DISTRIBUTED SYSTEMS

Can you parallelize all apps?

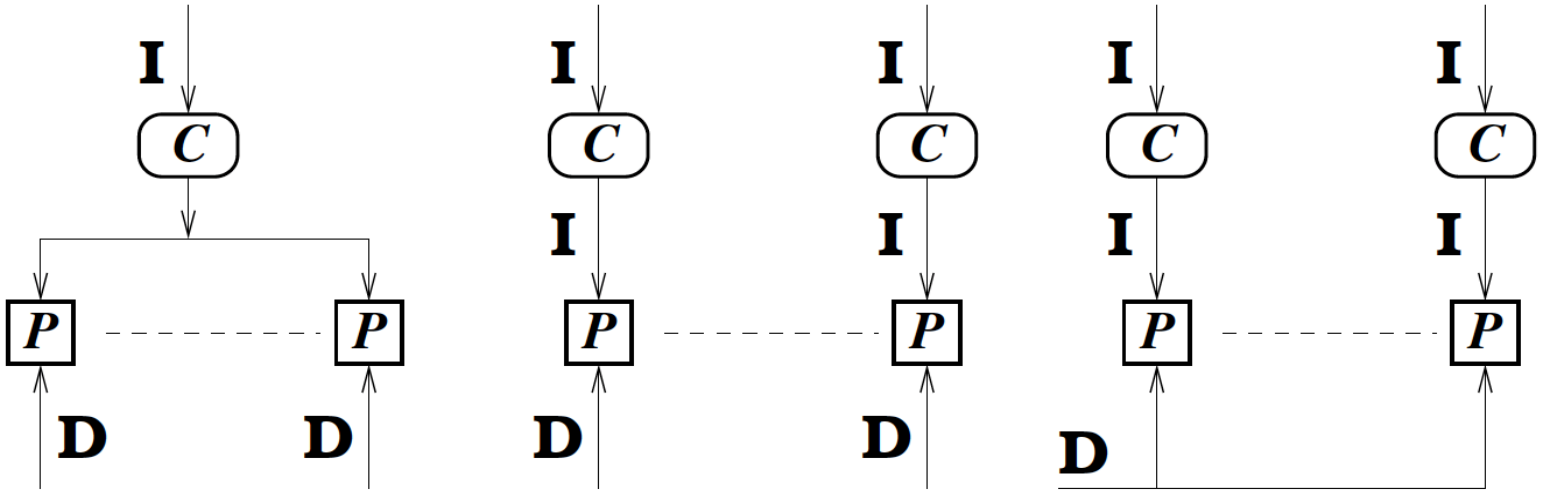
- What can you parallelize?
  - Identify Instructions that could execute in parallel?
- If the proportion of an application that you can parallelize is x ($0<x<1$),

  Maximum speed up = $1/(1-x)$
- Let us assume that a task needs ts time to run on one machine. How much time does it take to run the same task on p CPUs?
- Let running time be tp of a task running on p CPUs. Now $tp=to+ts*(1-x+x/p)$, where to is the overhead added due to parallelisation (communication, synchronization, etc)

Flynn's Classification

- Single Instruction Single Data (SISD) Stream
  - Traditional von Neumann architecture
- Single Instruction Multiple Data (SIMD) Stream
  - Scientific Applications, Vector Processors, array processors and so on
- Multiple Instruction Single Data (MISD) Stream
  - Visualization is an example
- Multiple Instruction Multiple Data (MIMD) Stream
  - Distributed Systems

Flynn's Classification



(a) SIMD *(b) MIMD* *(c) MISD*

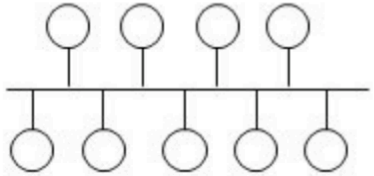
C *Control Unit*

P *Processing Unit*

I *instruction stream*

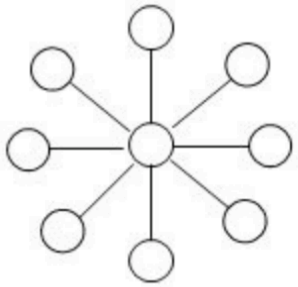
D *data stream*

Interconnection Topologies

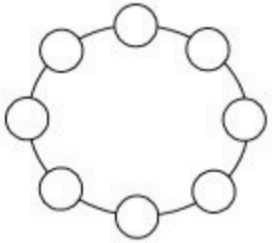


a) Bus

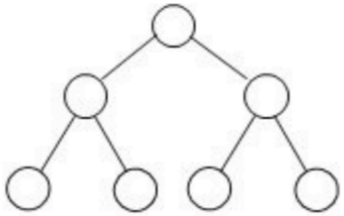
b) Linear array



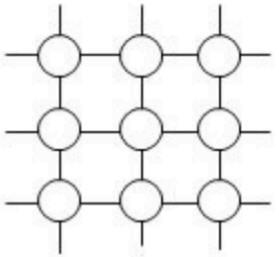
c) Star



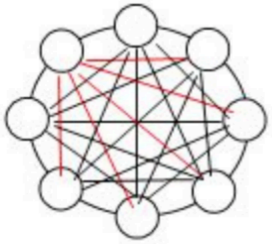
d) Ring



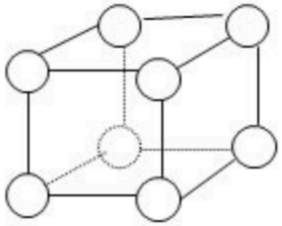
e) Tree



f) Near-neighbor mesh



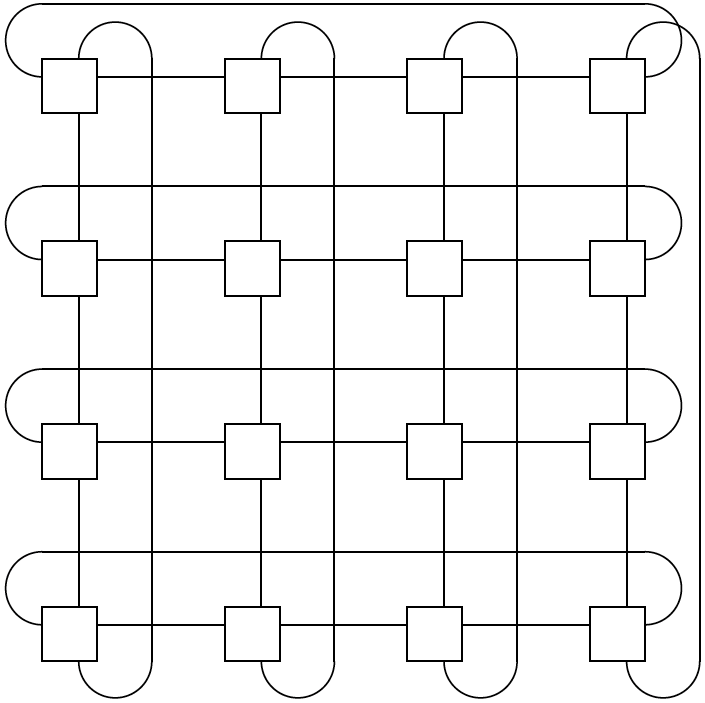
g) Completely connected



h) 3–cube (hypercube)

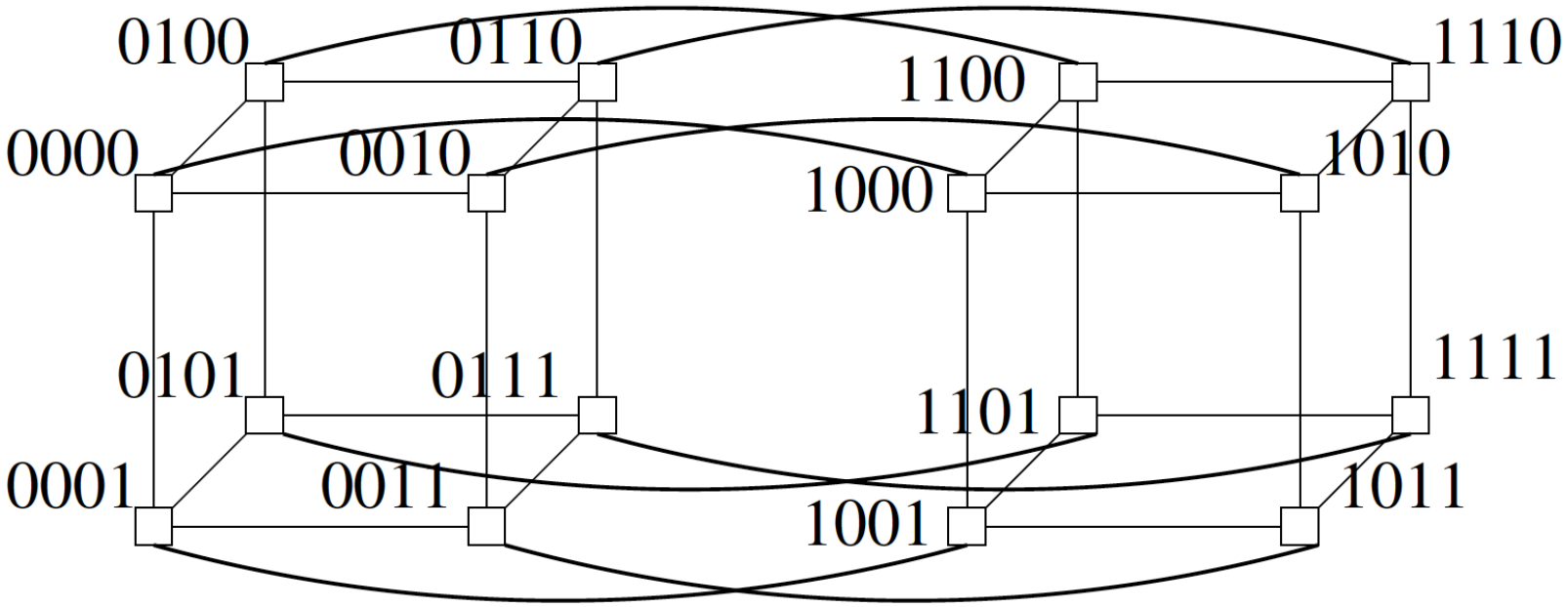
Interconnection Topologies (contd)

➔ Wrap – around mesh (torus)



k-ary d-cubes

➔ Hypercube of Dimension 4



➔ k-ary d-cudes (generalized version)

Message Passing Systems

Basic Primitive Operations

- Send
  - *send* message from process A to Process B

    A → B
- Receive
  - *receive* message at Process B from Process A

    B ← A
- Compute at A and / or B
  - do the specific computations at A and / or B

Message Passing vs. Shared Memory

- Emulate MP on SM
  - Partition the shared address space
  - Send / Receive messages via shared address space
- Emulate SM on MP
  - Model each shared location as a separate process
  - Send: Write to the shared object by sending messages to owner process for the object
  - Receive: Read from shared object by sending query to the owner process of the object

Synchronous vs. Asynchronous

- Synchronous (send / receive)
  - Handshake between sender and receiver
  - *send* completes only when *receive* completes
  - receive completes only when copying of the data to the buffer is over
- Asynchronous (send)
  - No need for handshake between sender and receiver
  - Control returns to the invoking process when data copied out of user-specified buffer

Blocking vs. non-blocking

- Blocking
  - Control returns to the invoking process after the the task completes
- Non-blocking
  - Control returns to the invoking process when data copied out of user-specified buffer
  - *send* completes even before copying the data to the user buffer
  - *receive* may happen even before data may have arrived from the sender
  - How to order EVENTs?

Applications

- Mobile Systems
- Sensor networks
- Pervasive Computing
  - Smart workplace
  - Intelligent Home
- Peer-to-peer computing
- Distributed Agents
- Distributed Data Mining
- Grid Computing
- Security aspects in Distributed Systems

Summary

- Focused aspects in this course:
  - Fundamental aspects while building distributed applications
  - Interconnection topologies
    - Number of Message exchanges
  - Primitives of Distributed Communications
    - Message Passing vs Shared memory systems
  - Properties of a distributed system
    - Many more to come up … stay tuned in …

How to reach me?

➔ Please leave me an email:

rajendra [DOT] prasath [AT] iiits [DOT] in

➔ Visit my homepage @

➔ http://www.iiits.ac.in/FacPages/index-rajendra.html

OR

➔ http://rajendra.2power3.com

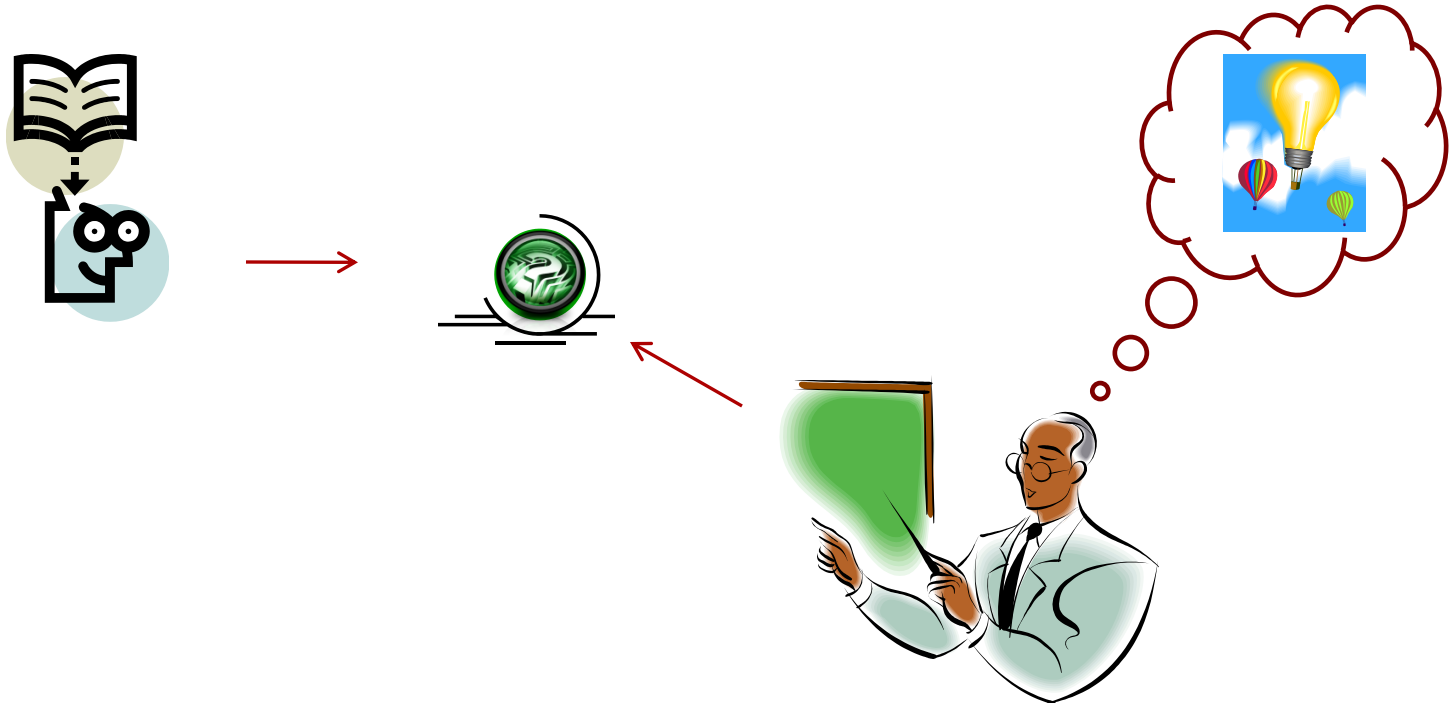
Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Thanks …



… Questions ???