Distributed Systems

Monsoon 2024

International Institute of Information Technology

Hyderabad, India

Pictures not cited explicitly are taken
from the course book or wikepedia.

Welcome to the Semester

- Hope you all had a great summer.
- New students
- New academic year
- New ideas – new plans – new resolutions

Motivation

•Let us think of some distributed systems that we use almost on a daily basis. (Can you name a few?)

Motivation

- Let us think of some distributed systems that we use almost on a daily basis. (A couple that I know)
  - Mobile telephone networks
  - The Internet with its various protocols
  - Banking/Reservation systems

Evolution

- From the early 1970s to present, how we see distributed systems.
  - Email/Bulletin Board systems
  - Internet
  - Grid Computing Systems
  - Peer-to-Peer file sharing systems including Gnutella, Napster, FileDonkey?,
- Main differences between the above systems are as follows:
  - Bulletin board systems existed in the era before Internet. Similar functionality to the current WWW.
  - Grid computing systems still require a little bit of centralized control.
  - Fully distributed systems do not require any central control.

Usual Computation Model

- A distributed system is a collection of N independent processes that communicate with each other via a shared medium or via exchange of messages.
- In most distributed computing systems, the data is not at one single location.
- Sites, processors, nodes, make local decisions based on the data they have, computations they perform, and messages received by them from other sites/processors/ nodes.

Some Variations on Models

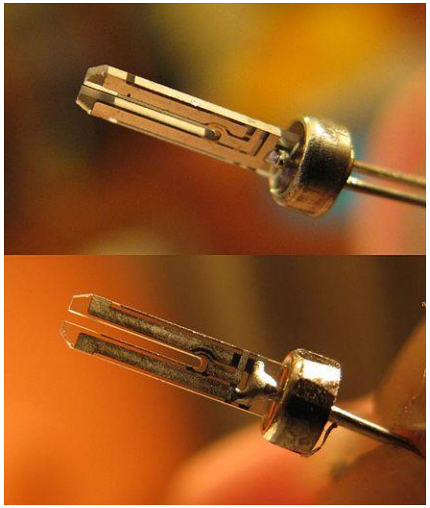
- Several variants of models are possible.
  - It is often said that there are as many models as there are researchers in this broad area of distributed computing.
- Synchronous or Asynchronous communication
- Message size
  - Limited or unlimited
- What is allowed in local computation?
- Message delivery order.
  - FIFO/Arbitrary/Causal/
- Blocking Vs. Non-Blocking Communication
- Faults in the system vs. Fault-free systems
  - Type of faults in a faulty system
- ...

Some High Level View of Challenges

- Quick questions:
- How is time measured in the physical world? In other words, how to accurately measure time?
- How does your computer get the current time on installation/boot ?
- How does it keep the time correct?

Some High Level View of Challenges

- Time in physical world (according to SI system) is now measured as
  - 1 second = Time it takes the cesium 133 atom to make exactly 9,192,631,770 transitions.
- Computer systems do not use the same model.
  - Quartz crystal and its oscillations. 1 second = 32,768 oscillations.
- But clocks based on quartz oscillations can differ over time and due to various reasons including ambient heat.
  - Called **clock drift.**
- Now consider the case with multiple computers separated physically but need to agree on a common time.
- The problem is called **clock synchronization**.



A Quartz crystal resonator.

Some High Level View of Challenges

- Delivering content on the Internet
  - Requires strong caching and search models
  - Video/Image transfer in real-time is challenging.
  - Also, has scope for reuse of content. Need not deliver from the source to the end user always.
  - Take locality into account.
  - Replication and Consistency

Some High Level View of Challenges

- Consider banking systems
  - Typically, accounts are maintained locally.
  - Till 1990’s
    - Transactions require time to take effect.
    - End-of-day batches to tally accounts across branches.
  - Early 2000s’ onwards in India
    - CORE banking
      - Centralized Online Real-time Exchange
      - Each branch runs the software and branches are interconnected
  - The current decade
    - UPI, transfer money via mobile apps,

Some High Level View of Challenges

- Consider banking systems
  - Typically, accounts are maintained locally.
  - Till 1990's
    - Transactions require time to take effect.
    - End-of-day batches to tally accounts across branches.
  - Early 2000s' onwards in India
    - CORE banking
      - Centralized Online Real-time Exchange
      - Each branch runs the software and branches are interconnected
  - The current decade
    - UPI, transfer money via mobile apps,
  - **Challenge**: Correct operation of accounts even under failures.
    - Note that debiting an account and crediting another account across different banks requires updating databases held in different locations and different administrative domains.

Some High Level View of Challenges

- Programming distributed systems
  - Need suitable abstractions.
  - Shared memory abstraction or no shared memory abstraction used.
  - Program has to deal with lots of uncertainty
  - Verification and debugging also get challenging
    - Replicate the same event sequence.

Some High Level View of Challenges

- Dealing with large data
  - We are in the era of big data with data volumes exceeding reasonable limits.
  - One way to deal with the volume is to assume that data is stored at multiple machines in a partitioned manner.
  - Several questions arise
    - Schemes to store the data
    - Search for required data in a fast and scalable manner
    - Efficiency concerns with coding

Some High Level View of Challenges

- Distributed systems cuts across multiple aspects of Computer Science.
  - Programming: Frameworks, APIs
  - Data structures: Hash tables, ...
  - Algorithms: Routing, search, ...
  - Databases: Distributed transactions
  - Operating Systems
    - Resource management, mutual exclusion, file systems
  - Applications: Internet, CDN, Google File System, Bitcoin,...
  - Security
    - Authentication/confidentiality/repudiation/availability/D-o-S
  - Software engineering: Distributed verification, debugging, ...

Course Policies

- Instructor
  - Myself, Kishore Kothapalli
  - Email: <kkishore@iiit.ac.in>
- Lectures
  - All students to attend and participate
- Homeworks
  - Both hand-written and programming based.
  - Strictly no copying.
  - Some of the homeworks will be group based.
- Office hours for me:
  - Tuesday 11 AM to 12 noon, or by appointment

Syllabus (Tentative)

- Module 1
  - Introduction
  - Communication models
  - Time and Synchronization
  - Practice: MPI/Map-Reduce
- Module 2
  - Distributed file systems
  - Consensus, Agreement, Locking
  - Practice: GFS, Chubby
- Module 3
  - Distributed Database systems
  - Practice: NoSQL, MongoDB

Syllabus (Tentative)

- Module 4
  - Limitations of distributed computing
  - Self-Stabilization
  - CAP Theorem

- Module 5
  - Distributed algorithms for graphs
  - Guest lectures for special topics

Course Policies

- Textbook(s)
  - Writing my own book for this subject. Co-authored with Prof. Sarangi, IIT Delhi.
  - Visit www.thedistsysbook.com for details.

- Other Reading Material to be made available as needed
- Grading policy – Tentative
  - Scheduled and unscheduled Quiz Exams: 20%
  - Mid Exam: 20%
  - Homeworks: 15%
  - Project: 20%
  - Final Exam: 25%

Course Policies

- Identified two Teaching Assistants so far
  - Lokesh V, lokesh.v@research.iiit.ac.in
  - Mahen N, mahen.n@research.iiit.ac.in
- Will add 2-3 more TAs in the coming days

- No plagiarism policy – Any work submitted for grading cannot be copied from any other sources.

- Please pay attention to classroom etiquette, including
  - Showing up on time, very important
  - Not get distracted with mobile phones and other devices.

Homework Exercise

- Homework 1: From the discussion in the first lecture, read in more detail one paper out of the five that will be posted and write a small report including:
  - What is the system/theory/result about?
  - What is the main result?
  - What is the significance, what problem does it solve?
  - How do you find it useful?
  - What do you find interesting in the system/result?
  - How can it be improved in some direction.

- To be submitted by August 5, 2024 (via moodle)

Primitives for Distributed Communication

- For communication purposes, most distributed systems support two primitives:
  - Send
  - Receive
- Small extensions/variations via collectives in some systems.
- Send ()
  - Two parameters: destination, message
- Receive()
  - Two parameters: Source, space for holding the message.

Primitives for Distributed Communication

- For communication purposes, most distributed systems support two primitives:
  - Send
  - Receive
- Quick Recall: Do TCP and UDP have send and receive primitives?
- How do you use these primitives?

Primitives for Distributed Communication

- For communication purposes, most distributed systems support two primitives:
  - Send
  - Receive
- User space vs. Kernel space operation
  - Usually, for the Receive() operation, kernel support is needed to hold the data while the user process has not invoked the Receive() operation.
  - For Send(), we assume that the data is copied from the user buffer to a kernel buffer before being placed to the network.
  - This is called as the buffered option.

Primitives for Distributed Communication

- For communication purposes, most distributed systems support two primitives:
  - Send
  - Receive
- Synchronous Primitives Vs. Asynchronous Primitives:
  - Synchronous if the Send() and the corresponding Receive() follow in a hand-shake.
    - The sender knows that the receiver is ready!
  - Asynchronous if a Send() primitive returns control to the process invoking the send after the data has been copied from the user buffer to the kernel buffer.

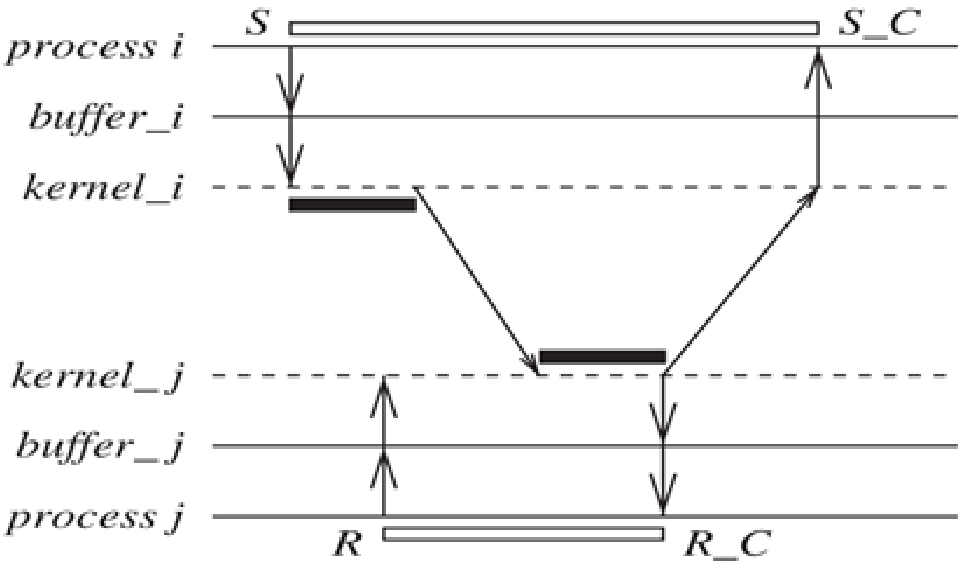
Primitives for Distributed Communication

- For communication purposes, most distributed systems support two primitives:
  - Send
  - Receive
- Blocking vs. Non-Blocking Primitives:
  - A Send() operation is blocking if control returns to the process invoking the send after the processing for the operation has been completed.
  - A Send()/Receive operation is non-blocking if control returns to the process invoking the send/Receive immediately after invocation – without waiting for the processing of the operation to complete.

Primitives for Distributed Communication

- Some of the possible send/receive combinations:
  - Blocking Synchronous Send and Blocking Receive
  - Non-blocking, Synchronous Send, Non-Blocking Receive
  - Blocking, Asynchronous Send
  - Non-blocking, Asynchronous Send
- The book by Kshemkalyani has pictures of the above possibilities. Take a look at them offline.

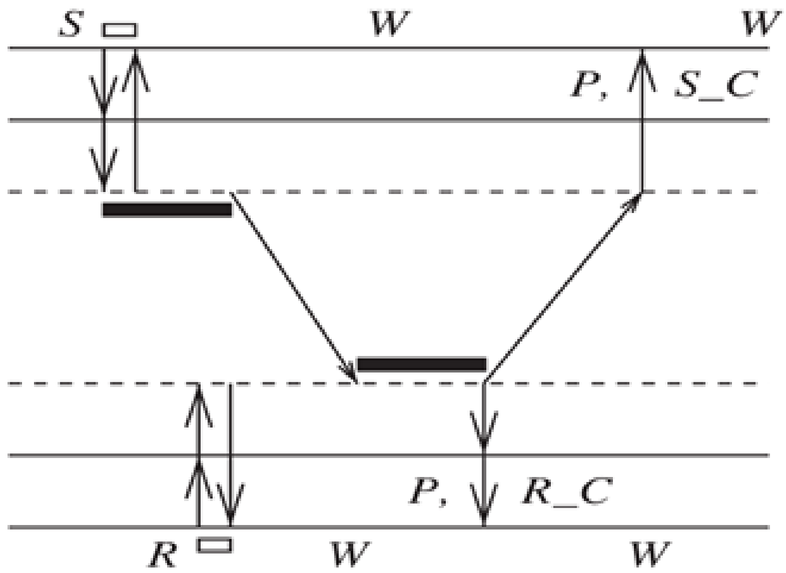
Blocking Synchronous Send, Blocking Recv.



(a) Blocking sync. *Send*, blocking *Receive*

▬	Duration to copy data from or to user buffer
▭	Duration in which the process issuing send or receive primitive is blocked
S	*Send* primitive issued
S_C	processing for *Send* completes
R	*Receive* primitive issued
R_C	processing for *Receive* completes
P	The completion of the previously initiated nonblocking operation
W	Process may issue *Wait* to check completion of nonblocking operation

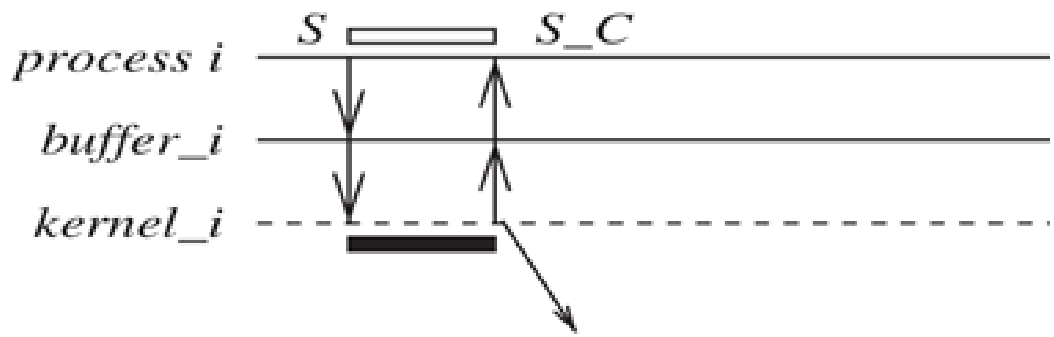
Non-blocking, Sync. Send, Non-Blocking Recv



(b) Nonblocking sync. *Send*, nonblocking *Receive*

(filled bar)	Duration to copy data from or to user buffer
(open bar)	Duration in which the process issuing send or receive primitive is blocked
S	*Send* primitive issued
S_C	processing for *Send* completes
R	*Receive* primitive issued
R_C	processing for *Receive* completes
P	The completion of the previously initiated nonblocking operation
W	Process may issue *Wait* to check completion of nonblocking operation

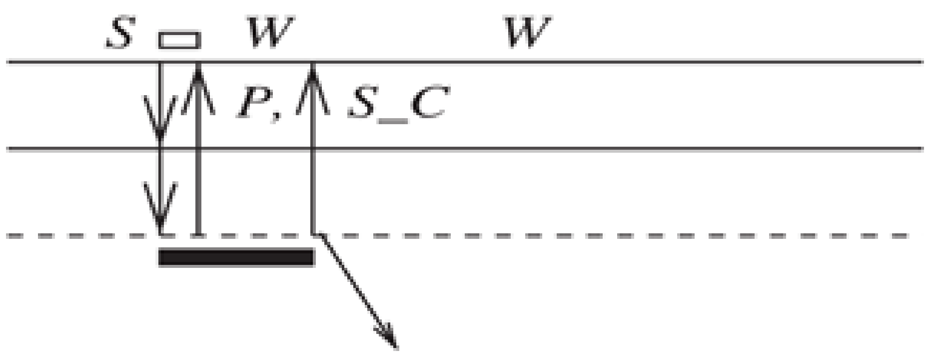
Blocking, Asynchronous Send



(c) Blocking async. *Send*

	Duration to copy data from or to user buffer		
	Duration in which the process issuing send or receive primitive is blocked		
S	*Send* primitive issued	*S_C*	processing for *Send* completes
R	*Receive* primitive issued	*R_C*	processing for *Receive* completes
P	The completion of the previously initiated nonblocking operation		
W	Process may issue *Wait* to check completion of nonblocking operation		

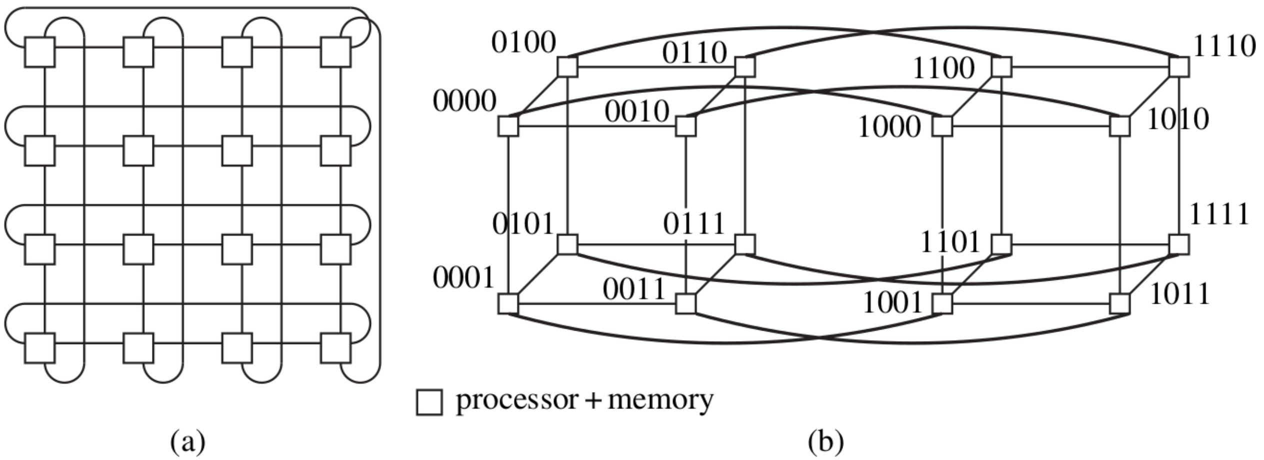
Non-blocking, ASync. Send



(d) Non-blocking async. *Send*

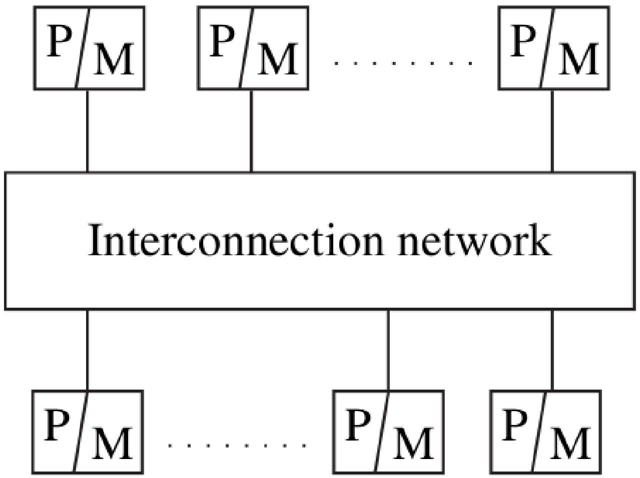
▬ (filled bar)	Duration to copy data from or to user buffer
▭ (open bar)	Duration in which the process issuing send or receive primitive is blocked
S	*Send* primitive issued
S_C	processing for *Send* completes
R	*Receive* primitive issued
R_C	processing for *Receive* completes
P	The completion of the previously initiated nonblocking operation
W	Process may issue *Wait* to check completion of nonblocking operation

Other Models of Distributed Computing



- Models specifying an underlying physical network are also popular. Called as the network model.
- This physical network can be also called as the interconnection network.
- Homogeneous processors and network characteristics.
- Examples of networks: mesh, butterfly, shuffle-exchange, hypercube, ...

Other Models of Distributed Computing



- Multicomputer parallel system:
  - Multiple processors with possibly no shared memory
  - No common clock, possibly.
  - With shared memory, for a given processor, not all locations have the same latency. Called the NUMA model.

Other Models of Distributed Computing

- Other currently less popular models
  - Array model
  - Systolic array
    - This formed the basis of the current TPU architecture too.