2

09/03/2023

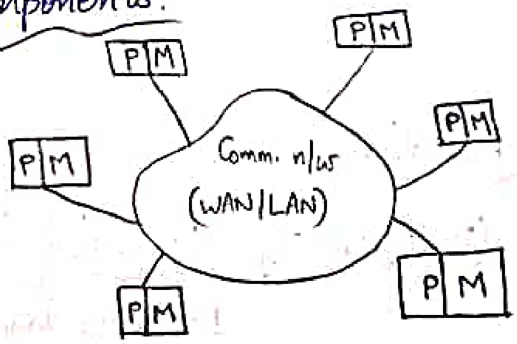
DISTRIBUTED SYSTEMS

Module 01: Introduction to DS

Distributed Systems:

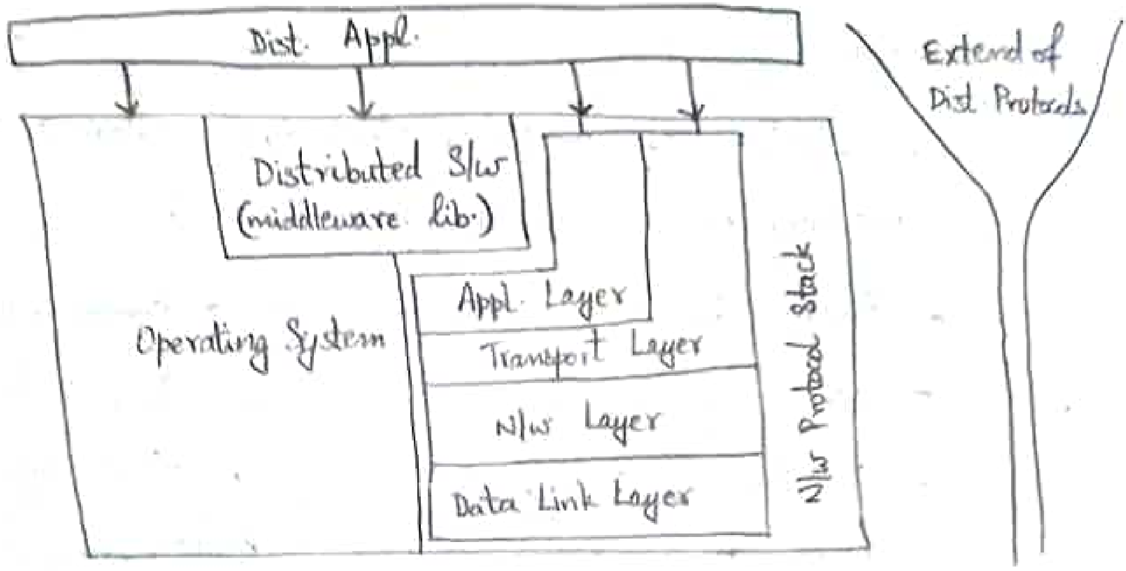
- A collection of computers that do not share a common memory or a common physical clock
- Comm. by msgs passing over a comm. n/w
- Each comp. has its own memory & runs its own OS.
- Collection of independent computers that appears to the users as a single coherent ~~system~~ computer.
- Collection of autonomous processors having features as:
  - * No common physical clock
    - ↳ distribution
    - ↳ inherent asynchrony among processors.
  - * No shared memory
    - ↳ requires msg passing for comm.
  - * Geographical separation
  - * Autonomy and heterogenity
    - ↳ Processors – "loosely coupled"
      - diff. speed, run on diff. OS

Relation to Computer System Components:



* Each comp. has a memory-processing unit
* Dist. s/w known as middleware.

* Dist. execution is the exec. of proc across the dist. sys. to collaboratively achieve a common goal. {computation/run}



* Layered architecture to break down complexity of sys. design.

* Middleware :- drives dist. sys,
  - provides transperancy of heterogenity at platform level.

* Primitives and calls to fns defined in various lib. of middleware layer are embedded in user pgm code.

* Standards : - Object Management Groups (OMG)
  - Common Object Request Broker Arch. (CORBA)
  - Remote Procedure Calls (RPC)

* Currently middleware uses: CORBA, Java & RMI (Remote Method Invocation)
  DCOM (Dist. Component Object Model)

Motivation:

(i) Inherently distributed Computations:

Ex: Money transfer-banks,
Reaching concensus among parties that are geographically distant.

(ii) Resource Sharing:

- Resources: peripherals, complete datasets in db, spl. libraries, data (variables / files)
  - → cannot be fully replicated at sites {not practical, not cost-effective}
  - → cannot be placed at single site {bottleneck}

- Ex: DB2 : dist. db

(iii) Access to geographically remote data and resources:

- Ex: * Payroll data - MNC
  - → replication at each branch: too large, too sensitive
  - → so, central server ← query by branch offices.

  * Spl. supercomputers exists only in certain locations

  * Resource constrained mobile devices & wireless tech.

(iv) Enhanced Reliability:

- Reliable
  - → replicating resources & exec.
  - → geo. dist. resources are not likely to crash at same time under normal circumstances.

- Aspects of reliability:
  * availability - accessable all times
  * integrity - state of rsrc to be correct
  * fault-tolerance - recover from sys. failures.

(v) Increased perf. / cost ratio:

- Higher throughput - not main aim, still
- Better perf. / cost ratio than spl. parallel machines.

(vi) Scalability:

- Mostly WAN: no bottleneck

(vii) Modularity and incremental expandability:

* Heterogenous processors - easily added {running on same middleware algorithms}

* Existing processors - easily replaced by another.

Relation to Parallel Systems:

Characteristics of Parallel Systems:

(i) A multiprocessor System:

- parallel system,
- multiple processors : direct access to shared memory (forms common addr. space)
- do not have a common clock.
- corresponds to Uniform Memory Access (UMA)
  ↳ access latency is same (access to any memory location from any processor).
- processors: close proximity (physical), connected by: interconn. n/ws. run on mostly same OS
- IPC: read/write on shared memory
  Message passing is also possible (emulation)
- H/w, s/w - tightly coupled.
- Interconnection Networks
  * n-input, n-output ⇒ log n stages and log n bits for addressing
  * Routing at stage k on 2x2 switch uses only k^{th} bit ⇒ computation at clock speed in h/w.
  * Ex: Omega, Butterfly, Clos, Shuffle-exchange n/ws.

Paths from diff. i/ps to any 1 o/p forms a spanning tree
↳ COLLISION Occurs

* Omega interconn. fn:

→ Connects: n-processors to n-memory units

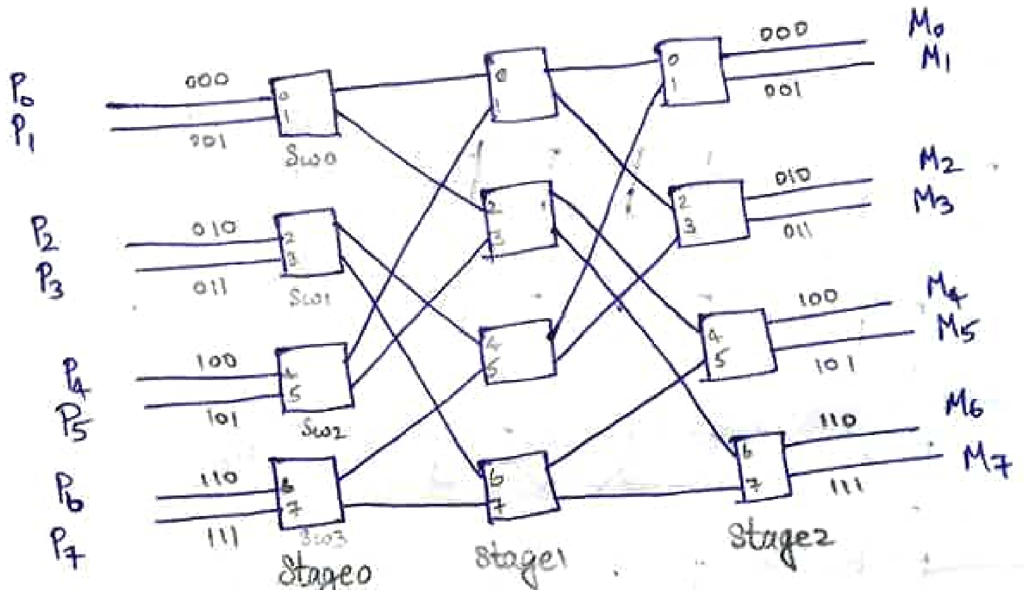
→ Has: $\frac{n}{2}\log_2 n$ switching elements of size 2x2 arranged in $\log_2 n$ stages. $\{s \in [0, \log_2 n-1]\}$

→ Left rotation function: output i of a stage to input j of next stage,
(in binary)

$$j = \begin{cases} 2i, & 0 \le i \le n/2 - 1 \\ 2i+1-n, & n/2 \le i \le n-1. \end{cases}$$

→ Routing: Input line: i, Output line: j, Stage number: s,
In stage s, $s+1^{th}$ MSB of j is 0 ⇒ upper o/p wire
$s+1^{th}$ MSB of j is 1 ⇒ lower o/p wire.

→ Ex: for n=8 ⇒ $\frac{8}{2}\log_2 8 = 4\cdot 3 = 12$ switching elements
⇒ $\log_2 8 = 3$ stages



Route from P_5 to M_6: Input: i = 101, Output: 110

P_5 → (Stage 0, Switch 2) $\xrightarrow{lower}$ (Stage 1, Switch 1) MSB considered: 1
MSB considered: 1

↓ lower

M_6 $\xleftarrow{upper}$ (Stage 2, Switch 3)
MSB considered: 0

* Butterfly interconn. fn:

→ Interconn. depends on stage number s and n

→ $M = n/2$ switches per stage

→ Switch denoted by tuple: <x,s> , $x \in [0, M-1]$ and $s \in [0, \log_2 n - 1]$

→ 2 outgoing edges from any switch <x,s> are as follows:

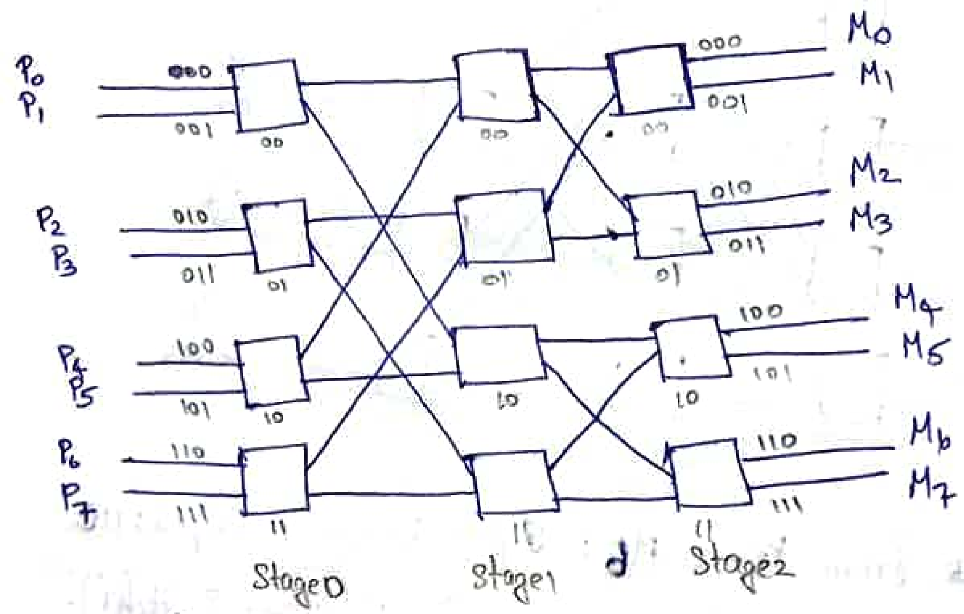
* <x,s> to <y,s+1> if

  x=y (or)
  x XOR y has exactly one bit which is in $(s+1)^{th}$ MSB.

→ For stage 's', apply rule above for $\frac{M}{2^s}$ switches.

→ Routing function: In a stage s, if $(s+1)^{th}$ MSB of j is 0 then, upper o/p wire otherwise, lower o/p wire.

→ Ex: n=8 ⇒ M=4



Route from P_5 to M_6: Input: i=101, Output: j=110

P_5 → (Stage 0, Switch 2) [1st MSB : 1] —lower→ (Stage 1, Switch 2) [2nd MSB : 1]

↓ lower

M_6 ←upper— (Stage 2, Switch 3) [3rd MSB = 0]

(ii) A multicomputer Parallel System:

- parallel system,
- multiple processors: do not have direct access to shared memory (may/may not form a common address space)
- do not have a common clock.
- processors: close physical proximity tightly coupled, connected by: interconn. n/ws.
- Comm.: Common addr. space (or) msg passing
  ↳ NUMA (Non-UMA)
- Ex: NYU Ultracomputer, Sequent Shared Memory machines.
- Mesh wraparound: k×k mesh, k^2 processors, Max. path length b/w any 2 processors is $2(\frac{k}{2}-1)$

{Hypercubes} Ex: Hamming dist b/w 0101 & 1100 = 2

(iii) Array Processors:

- parallel computers, physically co-located
- very tightly coupled
- have a common system clock
- may not share memory
- comm. via msg passing.
- Ex: Array processors & systolic arrays that perform tightly synch. proc. & data exchange in lock-up steps (DSP, Image processing).

Parallel Systems - not economically viable
→ overall market for appl. that need highspeedups → less
→ economy of scale & highproc. power → not cost-effective.

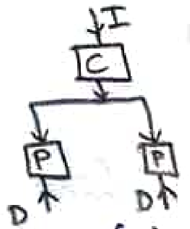
Flynn's Taxonomy:

- whether processors execute same/diff. instr. streams at the same time
- whether or not processors processed the same data at same time.

(i) Single Inst. Single Data Stream (SISD)

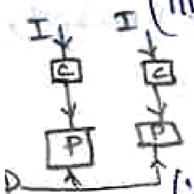
- single CPU, single memory unit
- conn. by system bus.

(ii) Single Inst. Stream Multiple Data Stream (SIMD)



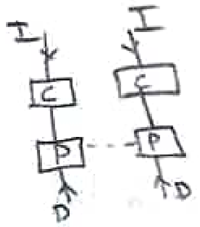
- multiple homogeneous processors
- execute in lock-steps on diff. data items.

(iii) Multiple IS Single DS (MISD)



- diff. operations in parallel on same data.

(iv) Multiple IS Multiple DS (MIMD)



- various processors execute diff. code on different data.
- no common clock.

Message Passing vs Shared Memory Systems:

Message Passing	Shared Memory
* Mainly used in dist. env., using single shared mem. not possible	* Used when all comm. proc. resides in a single computing sys
* More delay : syscall to kernel	* Less delay : only 1 syscall while creating shared mem.

* Useful for passing small amt of data	* Large amt of data - shared
* Comm. link required	* Comm. link not req.
* Itself provides mechanisms for comm. & synch.	* R/W code to be specified by appl. pgm.

(i) Emulating MP on SM:

- shared addr. space partioned into disjoint parts, one to each processor.
- S/R - writing/reading to proc.'s addr. space
- separate location for mailbox.

(ii) Emulating SM on MP:

- Each shared location can be modeled as separate ~~class~~ process
- "write": send update msg to owner
- "read": send query msg to owner.
- In DS, this is only an abstraction, as emulation is expensive and latency is high.

Primitives for Distributed Communication:

* Send(): 2 params - dest, buffer in user space
* Receive(): 2 params - src, user buffer.

S/R * Buffered Option: User buffer --copies--> kernel buffer --copies--> n/w

S * Unbuffered Option: User buffer --copies--> n/w

* Synchronous primitives:
  - Send(), Receive(): handshake with each other
  - Send completes after Receive invoked & completed.

- Receive completes when data is copied into receiver's user buffer.

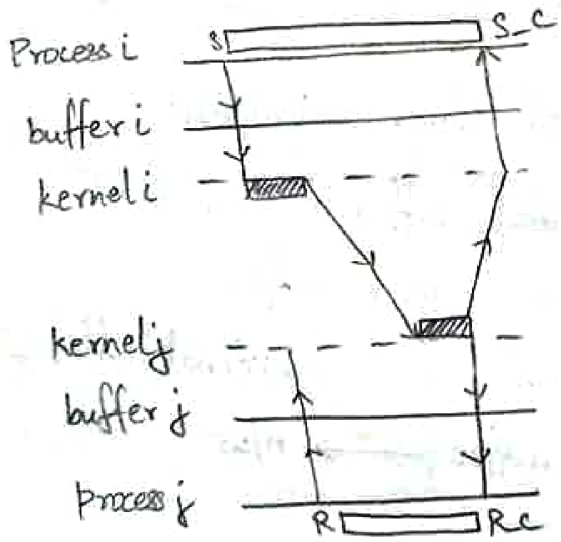
* Asynchronous primitives:
  - Send() - if control returns back to invoking process immediately after data to be sent is copied out of user specified buffer.
  - meaningless for Receive()

* Blocking primitives:
  - if control returns after processing of primitive completes.

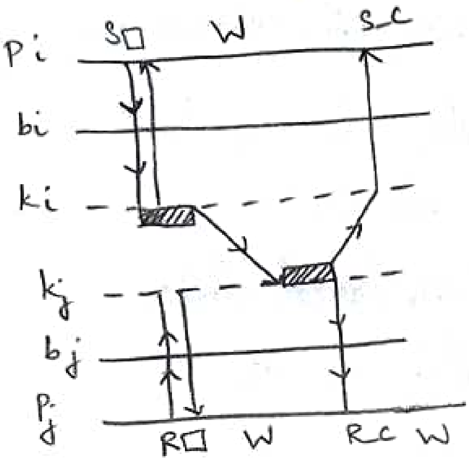
* Non-Blocking primitives:
  - if control returns immediately after invocation

* Handle: for non-blocking primitives, to check status of completion of call.
  - ↳ Periodic check
  - ↳ Wait. - blocks until one of the param handle is posted

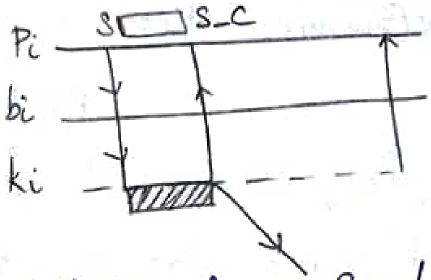
* Blocking synch. Send; Blocking Receive



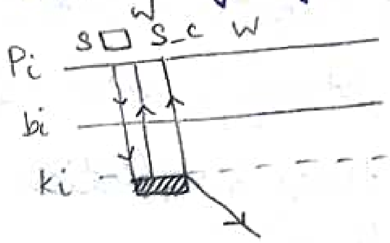
* NonBlocking Synch. Send:
  Non Blocking Receive



* Blocking Async. Send:



* NonBlocking Async Send:



Synchronous vs Asynchronous executions:

→ Asynch. exec.:

* no processor synchrony
* no bound on drift rate of processor clocks
* finite, unbounded msg delays.
* no proper upper bound to execute a step.

→ Synch. exec:

* processors synchronized
* bounded clock drift rate b/w any 2 processors
* msg delivery occur in 1 logical step/round
* known upper bound to execute a step.

Design Issues and Challenges:

→ Categories:

* Greater components related to systems design and OS design.
* Greater components related to algo. design.
* Emerging from recent tech. advances.

DS Challenges from System's Perspective:

* Communication : among proc. in n/w
  Ex: RPC, ROI, msg-Ori, stream-ori
* Processes: mngmt of proc./threads, code migration
* Naming: transparent, scalable manner.
* Synchronization : Mutual exclusion, logical clocks
* Data Storage & access: fast, scalable manner
* Consistency & replication
* Fault Tolerance
* Security
* API and transparency
  → Access transparency: hides diff. in data rep. on diff. systems and provides uniform op. to access system resources.
  → Location transparency: makes loc. transp. to users (of rsrcs)
  → Migration transparency: allows relocation of rsrcs without changing names
  → Relocation transparency: Relocate rsrces as they are being accessed.
  → Replication transparency: Do not let the user become aware of any replication
  → Concurrency transparency: Masking concurrent use of shared rsrcs to users.
  → Failure transparency: Sys - reliable, fault tolerant.
* Scalability and modularity : Rsrcs must be as dist. as possible (algo., data, services).

Algorithmic Challenges:

* Designing useful execution models and frameworks
* Dynamic dist. graph & routing algo.
* Time & global state in a DS.
    - ↳ Logical Time :- relative time, reduce/elim. overhead of physical time.
        - capture logic & inter process dep. within dist. pgm
        - track relative progress at each process.

* Synch. & Coordination mechanisms:
    - physical clock synch.
    - Leader election : all processes needs to agree on which process will play the role of a distinguished process - leader
    - Mutual exclusion
    - Deadlock detection and resolution
    - Termination detection
    - Garbage collection : Garbage - objects that are no longer in use and that are not pointed to by any other process.

* Group comm., multicast, ordered msg delivery
* Monitoring distributed events & predicates
* Dist. pgm design and verification tools
* Debugging dist. pgm: - concurrency in action
    - large # possible executions by interleaved concurrent actions.

* Data replication, consistency models and caching

* World Wide Web Design - caching, searching, scheduling

* Dist. shared mem. abstraction:
  - Wait-free algo.
  - Mutual exclusions
  - Register constructions
  - Consistency models.

* Reliable & Fault Tolerant DS:
  - Concensus algo. (Triple Modular Red.)
  - Replication and replica mngmt - TMR
  - Voting and quorum systems
  - Dist. db & dist. commit
  - Self-stabilizing sys: keep in good state
  - Checkpointing and recovery algo.
  - Failure detectors

* Load Balancing:
  {Higher Throughput, less user perceived latency}
  - Data migration: Ability to move data around in the sys., based on access patterns of users.
  - Computation migration: Ability to relocate proc. to perform redist. of workload.
  - Dist. scheduling: Better turnaround time by using idle proc. power in sys. more eff.

* Real Time Scheduling

* Performance
  - Metrics
  - Measurement methods / tools

Applications of DS and newer challenges:

* Mobile systems
  - routing, loc. mngmt, channel alloc., localization, position estimation, mobility mngmt.
  - Base Station/Cellular approach: base station comm.
  - Ad-hoc network approach: mobile nodes (no base station)

* Sensor networks
  - position estim., time estim.

* Pervasive Computing: class of computing where processors embedded in and seamlessly pervading through the environment perform appl. fns in background

* Peer-to-Peer Computing
  - obj. storage mechanisms, dynamic reconfig., privacy, security.

* Publish-Subscribe, content dist., multimedia

* Distributed agents.
* Dist. Data Mining
* Grid Computing
* Security in DS – confidentiality, authentication, availability.