cluster Programming

Message-Passing Model for High Performance Cluster (move data to computing)

- All process on same program
  difference subtasks → have to communicate → network

Message Passing Interface Process (MPI Process) ~คล้าย pthread คือ เรียกใช้ function

1) mpirun [programname] -np 3 on cmd
   number of process = node
2) แต่ละ process รู้จักกันเอง
3) ทำงานตั้งแต่คำว่า MPI_Init() ถึง MPI_Finalize()
4) ต้องแบ่งงานได้ แต่ละ Process มี Rank (MPI_Rank()) ~ Process ID

Communication

One-to-One / Point-to-Point	Collective
MPI_Send (void *buf, int count, MPI_Datatype dtype, int dest, int tag, MPI_Comm comm) *buf: start addr; count: length; dest: destination rank; tag: msg label; comm: group of process	1) One-to-Many • Broadcast vs Scatter ทุกคนเหมือนกัน / ต่างกันได้ • Gather vs Reduction
MPI_Receive (void *buf, int count, MPI_Datatype dtype, int source, int tag, MPI_Comm comm, MPI_status *status) source: from who (rank); status: return information (ack)	2) All-to-All • All gather

Map Reduce Programming Model for Big Data Cluster (move computing to data)

- ดังเพราะ Google ใช้ทำ Webpage indexing → Application : Inverted index
- High level
  * หลายปัญหาไม่ควรใช้วิธีนี้ เพราะไม่เป็นธรรมชาติ
- Distributed File System
  * ช้า เพราะเกิดการ R/W หลายครั้งบน Disk
- ใช้ประโยชน์ของ **Locality** of Data → minimized communication overhead
- Hadoop = de facto standard

Process
- Map : Applied map function to the local data (ทำบนแต่ละ node)
- Reduce : ย้าย Key เดียวกันมาอยู่ด้วยกัน(บน node ใหม่) ~ Sort, groupping → Disk R/W
- Shuffle : Process each group of data (execute)

Make it Fault tolerance –

Task Crash	Node Crash	Slow tasks
Retry on other nodes	Relaunch on other nodes	ทำบนหลาย node แล้วเลือกอันที่เร็วที่สุด

Apache Spark

- ใช้ Resilient Distributed Dataset (RDD) แทน Hadoop หลังทำ operation จะเก็บเป็น RDD ใหม่ ไม่เขียนทับ
- มี operation มากขึ้น → More flexible
- **lazy evaluation** ไม่ทำจนต้องทำงานจริง เหมือนสร้างกราฟเก็บไว้ ทำให้ optimize ได้

Process

Inputs (list / HDFS) —convert→ RDD —filter→ filtered RDD —map→ mapped RDD —action→ Result (Non-RDD)

Transformation (input RDD, output RDD)
1) Narrow (no-partitioning ทำบน node เก่าได้)
   Map, FlatMap, Map Partitions, Filter, Sample, Union
2) Wide
   Intersection, Distinct, ReduceByKey, GroupByKey, Join, cartesian

Action (input RDD, output non-RDD)
- Reduce, Collect, Count, First, Take, Take Sample, CountByKey, Save As Text File

Introduction to Distributed System

Distributed system
- not centralized
- autonomous computing **elements** (not just computer)
- connected by communication **network** (not just local)
- appears as a **single** coherent system
- **passing messages**
- components (software) located at networked computer

Example
The internet, Email, E banking, Multiplayer online game, VDO streaming, Social Network, Cloud storage, Sensor Network, Blockchain

Trends
Ubiquitous Computing มีคอมเล็กๆทุกที่
Cloud Computing มีคนคิดแทนให้ (Computational service server)

เหตุผลที่ใช้ Distributed System
- Resource Sharing
- Communication
- Flexibility to build system
- Scalability
- Reliability
  เปลี่ยนเครื่องเสียได้ง่าย / เพิ่มลดได้ง่าย

Consequences การใช้ Distributed Sys. ส่งผลให้
- Concurrent Program Execution
- Local information ต้องเลือกชุดข้อมูลเพื่อตัดสินใจ
- No global clock → ลำดับของเหตุการณ์
- Independent failure

Challenge (Parallel – performance, Distributed – Failure)

Heterogeneity = ความหลากหลาย
Key : Standard and Middleware
- Network – use internet **protocol**
- Computer Hardware – Big Endian / Little Endian } **converter**
- OS – set standard
- Programming Language – Character Representation, Data Structure
- Implementation – Developer (~Coding Style)

Transparency ทำให้ดูเป็น single system
Key: user รู้น้อยที่สุด สามารถทำได้ในวิธีเดียว แม้หลังบ้านจะมีหลายแบบ

Access (How to access resource)
Location (Path / Link)
Concurrency (มีคนเข้าพร้อมกันบ้าง)
Replication (ไม่ต้องบอกว่ามีกี่ชุด)
Failure (ไม่ต้องบอกว่าตรงไหนพัง)
Mobility (การเคลื่อนย้ายตำแหน่ง)
Performance
Scaling (ระบบใหญ่ขึ้น)

Security
Confidentiality เป็นความลับ
Integrity ไม่ถูกแก้โดยไม่ได้รับอนุญาต
Availability

Scalability
Extend physical Resource
Extend software Resource
Redesign (to avoid bottleneck)

Failure Handling
Detect = ตรวจสอบ ex. Checksum, Message ID
Masking = ปกปิด
- Retransmission ส่งใหม่
- Replication ลอกจากอันอื่น
- Drop ทิ้งไปเลย อาจจะเสีย

Tolerating = แจ้งกลับไปเลย
Recovery = rolled back
Redundancy = ป้องกัน

Concurrency การเข้าถึง resource เดียวกัน

Fallacies (ความเข้าใจผิด)
1) Network is reliable
2) Latency = 0 → ส่งผลต่อ Timeout
3) Bandwidth = ∞
4) Network is Secure
5) Topology doesn't change
   ↳ โครงสร้างขององค์ประกอบใน Network { Ring, Star, Tree
6) There is one administrator
7) Transport cost = 0
8) Network is homogeneous

Distributed System Architecture

Architectural Model ประกอบด้วย

1) Pattern of Placement of components (Task)
2) Communication Paradigms (Interprocess / Remote procedure)
3) Roles and Responsibility (Client / Server / Peer)

Middleware
- ก่อให้เกิด Abstraction
- เปิดเผยโปรแกรมผ่าน Middleware API

Overlay Network logical network over physical network
1) Structure – well define ex. tree, ring, star
2) Unstructure – randomly connection

Client - Server Architecture

- Servers may be clients of other servers
- Can have more servers than clients → improve performance + increase reliability by replication

ex. Web — HTTP protocol
Network File System (NFS) (ระบบฝากไฟล์) — Remote procedure call (RPC) ส่งคำขอไปทำงานบนเครื่องอื่น

* proxy servers and cache
ลดจำนวนครั้ง และเพิ่มความเร็วในการติดต่อ server โดยการเก็บข้อมูลไว้ใน Local หรือ proxy servers
↳ ลด load บน Web Server

Multitier Architecture
- layers of servers
- n-tier

Peer-to-Peer Architecture
- All processes have similar roles = both provider and receiver

ex. P2P File Sharing
1) Napster P2P File Sharing มี Master index เพื่อหาว่าใครมี File → centralized, loaded server
2) P2P with Circular (Ring) ส่งวนรอบกันไปเรื่อยๆ
3) P2P with Hierarchical (star) มี superpeer nodes (Ultra peer nodes) รู้จักกับลูกอื่นเยอะ

Hierarchical Architecture

ex. Network Time Protocal Servers
- Root nodes and leaf nodes

Communication

Fundamentals of Communication (Any communication in the world)

Purposes, Media, Message, Protocols, Addressing, Transmission system
- Message: -Format, -Content
- Transmission system: -How to deliver

Basic Networking Concept

packet-switching technique

message
↓ divided, encapsulated ~ใส่ซองจดหมาย เขียนที่อยู่
packet
↓ transmit via switches
receiver

Encapsulation
- Each layer build a protocol data unit (PDU) by adding header (and trailer)

~เหมือนใส่ซองจดหมายซ้อนหลายชั้น

Network Protocols : rules and formats
- sequence of messages
- addressing of endpoints
- format of data in messages

Network Layers
- each layer has its own protocols to communicate at the same level
- provides services to upper layer, uses services from lower layer
- propagated in sequence

Advantage
1) Reduce complexity
   Easy to understand and implement
2) Flexibility
   Implement and modify do not effect other layers

ISO Open systems Interconnection (OSI) protocol model (VS) **TCP/IP protocol model**

OSI		TCP/IP
App	Upper (Middleware: App, Presentation, Session)	App — ex. HTTP, FTP, DNS, NTP, SSH
Presentation		↕ Message streams: UDP or TCP
Session		
Transport		Transport
		↕ UDP or TCP
Network		Internet
		↕ IP
Data Link		Network Interface
Physical	Lower	Underlying Network

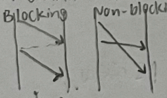
- Physical └ Circuits and Hardware that transmits data (Fiber / Cable)
- Data Link → transmit packets between nodes directly connected (Ethernet protocol)
- Network → Transfer packets between computer through routers (Internet protocol : IP)
  - ~~Switch~~ in Datalink layer
  - ~~Router~~ in Network layer
- Transport → **Lowest Level** of Message ~~(packet)~~
  Messages are addressed to communication port

User datagram protocol (UDP) (VS) Transmission control protocol (TCP)

UDP	TCP
- reduce overhead	- **connection-oriented** ต้อง connect ก่อนคุย
- realtime application network management	- reliable
Failure Handling : checksum	- **stream**-oriented
fail → drop, full buffer → drop	Failure Handling → overhead
- out-of-order **message**	Checksum, Sequence Number
	Timeout + Retry
	- retransmission if necessary
	- acknowledge to sender

Middleware Layer (between OS – Application)
- มี Library ในการสื่อสาร
- support communication model

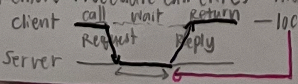
① Lower Middleware Layer



- Message passing
  - Sync (รอส่งจนเสร็จ, รอรับจนเสร็จ)
  - Async (ส่งทันทีที่ data ใน buffer พร้อม)
  - Blocking receive = รอรับข้อมูลทีละอัน
  - Non-blocking receive = รับๆมาเลยแล้วค่อยดูว่าใครส่งมา (polling)
- Socket
  - Abstraction สำหรับการเขียนโปรแกรม
  - Endpoint for communication between process
  - IP Address + Port Number
- Message Queue manage by queue manager
  - write allowing (1+ process)
  - async
  - put: append to specific queue
  - get: block until nonempty and remove 1st message
  - poll: get but not block
  - notify

② Upper Middleware Layer
- Message-Passing Interface (MPI) — group collective
- Indirect Communication — through intermediary
  - space uncoupling, time uncoupling
- Publish-Subscribe System (Pub-Sub)
  Distributed event-based system
  - Publisher: publish, advertise
  - Subscriber: subscribe, Notify

Remote Procedure Call (RPC) — hidden communication
— local procedure in server



Web Architecture

Internet = Inter-networking infrastructure based on IP and TCP/UDP

World-Wide Web = Internet Application based on HTTP (hypertext transfer protocol)

- static web pages = retrieving and displaying
- Dynamic web page = run scripts (the same as run local application)

Web Enabling Technologies

- Web server — Resource provider
- Web browser — User Interface
- HTTP — App layer protocol over TCP/IP
- HTML — Data format for web page
- URL (Uniform Resource Locator)
- MIME (Multipurpose Internet Mail Extentions) ~ พัฒนาต่อจาก E-mail
- Server-side Script — CGI PHP JSP Node.js
- Client-Side script — JavaScript VBScript

HTTP Protocol

- Text-based protocol — Simple
- HTTP request to web server
- HTTP response from web server as HTML
- พัฒนาเป็น HTTPS (HTTP on a secured connection SSL)
  มีการระบุ Port (HTTP ใช้ port 80)

หลักการทำงาน

static web :
1) ส่ง request ขอ index.html
2) ได้ HTML มา เช็คว่าต้องขอไฟล์อะไรเพิ่มหรือไม่
3) ขอข้อมูลเพิ่ม

Dynamic web :
1) ส่ง request ขอ script file ไป web server
2) ส่งคำสั่งไป execute บน AppServer อื่น
3) ได้ HTML คืนมาที่ webserver แล้วส่งกลับไป web browser

Architecture

1) Single-Tier Architecture
- monolithic application ทุกอย่างอยู่บนเครื่องเดียวกัน
- easy to manage
- scale-up only
- dumb remote terminal - ไม่ได้ใช้ logic

2) Multi-Tier Architecture → Presentation + Logic + Data

2.1 Two-Tier Architecture

Fat Clients
- (Presentation + Logic) and (Data)
- data sharing
- need fast clients
- app deployments หลายครั้ง

Thin Client
- (Presentation) and (Logic + Data)
- ภาระงานทั้งหมดไปอยู่ที่ server

2.2 Three-Tier Architecture
- แยก Presentation, Logic, Data
- Scale-out, Flexiblility
- More complex + Failure

2.3 Four-Tier Architecture
- Browser + Webserver + App server + Database
- แก้ปัญหาการแบ่งภาระงานบน server

Internet Service Application / service Oriented Architecture

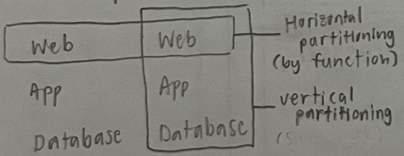
- Domain Application : E-mail, search, news, e-commerce, data storage
- Hardware : Cluster
- Software : Customized, multitier
- Software Release Frequency : Days
- Key metrics : Availability, functionality, scalability, manageability, throughput

Performance Metrics

1) Concurrent users (session) = Currently execute user
2) Concurrent Connections = (1 user = n connections)
3) Throughput = **complete** request / time
4) Service Time / Response Time
   more resource → larger, system upgrade → faster
5) Availability = $\frac{\text{uptime}}{\text{uptime} + \text{downtime}}$ four nines = 99.99%
   downtime → Failure
   → Maintenance

Scaling Web Application

1) Web Server / App Server optimization
: Workload partitioning (distributed workload)



2) Data layer optimization
: server load balancing

- DNS Round-robin ; not high availability, มีปัญหากับ DNS caching
- Reverse Proxy : user → proxy → back-end servers
  pre-config proxy
- Content Switching : กระจายงานตาม Content type / Request type / User type

3) Network optimization
: Memory / Storage Hierarchy closer = less latency

web caching
- + reduce latency, save bandwidth
- ♡ Popularity = everyone wants to access
- ♡ Locality = more recent content
- Client : Proxy cache + Local Cache
- server : Caching Load Balancer

Content Delivery Network (CDN)
- กระจาย Content ไปเก็บไว้ใกล้ๆ end-users
- high availability → high performance

Types of Large-Scale Internet Services
แบ่งตาม Usecase

Types	Example	Front	Back
Online service / Internet Portal	Hotmail	functional partitioned ←load balancing→	Single File, Single Database
Content hosting service	File Sharing	all the same	data partitioned, m servers to n storages
Read Mostly — read > write	Wikipedia	all the same	full replication, all to all

Time and Event Ordering

Event = the occurrence of a single action
- ex. sent / receive message , change of memory
- Some need timestamp accurately ex. auditing

Real Time Clock (RTC) = electronic oscillator circuit
- using the vibration of quartz crystal to control the frequency → not very accurate due to variation of quartz (คุณภาพ) and temperature
  → Sync with highly accurate external clock = Atomic Clock (Cesium)
- มี IC ที่ทำหน้าที่เป็น clock generator

Clock skew = Time A − Time B (in duration)

Clock Drift = Frequency A − Frequency B (in speed)

want **Monotonicity** time always increases ~ไปทางเดียวกัน

Continuity time doesn't jump ~สม่ำเสมอ

→ ถ้าไม่ตรง 2 เงื่อนไขนี้ จะเกิด Conflict ของการข้ามเวลา

ดังนั้น หากนาฬิกาตามหลัง ให้เร่ง Rate (ค่อยปรับตัวเวลา)

หากนาฬิกาเร็วเกิน ให้ลด Rate

Standard Time

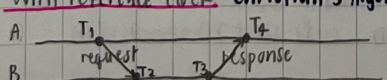
Coordinated Universal Time (UTC)
- International Atomic Time (TAI) = average of 400 atomic clocks worldwide
- Universal Time (UT-1) = based on rotation of Earth and around the Sun
- Adjusted by adding leap seconds (same concept as leap year)
  = adding 1 second to some of 30 June and 31 Dec
- broadcast through short-wave radio ($0.1 - 1.0 \times 10^{-3}$ s delay) and GPS satellites (1×10^{-6} s delay)
  → computers sync to this

BIPM Paris —calculate→ TAI

UT-1 —Set Leap Second→ TAI → UTC → UTC (NIMT)

Clock Synchronization and Reference Clock

with reference clock: Christian's Algorithm



with pair (offset, delay)

Offset $= \frac{1}{2}((T_2 - T_1) + (T_3 - T_4))$; A เร็วกว่า B ; หรือช้ากว่า

Delay $= \frac{1}{2}((T_4 - T_1) - (T_3 - T_2))$

ทำ n รอบ เลือกรอบที่ delay น้อยที่สุด

if offset > 0 , set time = currTime + offset

else , slow down clock

without reference clock: Berkeley Algorithm

1) leader asks times from others
2) average
3) sendback the offset

Network Time Protocol (NTP)

- to sync clocks over the internet, use **UDP** (user datagram protocol) = Less delay
- many sync time servers → hierarchy level = stratum
- reliable and scalability

Logical Time

- physical time has delay
- relatively order, not absolutely time

Happened-Before Relation by Lamport

- potential causal ordering
- denote by →

HB1 (intraprocess) : If ∃process P_i $(a \rightarrow_i b)$ then $(a \rightarrow b)$

HB2 (interprocess) : ∀m send(m) → receive(m)

HB3 (transitive) : If $(a \rightarrow b$ and $b \rightarrow c)$ then $(a \rightarrow c)$

- If they are not related by → (cannot tell who is before)
  then they are said to be **concurrent**, denote by ||
  (if no $a \rightarrow b$ then $a \| b$)

Logical Clock / Lamport Clock

- monotonically incresing software **counter** (นับกระโดดได้)
- use concept of HB relation
- logical timestamp for each process P_i with logical clock L_i (1 process / 1 LC)

LC1 : L_i is incremented by 1 **before** each event at process P_i

LC2 : (a) process P_i send(m) to process P_j ในนั้น $t = L_i$ ไปด้วย

(b) process P_j receive(m, t) ก่อน set $L_j = \max(L_j, t)$, ทำ LC1, แล้วค่อยทำ receive(m)

So $a \rightarrow b$ imply $L(a) < L(b)$

but $L(a) < L(b)$ **not** imply $a \rightarrow b$

- ถ้ามีการส่งจากหลาย process มา P นี้เรียงตาม Logical Timestamp (แม้ว่า L จะมาไม่เรียง) เพราะแต่ละการส่งมี delay ไม่เท่ากัน
  - อาจจะซ้ำกันได้ → ทำ Totally ordered logical clocks
    คือ เรียงตาม Logical Timestamp + process ID (หรือ Host ID + Process ID)

* There is no way to have perfect knowledge on ordering of events

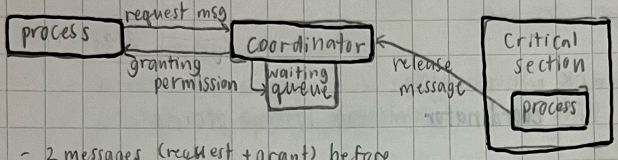
Coordination
- value and action agreement

Mutual exclusion: exclusive access to some resource, critical section

Requirement
- Safety - one process run critical section at a time
- Liveness - Request สุดท้ายจบ (ไม่เกิดการรอกันเป็นวงๆ) No deadlock (process รอซึ่งกันและกัน) / Starvation (process ถูกปฏิเสธไม่ให้ทำงานเรื่อยๆ)
- Fairness - ต้องทำตามลำดับที่ก่อนหน้านี้ จะใช้ก่อนที่ต้องมาก่อน

Solution
- Permission-based - ต้องขออนุญาตจากบาง process
- Token-based - ส่งต่อ token สำหรับ access resource

Centralized Permission-based Algorithm
- One process is elected as the **coordinator** ทำการจัดลำดับ



- 2 messages (request + grant) before
  1 message (release) after
- Disadvantage: delay on enter(), Sync delay, single point of failure and performance bottleneck

Distributed Permission-based Algorithm: Ricart and Agrawala Algorithm
- **send**s message (resource name, process id, current logical time) **to all processes** and waits until all OK
- คนถูกถาม มีทางใช้ 3 กรณี
  - ไม่มีใครจะใช้ → return OK
  - ใช้อยู่แต่ยังไม่เสร็จ → queue the request
  - จะใช้แต่ยังไม่มีใครใช้
    - จะใช้ก่อน → queue the request
    - จะใช้ทีหลัง → return OK
- หลังจากใช้ critical section เสร็จ return OK to every process in queue and clear queue
- Assume no message losts
- Disadvantage: All processes involved in all permission requests, N-point of failure

Token-ring Algorithm
- Logical ring
- ถ้าอยากใช้ → รอจนกว่าจะได้ Token → ใช้เสร็จส่ง Token ต่อ
  ถ้าไม่อยากใช้ → ส่งต่อ
- Disadvantage: ถ้ามี process เยอะ จะเกิด delay มาก

Election Algorithm : choosing a unique process to play a particular role
assume unique ID , select highest ID to be the role

Ring-based Election Algorithm

- notices that current coordinator is down
- send **election** message (this. processID)
  - receives and compare receive. ID , this. ID
  - send max processID until receive. ID = this. ID (วนครบรอบ)
- send **elected** message (elected. processID)
  - receives and set new coordinator

Bully Algorithm

- knows ID of all processes.
- ส่งคำท้าไปหา ID มากกว่า
  - ถ้ายัง running อยู่ให้ return OK แล้วหาคนอื่นต่อ
- ถ้าไม่ได้รับ OK ภายในเวลา T ให้ส่ง **coordinator** message ไปยังทุก process

(5) Fault Tolerance

- partial failure , ไม่ได้กระทบทั้งระบบ
- presence of faults → จัดการได้ = fault tolerance
- Availability, Reliability, Safety, Maintainability

Failure Metrics

- Mean Time To Failure = avg (uptime)
- Mean Time To Repair = avg (downtime)
- Mean Time Between Failure = MTTF + MTTR
- Availability = $\frac{MTTF}{MTBF} = \frac{uptime}{total}$

Failure Model

Crash failure หยุดทำงาน
Timing failure ทำงานเกินเวลา
Omission failure รับไม่ได้
(Receive / send) ส่งไม่ไป
Response failure ทำงานผิด
Arbitary failure ไม่มีเหตุผล

Failure Handling

Detect - CheckSum
Masking - Hide
(Retransmit, Replication, Dropping)
Tolerating - inform users
Recovery - rollback data
Redundancy - 2 replica of servers and routers

k-fault tolerant พังได้ k components แต่ยังทำงานได้

(*) Agreement Problems
- หาข้อตกลงอะไรบางอย่าง เช่น coordinator

Consensus Problem (centralized)
- finding a consensus
- centralized coordinator
- ทุกคนส่งโหวตมา → finding an agreed value (min, max, majority)
  → return decision
- 2k+1 process with k-fault can have majority

Byzantine Generals Problem (กองทัพ-นายพล)
- accept arbitrary failure
- cross check and select majority (commander may send wrong message)
  (any node may have response failure)
- 3k+1 process with k-fault
- Correct processes agree on the same value
- แม้มีพังได้ = ไม่ทำอะไรเลย เหมือนๆ กัน

CAP Theorem

Consistency : some node must reject requests (no most recent writes)
Availability : responses will be inconsistent