Chapter 1
Introduction

Hsung-Pin Chang
Department of Computer Science
National Chung-Hsing University
Why Distributed Systems?

- Previously
  - Centralized systems (or single-processor system)
- However, two technologies advances
  - Development of powerful microprocessors
  - Development of high-speed networks
- Create the Distributed System.
Chapter 1.1: Definition of a Distributed System

A distributed system is *a collection of independent computers that appears to its users as a single coherent system*.

To support heterogeneous computers and networks while offering a single-system view:
- Distributed systems are organized by layers
- Also called a *middleware*
An Abstract View

A distributed system organized as middleware. Note that the middleware layer extends over multiple machines.
Chapter 1.2 Goals

- Goals of a distributed system
  - Easily connect users to resources
  - Hide the fact that resources are distributed
    - Exhibit transparency
  - Support be open
  - Be scalable
Connecting Users and Resources

- For economics, resource would be shared
  - Like printer, files……

- However, how to solve the security problem?
  - Eavesdropping
  - Intrusion on communication
  - ......
Transparency

- Hide the fact the its processes and resources are physically distributed across computers
- But how to achieve “single-system image”, i.e., transparency?
- How to hide distribution?
  - From users?
  - From programs?
- Is it a good idea?
  - Usually trade-off between transparency and performance.
Transparency in a Distributed System

<table>
<thead>
<tr>
<th>Transparency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Hide differences in data representation and how a resource is accessed</td>
</tr>
<tr>
<td>Location</td>
<td>Hide where a resource is located</td>
</tr>
<tr>
<td>Migration</td>
<td>Hide that a resource may move to another location</td>
</tr>
<tr>
<td>Relocation</td>
<td>Hide that a resource may be moved to another location while in use</td>
</tr>
<tr>
<td>Replication</td>
<td>Hide that a resource may be replicated</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Hide that a resource may be shared by several competitive users</td>
</tr>
<tr>
<td>Failure</td>
<td>Hide the failure and recovery of a resource</td>
</tr>
<tr>
<td>Persistence</td>
<td>Hide whether a (software) resource is in memory or on disk</td>
</tr>
</tbody>
</table>

Different forms of transparency in a distributed system.
Transparency (Cont.)

- Access transparency
  - Example:
    - Intel-based computer: little endian
    - Sun SPARC workstation: big endian
  - Example
    - A distributed system may have computer systems with different OS
      - As a result, have different file-naming convention
Transparency (Cont.)

- Location transparency
  - Solution: a well-designed *naming* scheme
    - Resources may be assigned to a *logical names*

- Migration transparency

- Relocation transparency
  - Stronger than migration transparency since resource may be in use
  - Example: mobile users roaming from place to place but can continue to use their wireless network
Transparency (Cont.)

- Concurrency transparency
  - Key point: consistency
  - Solutions: locking or transaction

- Failure transparency
  - Mask failure

- Persistence transparency
  - Example: invoke methods on stored object
    - We must first copies the object from disk to memory, perform operation, and then write back to disk if necessary
    - User is unaware of the server is moving state between disk and memory
Transparency (Cont.)

- Degree of transparency
  - Trade-off between a high degree of transparency and the performance of a system

- Example: a copy of a file is changed
  - If the change should be propagated to all copies on different machines
  - A single update should take seconds to complete
Openness

- An open distributed system offers services according to standard rules that describe the syntax and semantics of those services
  - Example: in computer networks
    - Standard rules govern the format, contents, and meaning of message sent and received
    - Called protocols

- Openness guarantees
  - Interoperability
  - Portability
  - Flexibility and extensibility
Scalability

- Be measured at 3 different dimensions
  - Be scalable in size
    - Easily add more users, machines, and resources
  - Geographically scalable
    - Users and resources lie far apart
  - Administratively scalability
    - It can still be easy to manage even if it spans may independent administrative organizations
Scalability Problems

- Scaling with Respect to Size
- Scaling with Respect to Geographical
- Scaling with Respect to Administration
Scalability Problems - Scaling with Respect to Size

- If more users or resources need to be supported
  - Be confronted with the limitation of centralized services, data, and algorithms
  - See the next slide
Scalability Problems- Scaling with Respect to Size

<table>
<thead>
<tr>
<th>Concept</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized services</td>
<td>A single server for all users</td>
</tr>
<tr>
<td>Centralized data</td>
<td>A single on-line telephone book</td>
</tr>
<tr>
<td>Centralized algorithms</td>
<td>Doing routing based on complete information</td>
</tr>
</tbody>
</table>

Examples of scalability limitations.
Simple Principle

- Only decentralized algorithms should be used
  - Avoid centralized services, tables, and centralized algorithms.

- Characteristics of decentralized algorithms
  - No machine has complete information about the system state.
  - Machines make decisions based only on local information.
  - Failure of one machine doesn’t ruin the algorithm.
  - There is no implicit assumption of a global clock.
Scalability Problems- Scaling with Respect to Geographical

- Problems
  - Latency
    - In LAN, we may use synchronous communications
      - i.e., blocking send/receive
    - However, in WAN, result in hundreds of milliseconds delay
  - Reliability
    - WAN is unreliable compared with that of LAN
Scalability Problems- Scaling with Respect to Administration

- How to scale across multiple, independent administrative domains

- Problems
  - Conflicting policies with respect to resource usage, payment, management, and security
Scaling Techniques

- Hiding communication latencies
  - Asynchronous communication
  - Function shipping to clients
- Distribution of components
  - E.g., DNS lookup
- Caching and Replication
  - Consistency issues
Hiding Communication Costs

- Applicable in the case of geographical scalability

- Idea
  - Try to avoid waiting for responses to remote service request as much as possible
    - Asynchronous communication
  - Move part of the computation at the server to the client to reduce the overall communication latency
    - Example: Java applets that ships code to the client
Hiding Communication Costs

The difference between letting: (a) a server or (b) a client check forms as they are being filled
Information Distribution

An example of dividing the DNS name space into zones.
Replication and Caching

- Caching is a special form of replication

- Drawbacks
  - May adversely affect scalability since consistency problem
Chapter 1.3 Hardware Concepts

- Multiprocessors
- Homogeneous Multicomputer
- Heterogeneous Multicomputer
Hardware Concept

- Two groups
  - Multiprocessor: computers that have shared memory
  - Multicomputer: computers that have no shared memory
    - Homogeneous v.s. heterogeneous

- Each of the categories can be further divided by the architecture of the interconnection network
  - Bus v.s. Switched
Multiprocessor vs. Multicomputer
Chapter 1.3.1: Multiprocessor

- All the CPUs have direct access to shared memory
- 4 or 5 CPU will saturate the bus
  - Solution: high-speed cache memory
  - Problem: coherence problem
Multiprocessor (Cont.)

- Problem of bus-based architecture
  - Scalability, even with caches

- Solutions: Crossbar switch

- Problem of crossbar switch
  - A larger number of crosspoint switch is needed if $n$ is large

- Solutions: Omega Network
  - Problem: several stages between CPU and memory

- Solutions: NUMA (NonUniform Memory Access)
  - Access its own local memory quickly but others slowly
  - Problem: to determine the placement of programs and code is not easy
Crossbar & Omega Switch

(a) 
Cons: # of switches

(b) 
Cons: latency
Homogeneous Multicomputer Systems

- The network traffic is much lower than multiprocessor systems
  - CPU-to-CPU v.s. CPU-to-Memory
- Bus-based multicomputer
  - Example: fast-Ethernet cluster
  - Problems: scalability, 25-100 nodes at most
- Switch-based multicomputer
  - Meshes: two-dimension
  - Hypercubes: $n$-dimension cube
Switched-based Multicomputer

(a)  

(b)  

Four-dimensional cube
Homogeneous Multicomputer Systems (Cont.)

- **MPP: Massively Parallel Processors**
  - A supercomputer consists of thousands of CPU
  - Use proprietary high-performance interconnection network

- **Cluster of workstations (COWs)**
  - A collection of PCs or workstations
  - Myrinet or Gigabit network
Heterogeneous Multicomputer Systems

- Most distributed systems today are built on top of a heterogeneous multicomputer
  - Processor type, memory sizes, I/O bandwidth are different
  - The interconnection network may be highly heterogeneous as well
Chapter 1.4: Software Concept

- Distributed OS (DOS)
- Network OS (NOS)
- Middleware
Software Concept

- Mostly the software determines what a distributed system actually looks like
- Distributed systems are very much like traditional operating systems.
  - A resource managers for the underlying hardware
    - Allowing multiple users and applications to share resources
  - Provide a virtual machine on which applications can be easily executed
    - Hide the intricacies and heterogeneous nature of the underlying hardware.
Operating systems for distributed computers can be roughly divided into two categories:

- **Tightly coupled systems**
  - The OS maintain a single, global view of the resources
  - Generally referred to as a *distributed operating system (DOS)*
    - Used for managing multiprocessors and homogeneous multicomputers.

- **Loosely-coupled systems**
  - A collection of computers each running their own operating system.
    - However, these operating systems work together to make their own services and resources available to the others.
  - The loosely-coupled *network operating system (NOS)* is used for heterogeneous multicomputer systems.
Software Concept (Cont.)

- Middleware
  - Enhancement to the services of the *network operating system* to actually come to a distributed system
# Software Concepts

<table>
<thead>
<tr>
<th>System</th>
<th>Description</th>
<th>Main Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOS</td>
<td><strong>Tightly-coupled</strong> operating system for multi-processors and homogeneous multicomputers</td>
<td>Hide and manage hardware resources</td>
</tr>
<tr>
<td>NOS</td>
<td><strong>Loosely-coupled</strong> operating system for heterogeneous multicomputers (LAN and WAN)</td>
<td>Offer local services to remote clients</td>
</tr>
<tr>
<td>Middleware</td>
<td>Additional layer atop of NOS implementing general-purpose services</td>
<td>Provide distribution transparency</td>
</tr>
</tbody>
</table>
Distributed Operating Systems

- Two types of distributed operating systems.
  - A multiprocessor operating system
    - Manages the resources of a multiprocessor.
  - A multicomputer operating system
    - An operating system that is developed for homogeneous multicomputers
Uniprocessor Operating Systems (Review)

- Protection of resource when they are shared
- Solutions: CPU also least support \textit{kernel mode} and \textit{user mode}
  - All operating system code is executed in kernel mode
  - Application switches between user model and kernel mode (when making a system call)
Multiprocessor Operating Systems

- Multiple processors with shared memory
  - Support for multiple processors having access to a shared memory.

- Problem: consistency

- Solutions: synchronization
  - Semaphore
  - Monitor: programming-language concept since semaphore is error-prone
Multicomputer Operating Systems

- Multicomputer OS is completely different than multiprocessor OS
  - No shared physical memory
  - Communication by *message passing*
- Each node has its kernel
- Above each local kernel is a virtual machine layer
  - Either provide an abstraction of a multiprocessor machine
    - i.e., provide a software implementation of shared memory
  - Or offer only message-passing facilities to applications
    - Two issues: buffering and blocking
General Structure of a Multicomputer Operating System
Blocking and Buffering in Message Passing

- Places for buffering
  - At the sender’s site or at the receiver sites

- Four possible *synchronization points*, i.e., point at which sender or receiver would block
  - If buffer at the sender’s site
    - Block the sender when the buffer is full: S1 at the next slide
  - If there is no sender buffer, a sender would be blocked at
    - The message has been sent (S2)
    - The message has arrived at the receiver (S3)
    - The message has been delivered to the receiver (S4)
  - If there is no receiver buffer or the buffer is empty
    - The receiver can only be blocked at S3
Blocking and Buffering in Message Passing
Synchronization points also inference the communication is reliable or not.

<table>
<thead>
<tr>
<th>Synchronization point</th>
<th>Send buffer</th>
<th>Reliable comm. guaranteed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block sender until buffer has free slot</td>
<td>Yes</td>
<td>Not necessary</td>
</tr>
<tr>
<td>Block sender until message has been sent</td>
<td>No</td>
<td>Not necessary</td>
</tr>
<tr>
<td>Block sender until message has been received</td>
<td>No</td>
<td>Necessary</td>
</tr>
<tr>
<td>Block sender until message has been delivered</td>
<td>No</td>
<td>Necessary</td>
</tr>
</tbody>
</table>

Relation between blocking, buffering, and reliable communications.
Distributed Shared Memory Systems

- Programming multicomputer is much harder than multiprocessor
  - Solution: emulating shared memory on multicomputers, i.e., Distributed Shared Memory (DSM)

- Example: page-based DSM
  - The virtual machine capabilities support a large virtual address space
  - Each page is spread over all the processors
  - If remote (invalid) access → traps → page in → restart instruction
Page-Based DSM

a) Pages of address space distributed among four machines

b) Situation after CPU 1 references page 10: page 10 is moved from CPU2 to CPU1

c) Situation if page 10 is read only and replication is used
Distributed Shared Memory Systems (Cont.)

- Improvements:
  - Replicate read-only pages
  - Replicate all pages, not only read-only pages
    - Problem: prevent inconsistent copies
  - Large page
    - Problem: *false sharing*, data belonging two independent processes in the same page

- After 15 year of research
  - DSM cannot deliver high-performance on large-scale multicomputer
False Sharing on DSM

Machine A

Code using A

Page p

A

B

Page transfer when B needs to be accessed

Machine B

Code using B

Page p

A

B

Page transfer when A needs to be accessed

Two independent data items
Network Operating Systems

- NOS does not assume underlying hardware is homogeneous and be managed as a single system
  - Different from distributed OS

- Difference between NOS and DOS
  - DOS attempts to realize full transparency and provide a single-system view
Network Operating Systems

The machine and the OS can be different
Network Operating Systems Examples

- **Examples**
  - `rlogin machine`
  - `Rcp machine1:file1 machine2:file2`
  - *File servers*

![Diagram showing network setup with client devices requesting files from a file server.]
Network Operating System (3)

Different clients may mount the servers in different places – difficult to maintain a consistent “view” of the system.
Network Operating Systems Pros and Cons

- Disadvantages
  - Harder to use
  - Management problem
    - Each machine is managed independently
    - A user needs to have an independent account on each machine

- Advantages
  - Easy to add or remove a machine
Middleware

- DOS—doesn’t handle a collection of independent computers
- NOS—doesn’t provide a single coherent system
- Modern DS use a component that provides us with
  - The scalability and openness of NOS
  - The transparency and ease of use of DOS
  ➔ Middleware: add an additional layer on top of NOS
The Best of Both Worlds?

DOS: too inflexible (all systems of the same type)

NOS: too primitive (lowest common denominator – too much diversity)

“Middleware” – best possible compromise?

Middleware = additional software layer on the NOS
Positioning Middleware

- Goal: Transparency for applications.

- If applications use socket operations provided by the network operating systems
  - The distributed system is not transparent.
  - Middleware provides a higher level abstraction to applications.

- Sit between the NOS and applications
Positioning Middleware

General structure of a distributed system as middleware.
Middleware Models

- Middleware is based on some models, or paradigms, for communication

- Example:
  - Treating everything as a file, like Plan 9
    - All resources, such as keyboard, mouse, disk...are treated as files
  - Distributed File Systems
    - Less strict than Plan 9
  - Remote Procedure Call (RPC)
  - Distributed Objects
  - Distributed Documents: WWW
Middleware Services

- Services
  - Communication facilities to implement *access transparency*
    - Depends on the model assumed, like WWW
  - Naming
  - Persistent storage
  - Distributed transactions
  - Security
Middleware and Openness

In an *open* middleware-based distributed system, the *protocols* used by each middleware layer should be the same, as well as the *interfaces* they offer to applications.
# Comparing DOS/NOS/Middleware

<table>
<thead>
<tr>
<th>Item</th>
<th>Distributed OS</th>
<th>Network OS</th>
<th>Middleware-based DS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Multproc.</td>
<td>Multicomp.</td>
<td></td>
</tr>
<tr>
<td>Degree of transparency</td>
<td>Very High</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Same OS on all nodes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Number of copies of OS</td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Basis for communication</td>
<td>Shared memory</td>
<td>Messages</td>
<td>Files (or high-level message)</td>
</tr>
<tr>
<td>Resource management</td>
<td>Global, central</td>
<td>Global, distributed</td>
<td>Per node</td>
</tr>
<tr>
<td>Scalability</td>
<td>No</td>
<td>Moderately</td>
<td>Yes</td>
</tr>
<tr>
<td>Openness</td>
<td>Closed</td>
<td>Closed</td>
<td>Open</td>
</tr>
</tbody>
</table>
Chapter 1.5: The Client-Server Model

- Structure a distributed program as a process (server) that offers services to clients.

- Outline
  - Clients and Servers
  - Application Layering
  - Client-Server Architecture
Clients and Servers

- The application is modelled as a set of services that are provided by servers and a set of clients that use these services
  - Clients know of servers but servers need not know of clients
    - Server name/IP address and Port number
  - Clients and servers are logical processes
    - Can be in the same machine
  - The mapping of processors to processes is not necessarily 1 : 1
Clients and Servers (Cont.)

Connection protocol: connectionless or connection-oriented?
- Connectionless: efficient but unreliable
  - Cause problem when the message is “transfer &10000 from my bank account”
- Connection-oriented: inefficient but reliable
Application Layering

- Question: What goes into client? Server?
- A client-server application may consist of three parts
  - First part: handle interaction with user
    - User-interface level
  - Middle part: core functionality of the application
    - Processing level
  - End part: operate on database of file system
    - Data level
Thus, many applications can be viewed as having three processing levels.

- **User-interface level** (usually client)
  - Character-based or GUI-based
- **Processing level**
- **Data level** (usually server)
  - Relational databases or OO databases

**Example: Web Search Engine**

- Interface = enter a key word
- Processing = turn keyword into database query, execute
- Data = the database of web pages.
Application Layering

The general organization of an Internet search engine into three different layers.
Client-Server Architectures

- Possible architecture:
  - A client machine contains only programs implementing user-interface level.
  - A server machine contains the rest, i.e., the processing and data level.

- However, not really distributed.

- Other possible architectures:
  - Multitiered architecture
  - Modern architecture
Multitiered Architecture

- Two-tiered architecture

(d). Only operations on files or database go to the server
(e). The client’s local disk contains part of the data
Multitiered Architecture (Cont.)

- Three-tiered architecture
- Example: a new process called *transaction monitor*
  - Coordinates all transactions in transaction processing

An example of a server acting as a client
Modern Architecture

- Vertical distribution
  - Multitiered client-server architecture

- Horizontal distributions: modern architecture
  - A client or a server may be physically split up into logically equivalent parts
  - Each part is operating on its own share of the complete data set to balance the load

- Peer-to-peer distribution
  - No clients and servers. All processes play a similar role.
  - May act as clients and servers to each other.
Horizontal Distribution

An example of horizontal distribution of a Web service