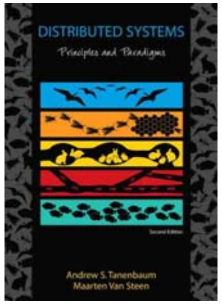
# Introduction to Distributed Systems

527950-1
Fall 2019
9/19/2019
Kyoung Shin Park
Applied Computer Engineering
Dankook University

## Chapter 1. Introduction




From Andrew S Tanenbaum, Maarten Van Steen
Distributed Systems: Principles and Paradigms
Edition 2, © Prentice Hall 2007

## Overview

- Definition of a Distributed System
- Goals of a Distributed System
  - Making Resources Accessible
  - Transparency
  - Openness
  - Scalability
  - Pitfalls
- Three Types of Distributed Systems
  - Distributed computing systems
    - Cluster computing, Grid computing, Cloud computing
  - Distributed information systems
    - Transaction processing system
  - Distributed systems for pervasive computing
    - Ubiquitous computing, Mobile computing, sensor network

## Definition of a Distributed System

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

- Differences between the various computers and the ways in which they communicate are mostly hidden from users.
- Users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.

# Definition of a Distributed System

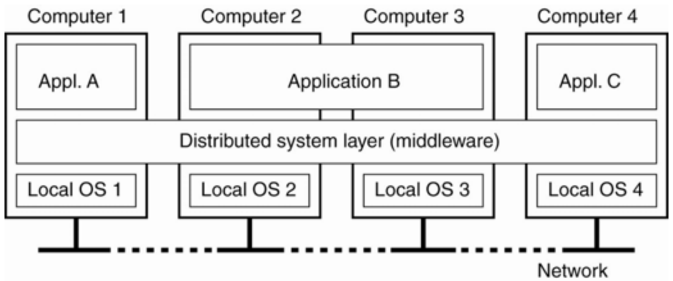



Figure 1-1. A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

# Goals of a Distributed System

- Goals of a distributed system (Tanenbaum & van Steen)
  - Making Resources Accessible
  - Transparency
  - Openness
  - Scalability

# Making Resource Accessible

- The main goal of a distributed system is to make it easy for the users (and applications) to **access** remote resources, and to share them in a controlled and efficient way.
- Resources can be things like printers, computers, storage facilities, data, files, Web pages, and networks, etc.

# Making Resource Accessible

- Main goal of a distributed system is to make it easy for the users (and applications) to **access** remote resources, and to share them in a controlled and efficient way.
  - Printers, computers, storage facilities, data, files, web pages, etc.
    - It is cheaper to let a printer be shared by several users in a small office than having to buy and maintain a separate printer for each user.
    - It makes economic sense to share costly resources such as supercomputers, high-performance storage systems, image setters, and other expensive peripherals.
- Connecting users and resources also makes it easier to collaborate and exchange information.
  - Exchanging files, mail. documents, audio, and video
  - Geographically widely-dispersed groups of people work together
  - Electronic commerce
- As connectivity and sharing increase, **security** is becoming increasingly important.

## Transparency

| Transparency | Description |
|---|---|
| Access | Hide differences in data representation and how a resource is accessed |
| Location | Hide where a resource is located |
| Migration | Hide that a resource may move to another location |
| Relocation | Hide that a resource may be moved to another location while in use |
| Replication | Hide that a resource is replicated |
| Concurrency | Hide that a resource may be shared by several ~~competitive~~ independent users |
| Failure | Hide the failure and recovery of a resource |

Figure 1-2. Different forms of transparency in a distributed system (ISO, 1995).

## Openness

- An **open** distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services.
- In distributed systems, services are generally specified through interfaces, which are often described in an **Interface Definition Language (IDL)**.
  - Interfaces specify precisely the names of the functions that are available together with types of the parameters, return values, possible exceptions that can be raised, and so on.
- An open distributed system should also be **extensible**.
  - It should be easy to add new components or replace existing ones without affecting those components that stay in place.

## Scalability

- If more users or resources need to be supported,
  - The server can become a bottleneck and will eventually prohibit further growth
  - A single database would saturate all the communication lines into and out of it
  - An enormous number of messages have to be routed. Collecting and transporting all the information would be a bad idea because message would overload part of the network.

| Concept | Example |
|---|---|
| Centralized services | A single server for all users |
| Centralized data | A single on-line telephone book |
| Centralized algorithms | Doing routing based on complete information |

Figure 1-3. Examples of scalability limitations.

## Scalability

- 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 does not ruin the algorithm.
  - There is no implicit assumption that a global clock exists.

## Scalability

- At least three components
  - Number of users and/or processes(**size scalability**)
  - Maximum distance between nodes(**geographical scalability**)
  - Number of administrative domains(**administrative scalability**)

Observation

Most systems account only, to a certain extent, for size scalability. Often solution : multiple powerful servers operating independently in parallel. Today, the challenge still lies in geographical and administrative scalability.

## Pitfalls when Developing Distributed Systems

- False assumptions made by first time developer:
  - The network is reliable.
  - The network is secure.
  - The network is homogeneous.
  - The topology does not change.
  - Latency is zero.
  - Bandwidth is infinite.
  - Transport cost is zero.
  - There is one administrator.

## Three Types of Distributed Systems

- **High performance distributed computing systems**
  - Parallel computing, Distributed shared memory systems
  - Cluster computing, Grid computing, Cloud computing
- **Distributed information systems**
  - Transaction Processing System
- **Distributed systems for pervasive computing**
  - Ubiquitous Systems
  - Mobile Computing

## Cluster Computing Systems

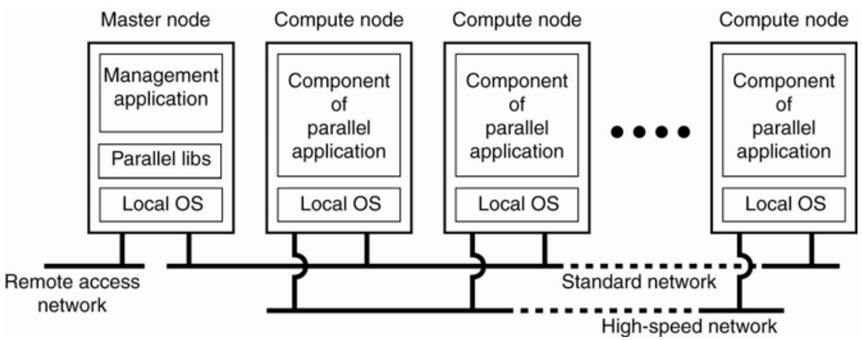



Figure 1-6. An example of a cluster computing system.

## Cluster Computing Systems

- Characteristic feature of Cluster Computing : homogeneity
  - Same or similar Computers, same OS, same network
- The underlying hardware consists of a collection of similar workstations or PCs, closely connected by means of a high-speed LAN. Each node runs the same OS.

## Grid Computing Systems

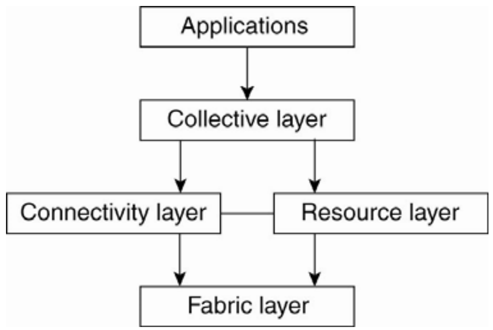



Figure 1-6. A layered architecture for grid

## Cluster Computing Systems

- Characteristic feature of Grid Computing : heterogeneity
  - No assumptions are made concerning hardware, OSs, networks, administrative domains, policies, etc

## Transaction Processing System

- Database applications
  - Operations on a database are usually carried out in the form of transactions.
  - BEGIN_TRANSACTION and END_TRANSACTION are used to delimit the scope of a transaction.
  - The operations between them form the body of the transactions.
  - The characteristic feature of a transaction is either all of these operations are executed or none are executed

| Primitive | Description |
|---|---|
| BEGIN_TRANSACTION | Mark the start of a transaction |
| END_TRANSACTION | Terminate the transaction and try to commit |
| ABORT_TRANSACTION | Kill the transaction and restore the old values |
| READ | Read data from a file, a table, or otherwise |
| WRITE | Write data to a file, a table, or otherwise |

Figure 1-8. Example primitives for transactions.

## Transaction Processing System

- Characteristic properties of transactions(ACID):
  - Atomic: To the outside world, the transaction happens indivisibly.
    - Each transaction either happens completely or not at all
    - While a transaction is in progress, other processes cannot see any of the intermediate states
  - Consistent: The transaction does not violate system invariants.
  - Isolated: Concurrent transactions do not interfere with each other.
    - Transactions are isolated or serializable.
    - If two ore more transactions are running at the same time, the final result looks at though all transactions ran sequentially in some (system dependent) order
  - Durable: Once a transaction commits, the changes are permanent.
    - No failure after the commit can undo the results or cause them to be lost.

## Transaction Processing System

- A Nested Transaction
  - The top-level transaction may fork off children that run in parallel with one another, on different machines, to gain performance or simplify programming.
  - Can be nested arbitrarily deeply

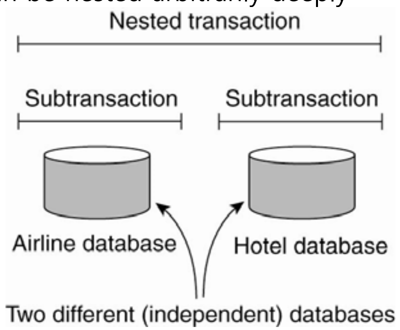



Figure 1-9. A nested transaction.

## Transaction Processing System

- Transaction Processing(TP) Monitor in enterprise middleware system
  - Allows an application to access multiple server/databases by offering it a transactional programming model

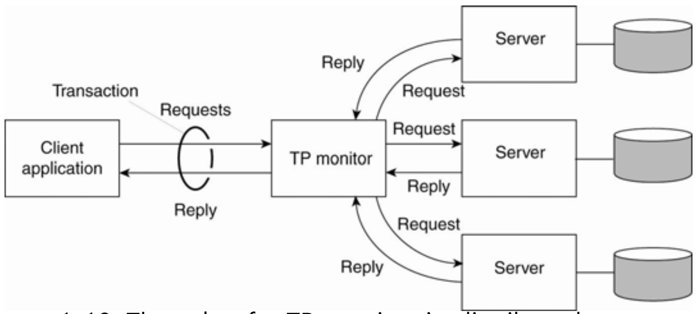



Figure 1-10. The role of a TP monitor in distributed systems.

## Enterprise Application Integration

- Communication Middleware
  - Applications are decoupled(independent) from the databases
  - Applications components should be able to communicate directly with each other
  - This need for inter-application communication lead to many different communication models
    - Remote Procedure Call(RPC) – operates on applications
    - Remote Method Invocation(RMI) – operates on objects

# Enterprise Application Integration

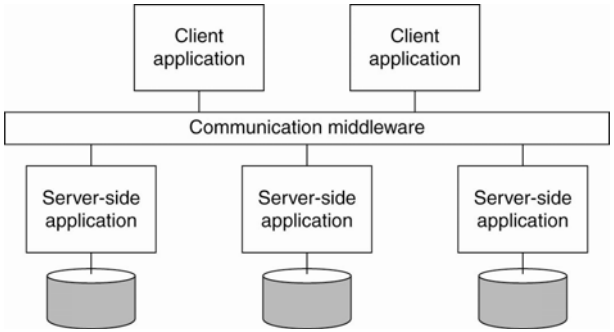



Figure 1-11. Middleware as a communication facilitator in enterprise application integration.

# Distributed Pervasive Systems

- **Three (overlapping) subtypes**
  - **Ubiquitous computing systems** : pervasive and continuously present, i.e., there is a continuous interaction between system and user.
  - **Mobile computing systems** : pervasive, but emphasis is on the fact that devices are inherently mobile.
  - **Sensor (and actuator) networks** : pervasive, with emphasis on the actual (collaborative) sensing and actuation of the environment.

# Distributed Pervasive Systems

- Introducing mobile and embedded computing devices
- **Requirements for pervasive systems**
  - Embrace contextual changes.
    - A device must be continuously be aware of the fact that its environment may change all the time
  - Encourage ad hoc composition.
    - Should be easy to configure the suite of applications running on a device
  - Recognize sharing as the default.
    - Should be easily read, store, manage, and share information

# Ubiquitous Computing Systems

- **Core Elements for Ubiquitous Systems**
  - (**Distribution**) Devices are networked, distributed and accessible in a transparent manner
  - (**Interaction**) Interaction between users and devices is highly unobtrusive
  - (**Context awareness**) The system is aware of a user's context in order to optimize interaction
  - (**Autonomy**) Devices operate autonomously without human intervention, and are thus highly self-managed
  - (**Intelligence**) The system as a whole can handle a wide range of dynamic actions and interactions