### Why Distributed Systems?

- **Sometimes, it is natural.** Think of geographically distributed applications: ATMs, servers, battlefields.
- **Speed-up.** Example: grid computing.
- **Resource sharing.** Distributed database, peer-to-peer networks, aggregate computing power (e.g., SETI@HOME).
- **Fault-tolerance.** Trade-off between total collapse and graceful degradation.

### Characteristics

- Concurrency
- Lack of global clock
- Lack of global knowledge
- Independent failures

### Theory vs. practice

- The community is divided into two camps, although they are expected to complement each other’s efforts.
  - Web, Freenet, Java, RPC, RMI, CORBA, DCOM, GRID, Kerberos, PGP

```markdown
- Models, proofs, algorithms, mutual exclusion, group communication, replication, transaction deadlock, consensus
```

### Parallel Computers/Systems

- A collection of processors that cooperate to solve large problems fast.
  - from a pile of PCs to a shared memory multiprocessor
  - Interconnection Network
  - Where is the memory physically located?
Distributed Systems vs. Parallel Systems

- Loosely coupled vs. tightly coupled
- Asynchrony vs. Synchrony
- Course-grained parallelism vs. fine-grained parallelism
- Concurrency vs. Parallelism

Heterogeneity

Source of heterogeneity:
- Networks
- Computer hardware
- Operating systems
- Programming languages
- Implementations by different developers

Possible solutions:
- Middleware (e.g., CORBA, DCOM, RPC, Java RMI)
- Virtual machine (e.g., Java byte code)

Openness

A system can be open or closed w.r.t.
- Hardware extensions (e.g., the addition of peripherals, memory, or communication interfaces)
- Software extensions (e.g., the addition of operating system features, communication protocols, and resource-sharing services).

Internet-related documents and specifications are published through “Requests For Comments” (RFC), each of which is identified by a number.

Openness enables systems to be extended in various ways.
Openness is achieved by specifying and documenting the key software interfaces of a system and making them available to developers.

Security

Distributed systems are vulnerable to security threats due to their openness. Security needs to guarantee:
- Confidentiality
- Integrity
- Availability
- Authentication
- Nonrepudiation
- Access control

Recent issues:
- Denial of service (DOS) attacks
- Security of mobile code

Scalability

A system is scalable if it will remain effective when there is a significant increase in the number of resources and the number of users.

Examples: Growth of the Internet and Web servers:

<table>
<thead>
<tr>
<th>Date</th>
<th>Computers</th>
<th>Web servers</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993, Jul</td>
<td>1,776,000</td>
<td>130</td>
<td>0.008</td>
</tr>
<tr>
<td>1995, Jul</td>
<td>6,642,000</td>
<td>23,500</td>
<td>0.4</td>
</tr>
<tr>
<td>1997, Jul</td>
<td>19,560,000</td>
<td>1,203,096</td>
<td>6</td>
</tr>
<tr>
<td>1999, Jul</td>
<td>56,248,000</td>
<td>6,588,677</td>
<td>12</td>
</tr>
</tbody>
</table>

By Jul. 2004, the Internet reached the following milestones:
- 353,284,187 IP Hosts
Distributed Information System Introduction

Growth of Internet Hosts

- Challenges:
  - Controlling the cost of physical resources
  - e.g., Adding $n$ more resources, the system should be able to handle $n$ times more requests.
  - Controlling the performance loss
  - e.g., Hierarchical domain name system
  - Preventing software resource running out
  - e.g., 32-bit IP addresses
  - Avoiding performance bottlenecks
  - In general, caching is very useful in dealing with scalability

Scalability (Cont.)

- Peer process model
- E.g., Napster, Freenet, GNU/zilla

Failure Handling

- Detecting failures
  - Checksums, timeout
- Masking failures
  - Handling failures in different levels
- Tolerating failures
- Recovering from failures (“rolled back”)
- Fault tolerance is usually achieved by redundancy, which also increases availability.
  - Replication

Architectural Models

- Client-server model
  - E.g., WWW, most transaction systems
- Peer process model
  - E.g., Napster, Freenet, GNU/zilla

Transparency (Cont.)

- Replication transparency: enables multiple instances of objects to be used to increase reliability and performance without knowledge of the replicas by the users or applications.
- Failure transparency: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of components.
- Mobility transparency: allows the movement of objects within a system without affecting the operation of users or applications.
- Performance transparency: allows the system to be reconfigured to improve performance as loads vary.
- Scaling transparency: allows the systems and applications to extend without change to the system structure or the application algorithms.

Transparency

- Transparency conceals the users and the applications programs from the separation of components in a distributed system.
  - In a nutshell, to make the system look like there is only one component, and only one user is in it.
- ISO identifies 8 forms of transparency:
  - Access transparency: enables local and remote information objects to be accessed using identical operations.
  - Location transparency: enables local and remote information objects to be accessed without knowledge of their location.
  - Concurrency transparency: enables several processes to operate concurrently using shared information objects without interference between them.

Architectural Models

- Client-server model
  - E.g., WWW, most transaction systems
- Peer process model
  - E.g., Napster, Freenet, GNU/zilla

Transparency

- Transparency conceals the users and the applications programs from the separation of components in a distributed system.
  - In a nutshell, to make the system look like there is only one component, and only one user is in it.
- ISO identifies 8 forms of transparency:
  - Access transparency: enables local and remote information objects to be accessed using identical operations.
  - Location transparency: enables local and remote information objects to be accessed without knowledge of their location.
  - Concurrency transparency: enables several processes to operate concurrently using shared information objects without interference between them.

Architectural Models

- Client-server model
  - E.g., WWW, most transaction systems
- Peer process model
  - E.g., Napster, Freenet, GNU/zilla

Transparency

- Transparency conceals the users and the applications programs from the separation of components in a distributed system.
  - In a nutshell, to make the system look like there is only one component, and only one user is in it.
- ISO identifies 8 forms of transparency:
  - Access transparency: enables local and remote information objects to be accessed using identical operations.
  - Location transparency: enables local and remote information objects to be accessed without knowledge of their location.
  - Concurrency transparency: enables several processes to operate concurrently using shared information objects without interference between them.

Architectural Models

- Client-server model
  - E.g., WWW, most transaction systems
- Peer process model
  - E.g., Napster, Freenet, GNU/zilla

Transparency

- Transparency conceals the users and the applications programs from the separation of components in a distributed system.
  - In a nutshell, to make the system look like there is only one component, and only one user is in it.
- ISO identifies 8 forms of transparency:
  - Access transparency: enables local and remote information objects to be accessed using identical operations.
  - Location transparency: enables local and remote information objects to be accessed without knowledge of their location.
  - Concurrency transparency: enables several processes to operate concurrently using shared information objects without interference between them.

Architectural Models

- Client-server model
  - E.g., WWW, most transaction systems
- Peer process model
  - E.g., Napster, Freenet, GNU/zilla
Partition of data or replication of the servers
Caching of data by proxy servers and clients
Mobil code and mobil agents
Multi-Tier Client/Server Architecture
Thin clients

Asynchronous Systems: make no assumptions about process execution speeds and/or message delivery delays.
Synchronous Systems: Timeouts and other time-based protocol techniques are possible only when a system is synchronous.
- time to execute each step of a process has known lower and upper bound.
- message transmission time is bounded.
- local clock whose drift rate from real time is bounded.

Synchronous Systems:
- More general
Synchronous Systems:
- System’s performance can be deduced from time.
- Algorithms may be easier to design.
- However, synchrony may cause performance degradation.
- A problem that cannot be solved in synchronous systems cannot be solved in asynchronous system, either.

Election Problem
A set of processes $P_1, P_2, ..., P_n$ must select a leader. Each process $P_i$ has a unique identifier $uid(i)$. Devise a protocol so that all of the processes learn the identity of the leader. Assume all processes start executing at the same time, and that all communicate using reliable broadcasts.

Election Problem (cont.)
Asynchronous systems: need $n$ broadcasts.
Synchronous systems: one broadcast is enough!

Each process wait until either (1) it receives a broadcast, or
(2) $t * \text{uid}(i)$ seconds elapse at which time it broadcast $\text{uid}(i)$.
The first process that makes broadcast is elected.