Basics

Topics to be covered
Definitions and Examples
Goals
Models (architectural, fundamental)
Hardware and Software Concepts
The Client-Server Model

Historical
Two developments from mid 50s
• Powerful microprocessors
  100 million dollars -- 1 instr per sec
  1000 dollars -- 10 million instr per sec
  $10^12$ price/performance gain
  Rolls Royce cost 1 dollar -- a billion miles per gallon
  (200-page manual to open the door)
• Local and Wide Area networks (LANs and WANs)

Definition of a Distributed System
A distributed system is:
- a collection of independent computers that appears to its users as a single coherent system

Characteristics
(1) Heterogeneity hidden
(2) Interact with a consistent and uniform way
(3) Continuous availability
(4) Scale

Issues
(1) Concurrency
(2) No global clock
(3) Independent failures

A Distributed System as Middleware
Note that the middleware layer extends over multiple machines.
Examples of Distributed Systems

The Internet
Intranets
Mobile and Ubiquitous Computing
The Web
p2p systems (such as Napster)
File systems (SUN, CODA, Adrews)
Storage Systems (Ocean)
Object-based Systems (CORBA, DCOM, etc)
Groupware

A typical portion of the Internet

• ISP: Internet Service Providers
• Backbone links the Intranets together

A Typical Intranet

- A portion of the Internet separately administrated
- Several LANs linked by backbone connections
- Connected to the Internet via a router
- Firewalls protect an intranet by preventing unauthorized messages leaving or entering; implementing by filtered messages

Portable and handheld devices in a distributed system

- Devices: laptop computers, handheld devices (e.g., PDAs, video cameras), wearable devices, devices embedded in appliances
- Mobile computing, ubiquitous computing, location-aware computing
- In the figure above: 3 different forms of wireless connections: wireless LAN, mobile phone through WAP, infrared link

Resource sharing on the Web

- WWW: a system for publishing and accessing resources and services across the Internet
- Web browsers act as clients
- Request resources (e.g., web pages) from web servers
- CERN, 1989
- Hypertext structure among documents

Resource sharing on the Web

- **HTML**: a language for specifying the content and layout of pages to be displayed by web browsers
- **URL**: resource locators - goal identify a resource to enable the browser to locate it
  scheme:scheme-specific-location
  scheme: ftp, http
  http://servername[:port]/pathnameofServer/?argument
  Optional number of the port on which the server listens for requests
  Optional path name of the server's resource
  Set of arguments in the case of a program e.g., search/keyword
  Servername is a DNS name
- **HTTP**: simple protocol specifying the ways in which browsers interact with web servers
Computers in the Internet

<table>
<thead>
<tr>
<th>Date</th>
<th>Computers</th>
<th>Web servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1979, Dec.</td>
<td>188</td>
<td>0</td>
</tr>
<tr>
<td>1989, July</td>
<td>130,003</td>
<td>0</td>
</tr>
<tr>
<td>1999, July</td>
<td>56,218,000</td>
<td>5,560,866</td>
</tr>
</tbody>
</table>

Computers vs. Web servers in the Internet

<table>
<thead>
<tr>
<th>Date</th>
<th>Computers</th>
<th>Web servers</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993, July</td>
<td>1,776,000</td>
<td>130</td>
<td>0.008</td>
</tr>
<tr>
<td>1995, July</td>
<td>6,642,000</td>
<td>23,500</td>
<td>0.4</td>
</tr>
<tr>
<td>1997, July</td>
<td>19,540,000</td>
<td>1,203,056</td>
<td>6</td>
</tr>
<tr>
<td>1999, July</td>
<td>56,218,000</td>
<td>6,598,676</td>
<td>12</td>
</tr>
</tbody>
</table>

Goals

1. Connecting Users and Resources
2. Transparency
3. Openness
4. Scalability

Connecting Users and Resources

Typical resources:
- Printers, computers, storage facilities, data, files

Why sharing?
- Economics
- Collaboration, Information Exchange (groupware)

Problems with sharing:
- Security
- Unwanted Communication

Transparency in a Distributed System

Transparent distributed system:
Looks to its users as if it were only a single computer system

access transparency
Hide differences in data representation and how a resource is accessed

Intel (little endian format)/Sun SPARC (big endian) (order of bytes)
OS with different file name conversions
Transparency in a Distributed System

location transparency
Hide where a resource is located
importance of naming, e.g., URLs

migration transparency
Hide that a resource may move to another location

relocation transparency
Hide that a resource may move to another location while in use
example, mobile users

replication transparency
Hide that a resource is replicated
subsumes that all replicas have the same name (and thus location transparency)

concurrency transparency
Hide that a resource may be shared by several competitive users
leave the resource in a consistent state
more refined mechanism: transactions

failure transparency
Hide the failure and recovery of a resource

persistent transparency
Hide whether a (software) resource is in memory or disk

L. Lamport: You know you have a [distributed system] when the crash of a computer you’ve never heard of stops you for getting any work done

Important problem: inability to distinguish between a dead resource and a painfully slow one

Different Forms of Transparency in a Distributed System (summary)

<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 move to another location while in use</td>
</tr>
<tr>
<td>Replication</td>
<td>Hide that a resource may be shared by several competitive users</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>

Access transparency: enables local and remote resources to be accessed using identical operations. (same)

Location transparency: enables resources to be accessed without knowledge of their location. (same) - also migration and relocation

Mobility transparency: allows the movement of resources and clients within a system without affecting the operation of users or programs.

Replication transparency: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

Concurrency transparency: enables several processes to operate concurrently using shared resources without interference between them.

Failure transparency: enables the concealment of faults, allowing users and application programs to continue to operate despite the failure of hardware or software components - also persistent transparency

Performance transparency: allows the system to be reconfigured to improve performance as loads vary.

Scaling transparency: allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Degree of Transparency

Not always desirable
Examples?
Users located in different continents (context-aware)

Not always possible
Examples?
Hiding failures (you can distinguish a slow computer from a failing one/whether an action was performed before a crash)

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

Keep web caches exactly up-to-date
Immediately flushing write operations to disk
Retry to access a web page to mask a failure
Goals

1. Connecting Users and Resources
2. Transparency
3. Openness
4. Scalability

Openness

Open distributed system
Be able to interact with services from other open systems, irrespectively of the underlying environment
Offers services according to standard rules that describe the syntax and the semantics of these services
- Rules formalized in protocols
- Services specified through interfaces (described in an Interface Definition Language (IDL) (but only the syntax part)
- Neutral and complete specifications (with regards to a potential implementation)

Openness

- Interoperability: to what extend can work together
- Portability: to what extend an application developed for A can be executed on B that implements the same interface with A

Openness

- A system organized as a collection of relatively small and easily replaceable or adaptable components
- Provide definitions of interfaces to internal parts of the system as well
- Separate Policy from Mechanism
A distributed system provides only mechanisms
Policies specified by applications and users
Example policies:
- What level of consistency do we require for client-cached data?
- Which operations do we allow downloaded code to perform?
- Which QoS requirements do we adjust in the face of varying bandwidth?
- What level of secrecy do we require for communication?

Scalability

Along three different dimensions:
- size (number of users and/or resources processes)
- geographical (maximum distance between nodes)
- administrative (number of administrative domains)

The (non) solution: powerful servers

Scalability Problems (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>

Decentralized algorithms
- No complete information about the system state
- Make decision only on local information
- Failure of one machine does not ruin the algorithm
- No assumption of a global clock
Scalability (geographical)

Geographical scalability:
- Synchronous communication
  In WAN, unreliable and point-to-point

Related to centralized solutions

---

Scalability (administrative)

How to scale a distributed system across multiple, independent administrative domains: conflicting policies with respect to resource usage (and payment), management and security

Expand to a new domain
- Protect itself against malicious attacks from the new domain
- The new domain has to protect itself against malicious attacks from the distributed system

---

Scaling Techniques

Three techniques:
- hiding communication latencies
- distribution
- replication

---

Scaling Techniques

Hiding communication latencies

try to avoid waiting for responses to remote service requests as much as possible
- asynchronous communication (do something else)
- moving part of the computation to the client process
  code shipping in the form of java applets

---

Scaling Techniques

Distribution
Taking a component, splitting into smaller parts, and spreading these parts across the system

Example:
(1) The World Wide Web
(2) Domain Name Service (DNS)
  hierarchically organized into a tree of domains
  Each domain divided into non overlapping zones
  The names in each domain handled by a single name server
Scaling Techniques

Replication
- increase availability
- balance the load
- reduce communication latency
- but, consistency problems

Caching (client-driven)

An example of dividing the DNS name space into zones.

Models

System Models
1. Architectural models
2. Fundamental models

An architectural model of a distributed system is concerned with the placement of its parts and the relationships between them.

Examples include the client-server model and the p2p model:
- determine the distribution of data and computational tasks amongst the physical nodes of the system
- useful when evaluating the performance, reliability, scalability and other properties of distributed systems

Architectural Models

Software Layers: Structuring of software as layers or modules

Applications, Service
Middleware
Operating Systems
Computer and Network Hardware

The end-to-end argument
Some communication activities can be completely and reliably implemented only with the knowledge of applications standing at the end points of the communication system

Architectural Models

System architectures: division of responsibilities between system components and the placement of components on computing nodes in the network

Client-server model and its variations
Fundamental Models

Fundamental models are concerned with a more formal description of the properties that are common in all of the architectural models.

Models:

- **Interaction model**: deals with performance and with the difficulty of setting time limits in distributed systems, for example for message delivery.
- **Failure model**: gives a precise specification of the faults that can be exhibited by processes and communication channels. Defines reliable communication and correct processes.
- **Security model**: discusses the possible threats to processes and communication channels.

Interaction Model

- Distributed systems are composed of multiple interacting processes.
- Their behavior and state can be described by a distributed algorithm: a definition of the steps to be taken by each process including the transmission of messages between them.
- Messages are transmitted between processes to transfer information among them and to coordinate their activity.

Communication performance characteristics:

- **Latency**: delay between sending a message by one process and its receipt by another.
- **Bandwidth**: total amount of information that can be transmitted over it in a given time.
- **Jitter**: the variation in the time taken to deliver a series of messages.

Computer clocks:

- **Clock drift rate**: relative amount that a computer clock differs from a perfect reference clock.

Variants of the Interaction Model

- **Synchronous distributed systems**: can set timeouts, can be built.
- **Asynchronous distributed systems**: (e.g., Internet, web).

Despite the lack of accurate clocks, the execution of a system can be described in terms of events and their ordering.

Failure Model

Classification of failures:

- **Omission failures**: when a process or communication channel fails to perform actions that is supposed to do.

- **Process omission failure**: crash.

- **Failure**: crash if other processes can detect certainly that the process has crashed.

Communication errors:

- **Send-omission**: loss of messages between the sending process and the outgoing message buffer.
- **Receive-omission**: loss of messages between the incoming buffer and the receiving process.
- **Channel omission**: loss of messages between communication channels.
Failure Model

Classification of failures:

Arbitrary or Byzantine failures

Arbitrary failures of processes and channels

Omission and arbitrary failures

<table>
<thead>
<tr>
<th>Class of Failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail-stop</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may detect this state.</td>
</tr>
<tr>
<td>Crash</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may not be able to detect this state.</td>
</tr>
<tr>
<td>Omission</td>
<td>Channel</td>
<td>A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.</td>
</tr>
<tr>
<td>Send-omission</td>
<td>Process</td>
<td>A process completes a send, but the message is not put in its outgoing message buffer.</td>
</tr>
<tr>
<td>Receive-omission</td>
<td>Process</td>
<td>A message is put in a process's incoming message buffer, but the process does not receive it.</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>Process or Channel</td>
<td>Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions, a process may stop or take an incorrect step.</td>
</tr>
</tbody>
</table>

Timing failures

In synchronous systems:

<table>
<thead>
<tr>
<th>Class of Failure Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Process's local clock exceeds the bounds on its rate of drift from real time.</td>
</tr>
<tr>
<td>Performance</td>
<td>Process exceeds the bounds on the interval between two steps.</td>
</tr>
<tr>
<td>Performance</td>
<td>Channel</td>
</tr>
</tbody>
</table>

Security Model

To model security threats, we postulate an enemy (or adversary)

Send any message to any process and reading/copying any message between a pair of processes

1. Threats to processes (cannot identify the identity of the sender: holds for both clients and servers)
2. Threats to communication channels
3. Other possible threats (mobile code, denial of service)

Security Model

Securing the processes and the channel and protecting the objects

Protecting the objects: access rights (specify who is allowed to perform each operation of an object)

Associate with each invocation and each result the authority on which it is issued called a principal

Hardware Concepts
Classification of Multiple CPU Computer Systems

Into two groups:

- **Multiprocessors** (shared memory): there is single physical address shared by all CPUs
- **Multicomputers**: each machine has its own private memory. Either Homogeneous or Heterogeneous

Further divided based on the architecture of the interconnection network:

- **Bus**: a single network that connects all machines
- **Switch**

Overload the bus \(\rightarrow\) cache memory
High hit rate drops the amount of bus traffic
But incoherency

Scalability
Different method to connect the memory with the CPU \(\rightarrow\) divide the memory in modules

Homogeneous Multicomputer Systems

CPU-to-CPU communication
aka System Area Networks (SANs)

Bus-based: connected through a multi-access network such as Fast Ethernet, problem?

Switch-based: routed instead of broadcast
Different topologies

Hardware Concepts

Multicomputers

Grid Hypercube (n-dimensional cube)
Massively Parallel Processors (MPPs)
Clusters of Workstations (COWs)
Heterogeneous Multicomputer Systems

Heterogeneous machines
- High performance parallel systems (multiprocessors and multicomputers)
- High-end PCs and workstations (servers)
- Simple network computers
- Mobile computers (laptops, palmtops)
- Multimedia workstations

and interconnection networks
- Local-area gigabit networks
- Wireless connections
- Long-haul high-latency connections
- Wide-area switched megabit connections

Scale
Lack of global view
transparency is harder

Software Concepts

• Much like an OS (resource managers, hides underlying hardware)
• Tightly-coupled (maintain a global view) - loosely coupled
  - DOS (Distributed Operating System)
  - NOS (Network Operating System)
  - Middleware

Provide distribution transparency
Additional layer atop of NOS implementing general-purpose services
Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)

Distributed Operating Systems

• Two types: multiprocessor OS and multicomputer OS
  Multi-processor OS
  Shared memory
  Functionality similar to traditional OSes but handle multiple CPUs
  - Aim at supporting high performance through multiple CPUs, make their number transparent to the application
  - Similar to multitasking uniprocessor OS:
    a) All communication done by manipulating data at shared memory locations.
    b) Protection is done through synchronization primitives

Multi-computer Operating Systems

Harder than traditional (multiprocessor) OS: Because memory is not shared
Emphasis shifts to processor communication by message passing
- OSs on each computer knows about the other computers
- OS on different machines generally the same
- Services are generally (transparently) distributed across computers

Multicomputer Operating Systems

General structure

Each node has its own kernel: modules for managing local resources (memory, local CPU, local disk, etc) + handling interprocess communication (sending and receiving messages to and from other nodes)

Common layer of software: implements the OS as a virtual machine supporting parallel and concurrent execution of tasks

Facilities: assigning a task to a processor, providing transparent storage, general interprocess communication
Multicomputer Operating Systems

Processor communication by message passing
- Often no simple global communication (e.g., broadcast)
- No simple system-wide synchronization mechanisms
- Virtual (distributed) shared memory requires OS to maintain global memory map in software (Distributed Shared Memory (DSM) vs Only message passing)
- Inherent distributed resource management: no central point where allocation decisions can be made

Practice: only very few truly multicomputer OS exist

Semantics of message passing

Buffering of messages at the sender or the receiver

Four possible synchronization points:
- S1 (block the sender when its buffer is full)
- S2 (message has been send)
- S3 (message has arrived at the receiver)
- S4 (message has been delivered to the receiver)

Is the communication reliable?

<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 not full</td>
<td>Yes</td>
<td>Not necessary</td>
</tr>
<tr>
<td>Block sender until message sent</td>
<td>No</td>
<td>Not necessary</td>
</tr>
<tr>
<td>Block sender until message received</td>
<td>No</td>
<td>Necessary</td>
</tr>
<tr>
<td>Block sender until message delivered</td>
<td>No</td>
<td>Necessary</td>
</tr>
</tbody>
</table>

Distributed Shared Memory Systems (DSMs)

The address space is divided up into pages with the pages being spread over all the processors in the system

When a processor references an address that is not present locally, a trap occurs, and the OS fetches the page

Network Operating System

Do not assume that the underlying hardware is homogeneous and that it should be managed as if it were a single system

Provide facilities to allow users to make use of services provided on a specific machine

Some provide a shared global file system
Different clients may mount the servers in different places.

Some characteristics:
- Each computer has its own OS with networking facilities
- Computers work independently (i.e., they may even have different OS)
- Services are to individual nodes (ftp, telnet, www)
- Highly file oriented (basically, processors share only files)
- Compared to distributed OSs
  - Lack of transparency (harder to use; need to be managed independently)
  - Easier to add/remove a machine (scalability, openness)

Middleware

DOS: transparency
NOS: scalability & openness

Middleware: add a layer on top of a NOS for transparency

Middleware Models

Based on some model or paradigm, such as:
- All resources are treated as files (UNIX and Plan 9)
- Distributed file systems
- Remote Procedure Calls (RPCs): allow a process to call a procedure whose implementation is located on a remote machine
- Distributed objects: transparently invoke objects residing on remote machines
- Distributed documents

Middleware Services

Communication facilities (offer high-level communication facilities to hide low-level message passing)
- Procedure calls across networks
- Remote-object method invocation
- Message-queuing systems
- Advanced communication streams
- Event notification service
Middleware Services

Information system services (help manage data)
- Large scale system-wide naming services
- Advanced directory services (search engines)
- Location services for tracking mobile objects
- Persistent storage facilities
- Data caching and replication

Middleware Services

Control services (giving applications control over when, where and how they access data)
- Code migration
- Distributed transaction processing

Security services
- Authentication and authorization services
- Simple encryption services
- Auditing service

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.

Comparison between Systems

<table>
<thead>
<tr>
<th>Item</th>
<th>Distributed OS</th>
<th>Network OS</th>
<th>Middleware-based OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree of transparency</td>
<td>Very High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Same OS on all nodes</td>
<td>Yes</td>
<td>No</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</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>

Clients and Servers

Processes are divided into

Servers: implementing a specific service
Clients: requesting a service from a server by sending it a request and subsequent waiting for the server's reply
Distributed across different machines

Follow a request-reply:

Client
Wait for result

Server
Provide service

Time

The Client-Server Model

(and other system architectures)
Application Layering

Traditional three-layered view

User-interface layer: programs that allow end users to interact with the application; differ in their sophistication

Processing layer: contains the functions of an application

Data layer: contains the data that a client wants to manipulate through the application components (persistence, consistency, data independence)

Multitiered Architectures

Alternative client-server organizations

Vertical distribution: placing logically different components on different machines

Horizontal distribution: a client or server may be physically split up into logically equivalent parts; each operating on its own share of the complete data

Alternative Architectures

An example of a server acting as a client.
Modern Architectures
An example of horizontal distribution of a Web service.

Alternative Architectures
Cooperating servers: service is physically distributed across a collection of services:
- Traditional multi-tiered architectures
- Replicated files systems
- Network news services
- Large-scale naming systems, etc

Cooperating clients: distributed applications exist by virtue of client collaboration:
- Teleconferencing
- Publish/subscribe

Collaborating servers

Proxy servers

Web applets
a) client request results in the downloading of applet code

b) client interacts with the applet

Alternative Architectures
Peer-to-Peer Systems
Basics