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

- 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)

Definition of a Distributed System

A distributed system is:

A collection of independent computers that appears to its users as a single coherent system

Two aspects:

(1) Independent computers
(2) Single system → middleware

Characteristics

(1) Heterogeneity hidden
(2) Interact with a consistent and uniform way
(3) 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, Andrews)
- Storage Systems (Ocean)
- Object-based Systems (CORBA, DCOM, etc)
- Groupware

A typical portion of the Internet

- ISPs: 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 filtering 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, infra-red link

Resource sharing on the Web

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

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>181</td>
<td>0</td>
</tr>
<tr>
<td>1989, July</td>
<td>130,000</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>1,203,096</td>
<td>6</td>
</tr>
<tr>
<td>1999, July</td>
<td>56,218,000</td>
<td>6,598,697</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

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

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
Transparency in a Distributed System

**Failure Transparency**
Hide the failure and recovery of a resource

*L. Lamport: You know you have one (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

**Persistent Transparency**
Hide whether a (software) resource is in memory or disk

---

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 be moved 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.

Concurrent transparency: enables several 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.

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.

---

Goals

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

---

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)

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

---

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 extent can work together
• Portability: to what extent 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 the 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 processes)
• geographical (maximum distance between nodes)
• administrative (number of administrative domains)

The (non) solution: powerful servers

Scalability Problems

<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

Scaling Techniques

Three techniques:
• hiding communication latencies
• distribution
• replication

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

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

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

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

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

Caching (client-driven)

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

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

**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 a computer network 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**

Based on whether they set time limits (lower and upper bounds) on:

- Process execution speeds
- Message transmission delays
- Clock drift rates

**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.

Fail-stop crash is other processes can detect certainly that the process has crashed.
Failure Model

Communication omission failures: send-omission, channel-omission, and receive omission

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 fails and remains halted. Other processes may not detect this state.</td>
</tr>
<tr>
<td>Crash</td>
<td>Process</td>
<td>Process fails 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 that process does not receive it.</td>
</tr>
<tr>
<td>Arbitrary (Byzantine)</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</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Process</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</td>
<td>Process exceeds the bounds on the interval between two steps.</td>
</tr>
<tr>
<td>Performance</td>
<td>Channel</td>
<td>A message's transmission takes longer than the stated bound.</td>
</tr>
</tbody>
</table>

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 a principal
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
Homogeneous Multicomputer Systems

- Grid
- Hypercube (n-dimensional cube)
- 4-dimensional

Massively parallel processors (MPPs)
Clusters of Workstations (COWs)

Heterogeneous Multicomputer Systems

- Heterogeneous machines, interconnection networks
- 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

<table>
<thead>
<tr>
<th>System</th>
<th>Description</th>
<th>Main Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOS</td>
<td>Tightly-coupled operating system for multi-processors and homogeneous multicomputers</td>
<td>Hide and manage hardware resources</td>
</tr>
<tr>
<td>NOS</td>
<td>Loosely-coupled 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: multiprocessor OS and multicomputer OS
  - Multi-processor OS
    - Shared memory
    - Functionality similar to traditional OSs but handle multiple CPUs
      - Aim at supporting high performance through multiple CPUs, make their number transparent to the application
      - Similar to multitasking uniprocessor OS:
        - All communication done by manipulating data at shared memory locations.
        - Protection is done through synchronization primitives
  - Multicomputer OS
    - Different computers, interconnected
    - Services are generally (transparently) distributed across computers
    - Emphasis shifts to processor communication by message passing
    - OSs on each computer knows about the other computers
    - OSs on different machines generally the same

Multicomputer 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
- OSs 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

Processor communication by message passing
- Often no simple global communication
- 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

Multicomputer Operating Systems

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.</th>
<th>guaranteed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block sender until buffer not full</td>
<td>Yes</td>
<td>Not necessary</td>
<td></td>
</tr>
<tr>
<td>Block sender until message sent</td>
<td>No</td>
<td>Not necessary</td>
<td></td>
</tr>
<tr>
<td>Block sender until message received</td>
<td>No</td>
<td>Necessary</td>
<td></td>
</tr>
<tr>
<td>Block sender until message delivered</td>
<td>No</td>
<td>Necessary</td>
<td></td>
</tr>
</tbody>
</table>

Multicomputer Operating Systems

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 (rlogin, rcp)

General structure
Network Operating System

Some provide a shared global file system

Distributed Systems, Spring 2003

Network Operating System

Different clients may mount the servers in different places.

Distributed Systems, Spring 2003

Network Operating Systems

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)

Distributed Systems, Spring 2003

Middleware

- Middleware itself does not manage an individual node
- OS on each computer need not know about the other computers
- OS on different computers need not be the same
- Services are generally (transparently) distributed across computers

Distributed Systems, Spring 2003

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

Distributed Systems, Spring 2003

Middleware Services

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

Distributed Systems, Spring 2003
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>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</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

Process are divided in
Server: implementing a specific service
Client: 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

The Client-Server Model
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.

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**

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

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

---

**Modern Architectures**

An example of horizontal distribution of a Web service.

---

**Extra Slides**

---

**Uniprocessor Operating Systems**

Separating applications from operating system code through a microkernel.

---

**Multiprocessor Operating Systems**

```c
monitor Counter {
    private:
        int count = 0;
    public:
        int value() { return count; }
        void incr() { count = count + 1; }
        void decr() { count = count - 1; }
}
```

A monitor to protect an integer against concurrent access.
A monitor to protect an integer against concurrent access, but blocking a process.

### Multiprocessor Operating Systems

```cpp
monitor Counter {

    private:
    int count = 0;
    int blocked_procs = 0;
    condition unblocked;

    public:
    int value () { return count; }
    void incr () {
        if (blocked_procs == 0)
            count = count + 1;
        else
            signal (unblocked);
    }
    void decr () {
        if (count == 0) {
            blocked_procs = blocked_procs + 1;
            wait (unblocked);
            blocked_procs = blocked_procs - 1;
        } else
            count = count - 1;
    }
}
```

### Distributed Shared Memory Systems

**a)** Pages of address space distributed among four machines

**b)** Situation after CPU 1 references page 10

**c)** Situation if page 10 is read-only and replication is used

### False sharing of a page between two independent processes.

### An Example Client and Server (1)

The `header.h` file used by the client and server.

### An Example Client and Server (2)

A sample server.

### An Example Client and Server (3)

A client using the server to copy a file.
Figure 2.13
Objects and principals

Figure 2.14
The enemy

Figure 2.15
Secure channels