Definition of a Distributed System

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

**Hardware:** set of autonomous machines

**Software:** single system
Distributed System

1.1 Distributed System Goals

- Connecting Users and Resources
  - Share resources
  - Collaborate and exchange
  - Security and tracking
- Openness
  - Interoperability
  - Portability
  - Extensibility
- Transparency
- Scalability

middleware layer extends across multiple machines
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 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>

Scalability

- with respect to size
- geographically
- administratively

will performance suffer?
Scalability Problems

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>

Scaling techniques

- hiding communication latencies
  - asynchronous communication
- distribution
- replication
Scaling Techniques: distribution

(a)

(b)

Scaling Techniques

distribution and replication
Hardware Concepts

Multiprocessors (1)

bus-based multiprocessor
Multiprocessors (2)

switch-based multiprocessors

Homogeneous Multicomputer Systems

Grid

Hypercube
Heterogeneous Multicomputer System

- Common place

Software Concepts

- Managing resources:
  - Connecting Users and Resources
    - Share resources
    - Collaborate and exchange
    - Security and tracking
  - Openness
    - Interoperability
    - Portability
    - Extensibility
  - Transparency
  - Scalability
Software Concepts

<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>

Uniprocessor Operating Systems

No direct data exchange between modules

OS interface

User application

Memory module

Process module

File module

User mode

Kernel mode

System call

Microkernel

Hardware
A monitor to protect an integer against concurrent access

A monitor to protect an integer against concurrent access, might cause process to block
Multicomputer Operating Systems

MOS: enable shared memory

Communication via message passing
Need for reliable communication

<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 (1)

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
Distributed Shared Memory Systems (2)

Problem: false sharing of a page between two independent processes

Network Operating System (1)

General structure of a network operating system.
Network Operating System (2)

Two clients and a server in a network operating system.

Network Operating System (3)

Different clients may mount the servers in different places.
Positioning Middleware

Middleware Models

- distributed file system
- remote procedure calls
- distributed objects
- distributed documents
Middleware Services

- communication facilities
- naming
- persistence
- distributed transaction
- 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.
## 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></td>
<td>Multiproc.</td>
<td>Multcomp.</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</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>

## Application Models

- client server
- layering
- multi-tiers
- horizontal distribution
Clients and Servers

General interaction between a client and a server.

An Example Client and Server (1)

The `header.h` file used by the client and server.
An Example Client and Server (2)

```
#include <header.h>

void main(void) {
    struct message m1, m2; /* incoming and outgoing messages */
    int r; /* result code */

    while(TRUE) { /* server runs forever */
        receive(FILE_SERVER, &m1); /* block waiting for a message */
        switch(m1.opcode) { /* dispatch on type of request */
            case CREATE: r = do_create(&m1, &m2); break;
            case READ: r = do_read(&m1, &m2); break;
            case WRITE: r = do_write(&m1, &m2); break;
            case DELETE: r = do_delete(&m1, &m2); break;
            default: r = E_BAD_OPCODE;
        }
        m2.result = r; /* return result to client */
        send(m1.source, &m2); /* send reply */
    }
}
```

A sample server

An Example Client and Server (3)

```
#include <header.h>

int copy(char *src, char *dst) {
    struct message m;
    long position;
    long client = 110;

    initialize(); /* prepare for execution */
    position = 0;
    do {
        mi.opcode = READ;
        mi.offset = position;
        mi.count = BUF_SIZE;
        strcpy(&mi.name, src);
        send(FILESERVER, &mi);
        receive(client, &mi);

        /* operation is a read */
        /* current position in the file */
        /* how many bytes to read */
        /* copy name of file to be read to message */
        /* send the message to the file server */
        /* block waiting for the reply */

        /* procedure to copy file using the server */
        /* message buffer */
        /* client's address */

        /* Write the data just received to the destination file. */
        mi.opcode = WRITE;
        mi.offset = position;
        mi.count = mi.result;
        strcpy(&mi.name, dst);
        send(FILESERVER, &mi);
        receive(client, &mi);
        position += mi.result;
    } while(mi.result > 0); /* iterate until done */
    return(mi.result == 0 ? OK : mi.result);
}
```

A client using the server to copy a file
Application Layering

Example: Internet search engine

Two-tiered Architecture
Two-tiered Architectures (2)

An example of a server acting as a client

Modern Architectures

Horizontal distribution of a Web service