Introduction

Chapter 1
A distributed system is:

A collection of independent computers that appears to its users as a single coherent system.
A distributed system organized as middleware.
Note that the middleware layer extends over 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>

Different forms of transparency in a distributed system.
## Scalability Problems

Examples of scalability limitations.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Example</th>
<th>Concept</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized services</td>
<td>A single server for all users</td>
<td>Centralized algorithms</td>
<td>Doing routing based on complete information</td>
</tr>
<tr>
<td>Centralized data</td>
<td>A single on-line telephone book</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The difference between letting:

a) a server or

b) a client check forms as they are being filled
Scaling Techniques (2)

An example of dividing the DNS name space into zones.
Different basic organizations and memories in distributed computer systems
Multiprocessors (1)

A bus-based multiprocessor.
Multiprocessors (2)

a) A crossbar switch

b) An omega switching network
Homogeneous Multicomputer Systems

(a) Grid

(b) Hypercube
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>

An overview between
• DOS (Distributed Operating Systems)
• NOS (Network Operating Systems)
• Middleware
Uniprocessor Operating Systems

Separating applications from operating system code through a microkernel.
A monitor to protect an integer against concurrent access.
A monitor to protect an integer against concurrent access, but blocking a process.
General structure of a multicomputer operating system

- Machine A
  - Distributed applications
  - Distributed operating system services
  - Kernel

- Machine B
  - Distributed applications
  - Distributed operating system services
  - Kernel

- Machine C
  - Distributed applications
  - Distributed operating system services
  - Kernel

Network
Alternatives for blocking and buffering in message passing.
<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>

Relation between blocking, buffering, and reliable communications.
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)

Two independent data items

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

General structure of a distributed system as middleware.
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></th>
<th>Network OS</th>
<th>Middleware-based OS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Multiproc.</td>
<td>Multicomp.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Degree of transparency</td>
<td>Very High</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Same OS on all nodes</td>
<td>Yes</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>
<td>N</td>
</tr>
<tr>
<td>Basis for communication</td>
<td>Shared memory</td>
<td>Messages</td>
<td>Files</td>
<td>Model specific</td>
</tr>
<tr>
<td>Resource management</td>
<td>Global, central</td>
<td>Global, distributed</td>
<td>Per node</td>
<td>Per node</td>
</tr>
<tr>
<td>Scalability</td>
<td>No</td>
<td>Moderately</td>
<td>Yes</td>
<td>Varies</td>
</tr>
<tr>
<td>Openness</td>
<td>Closed</td>
<td>Closed</td>
<td>Open</td>
<td>Open</td>
</tr>
</tbody>
</table>

A comparison between multiprocessor operating systems, multicomputer operating systems, network operating systems, and middleware based distributed systems.
Clients and Servers

General interaction between a client and a server.
An Example Client and Server (1)

/* Definitions needed by clients and servers. */
#define TRUE 1
#define MAX_PATH 255 /* maximum length of file name */
#define BUF_SIZE 1024 /* how much data to transfer at once */
#define FILE_SERVER 243 /* file server’s network address */

/* Definitions of the allowed operations */
#define CREATE 1 /* create a new file */
#define READ 2 /* read data from a file and return it */
#define WRITE 3 /* write data to a file */
#define DELETE 4 /* delete an existing file */

/* Error codes. */
#define OK 0 /* operation performed correctly */
#define E_BAD_OPCODE -1 /* unknown operation requested */
#define E_BAD_PARAM -2 /* error in a parameter */
#define E_IO -3 /* disk error or other I/O error */

/* Definition of the message format. */
struct message {
    long source; /* sender’s identity */
    long dest; /* receiver’s identity */
    long opcode; /* requested operation */
    long count; /* number of bytes to transfer */
    long offset; /* position in file to start I/O */
    long result; /* result of the operation */
    char name[MAX_PATH]; /* name of file being operated on */
    char data[BUF_SIZE]; /* data to be read or written */
};

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

```c
#include <header.h>
void main(void) {
    struct message ml, m2;  /* incoming and outgoing messages */
    int r;                   /* result code */

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

A sample server.
An Example Client and Server (3)

```c
#include <header.h>
int copy(char *src, char *dst){
    struct message ml;
    long position;
    long client = 110;
    initialize();
    position = 0;
    do {
        ml.opcode = READ;
        ml.offset = position;
        ml.count = BUF_SIZE;
        strcpy(&ml.name, src);
        send(FILESERVER, &ml);
        receive(client, &ml);
        /* Write the data just received to the destination file.*/
        ml.opcode = WRITE;
        ml.offset = position;
        ml.count = ml.result;
        strcpy(&ml.name, dst);
        send(FILE_SERVER, &ml);
        receive(client, &ml);
        position += ml.result;
    } while( ml.result > 0 );
    return(ml.result >= 0 ? OK : ml.result);
}
```

A client using the server to copy a file.
The general organization of an Internet search engine into three different layers.
Multitiered Architectures (1)

Alternative client-server organizations (a) – (e).

Alternative client-server organizations (a) – (e).
Multitiered Architectures (2)

An example of a server acting as a client.
Modern Architectures

An example of horizontal distribution of a Web service.