Introduction
Distributed Systems

The development of low cost powerful microprocessors together with the invention of high speed networks enable us to construct computer systems by connecting a large number of computers.

A distributed system is a collection of independent computers that appears to its users as a single coherent system.
Often a distributed system is organized as a middleware.

Note that the middleware layer extends over multiple machines.
Goals of distributed systems

When should we build a distributed system?

- Make resources easily accessible
- Provide distribution transparency
- Should be open
- Be scalable
Forms of transparency

<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>

Sometimes, due to eg performance/comprehensibility, we prefer to expose distribution rather than hide it
Openness

An open distributed system is a system that offers services according to standard service syntax/semantic descriptions (interfaces)

- Interoperability
- Portability
- Extensibility
- Separation of policy and mechanism
Scalability

Scalability with respect to

- Size
- Geographic distribution
- Administrative domains
**Scalability Limitations**

- Limitations to size scalability due to centralized solutions

<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>

- Limitations to geographic scalability due to assumptions on communications (synchronous vs asynchronous)
- Limitations to administrative scalability due to conflicting usage, management, and security policies
Scaling techniques

- Use asynchronous communication
- Move computation from the servers to the clients
- Use distribution, which is the splitting of a component into smaller parts and spread across the system
- Use replication
  - Need to handle replica consistency
Example scaling techniques: Web forms
Example scaling techniques: DNS
Peter Deutch identifies false assumptions made when building distributed systems

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator
Types of distributed systems

- Cluster computing systems
  - Beowulf and MOSIX clusters

- Grid computing systems
  - 4-layered architecture (fabric, connectivity, resource, and collective layers)

- Transaction processing monitors

- Pervasive systems
  - Systems with small, battery-powered, mobile, network capable embedded devices
  - Home systems
  - Body-Area networks
  - Sensor networks (in-network data processing)
Hardware Concepts

Different basic organizations and memories in distributed computer systems
A bus-based multiprocessor.
**Multiprocessors**

a) A crossbar switch

b) An omega switching network (shuffle exchange network)
Homogeneous Multicomputer Systems

- Grid
- Hypercube

(a) [Diagram of Grid]
(b) [Diagram of Hypercube]
Software Concepts

An overview of
- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)
- 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</td>
<td>Hide and manage hardware resources</td>
</tr>
<tr>
<td></td>
<td>multicomputers</td>
<td></td>
</tr>
<tr>
<td>NOS</td>
<td>Loosely-coupled operating system for heterogeneous multicomputers (LAN and</td>
<td>Offer local services to remote clients</td>
</tr>
<tr>
<td></td>
<td>WAN)</td>
<td></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

- Separating applications from operating system code through a microkernel.
Multicomputer Operating Systems

General structure of a multicomputer operating system

Machine A

Machine B

Machine C

Distributed applications

Distributed operating system services

Kernel

Kernel

Kernel

Network
Alternatives for blocking and buffering in message passing.
# Multicomputer Operating Systems

Relation between blocking, buffering, and reliable communications.

<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

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

- False sharing of a page between two independent processes.
General structure of a network operating system.
Network Operating System

- Two clients and a server in a network operating system.

Diagram:

- Client 1
- Client 2
- File server
  - Disks on which shared file system is stored
  - Network
    - Request
    - Reply
Different clients may mount the servers in different places.
Positioning Middleware

- General structure of a distributed system as middleware.
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

A comparison between multiprocessor operating systems, multicomputer operating systems, network operating systems, and middleware based distributed 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>Multicomp.</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>

CMSC 621 30
System architectures

- Architecture = components + connectors + configuration

Common architectures
- Layered
- Object-based
- Data –centered
- Event-based

Centralized architectures
- 2-tier Client-server and application layering
- Multi-tiered

Decentralized architectures
- Vertical and horizontal distribution
- Peer-to-peer architectures (structured vs unstructured)

Hybrid architectures
- Adaptive and self-managed systems
Clients and Servers

- General interaction between a client and a server.
An Example Client and Server

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

```c
/* 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 */
};
```
An Example Client and Server

A sample server.

```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 */
    }
}
```
An Example Client and Server

A client using the server to copy a file.

```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(FILESERVER, &ml);
        receive(client, &ml);
        position += ml.result;
    } while(ml.result > 0);

    return(ml.result >= 0 ? OK : ml result);
}
```
Application layering example

User interface

Keyword expression

Query generator

Database queries

Database with Web pages

HTML page containing list

HTML generator

Ranked list of page titles

Ranking component

Web page titles with meta-information

User-interface level

Processing level

Data level
Multitiered Architectures

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

- An example of a server acting as a client in a 3-tiered architecture

User interface (presentation) → Wait for result

Application server

Request operation

Wait for data

Database server

Request data

Return data

Return result

Time
Modern Architectures

- An example of horizontal distribution of a Web service.
Peer to Peer Systems

- Structured P2P: Distributed Hash Tables
  - CHORD and CAN
- Unstructured P2P
  - Random graphs: exchange of segments of partial views
- Hybrid + Superpeers