Distributed Systems
LEEC (2005/06 – 2º Sem.)

Introduction

João Paulo Carvalho

Universidade Técnica de Lisboa / Instituto Superior Técnico
Outline

• Definition of a Distributed System
• Goals
  – Connecting Users and Resources
  – Transparency
  – Openness
  – Scalability
• Hardware Concepts in Distributed Systems
  – Multiprocessors
  – Homogeneous Multicomputer Systems
  – Heterogeneous Multicomputer Systems
• Software Concepts in Distributed Systems
  – Distributed Operating Systems
  – Network Operating Systems
  – Middleware
• The Client-Server Model
  – Clients and Servers
  – Application Layering
  – Client-Server Architectures
Definition of a Distributed System

Computer Evolution + Computer Network Evolution = Distributed Systems

“A Distributed System is a collection of independent computers that appear to its users as a single coherent system”

Examples of Distributed Systems:
• Computer cluster in a university
• Rent-a-Car database and reservation system
• SETI@home
• WWW
Distributed System Goals

GOALS:

• Connect users and resources
• Transparency
• Openness
• Scalability
Connect users and resources

Resources = printers, computers, files, data, etc.

Why:

– Economic reasons and resource optimization:
  • It’s cheaper to share a printer than to buy one for each user…;
  • Take advantage of idle computers (SETI@home)…

– Cooperation and information exchange:
  • Groupware;
  • WWW…
DS Goals (3)

Transparency
Hide the fact that processes and resources are physically distributed across multiple computers
Different 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 is replicated</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>
Transparency (cont.)

There is a trade-off between Transparency and Performance.

- **Example 1 – Replication:** Ensuring that several replicas of a database scattered around the world are always consistent before allowing new operations might slow the system too much.

- **Example 2 – Concurrency:** Allowing a large number of users to access the same resource simultaneously obviously degrades performance without apparent reason for the user (the user is not aware of other users).
Openness

Offer services according to standard rules (protocols) that describe the syntax and semantics of those services

- Protocols are specified through Interfaces
- Interfaces allow different distributed separate systems to “communicate” with each other
- Interfaces are often described in Interface Description Language (IDL)
**DS Goals (6)**

**Scalability**

Provide systems that can grow in user and resource size, allow increased user and resource distance, and be able to manage additional independent administrative organizations

- **Scalability Problems:**
  - Centralized services, data and algorithms (single service, distributed users), suffer with user growth; However, decentralization brings additional problems like synchronization, replication, etc.
### DS Goals (7)

#### Scalability

- Scalability Problems (cont.):

<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>
Scalability

• Scalability Problems (cont.):
  – Geographical scalability problems result from the fact that inter-process communication is no longer instantaneous or reliable:
    • While delays can be in the order of microseconds in LANs, in WANs they can become hundreds of milliseconds. This means that synchronous communications require a great deal of care;
    • Communication must use point-to-point instead of broadcast
Scalability

• Scalability Problems (cont.):
  – Scaling a distributed system across multiple independent administrative domains brings the problem of conflicting policies regarding resource usage (and payment), management and security

• Scaling Techniques:
  – Hide communication latencies (use asynchronous communications and interrupts);
  – Transfer some of the processing from the server to the client;
DS Goals (10)

Scalability (scaling techniques – cont.)

(a) The server does all the work (checks each input from the client)

(b) The client checks the form as they are being filled
**DS Goals (11)**

**Scalability**

- **Scaling Techniques (cont.):**
  - Distribution (split a component into smaller more manageable parts). Example: DNS – Dividing the namespace into zones
Scalability

- Scaling Techniques (cont.):
  - Replication (increases availability, balances the load between components and can hide communication latency);
  - Caching (limited replication with consistency problems)
  - ...
Hardware Concepts in DS

- Hardware in Distributed Systems can be organized in several different ways:
  - Shared Memory (Multiprocessors, which have a single address space);
  - Private Memory (Multicomputers, each CPU has a direct connection to its local memory).
Hardware Concepts in DS(2)

Multiprocessors – Bus Based

- Have limited scalability
- Cache Memory help avoid bus overloading
Hardware Concepts in DS(3)

Multiprocessors – Switch Based

- Different CPUs can access different memories simultaneously
- The number of switches limits the number of CPUs that can access memory simultaneously

a) A crossbar switch

b) An omega switching network

(a) Crosspoint switch

(b) 2x2 switch
Multicomputers

- Homogeneous:
  - All CPUs and memory are identical;
  - Connected through a broadcast shared multi access network (like Ethernet) in bus based systems;
  - Messages routed through an interconnection network in switch-based multicomputers (e.g., grids, hipercubes…).

- Heterogeneous:
  - The most usual topology;
  - Computers may vary widely with respect to processor type, memory size, I/O bandwidth;
  - Connections are also diverse (a single multicomputer can simultaneously use LANs, Wide Area ATM, and frame relay networks);
  - Sophisticated software is needed to build applications due to the inherent heterogeneity;
  - Examples: SETI@home, WWW…
Uniprocessor Operating Systems

• An OS acts as a resource manager or an arbitrator
  – Manages CPU, I/O devices, memory
• OS provides a virtual interface that is easier to use than hardware
• Structure of uniprocessor operating systems:
  – Monolithic (e.g., MS-DOS, early UNIX)
    • One large kernel that handles everything
  – Layered design
    • Functionality is decomposed into N layers
    • Each layer uses services of layer N-1 and implements new service(s) for layer N+1
Microkernel architecture:
- Small kernel
- User-level servers implement additional functionality
Software Concepts in DS(3)

Distributed Operating Systems

- Act as resource managers for the hardware while attempting to hide intricacies and the heterogeneous nature of the underlying hardware
- Look to the user like a centralized OS
  - But operates on multiple independent CPUs
- Provide transparency
  - Location, migration, concurrency, replication,…
- Present users with a virtual uniprocessor
Types of Distributed Operating Systems

- **Tightly Coupled Systems:** The OS tries to maintain a single global view of the resources it manages.
- **Loosely Coupled Systems:** A collection of computers, each running its own OS. Each OS cooperates in order to make their own services and resources available to the others.

<table>
<thead>
<tr>
<th>System</th>
<th>Description</th>
<th>Main Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOS (Distributed Operating System)</td>
<td>Tightly-coupled operating system for multi-processors and homogeneous multicomputers</td>
<td>Hide and manage hardware resources</td>
</tr>
<tr>
<td>NOS (Network Operating System)</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>
DOS - Multiprocessor Operating Systems

- Like a uniprocessor operating system
- Manage multiple CPUs transparently to the user
- Each processor has its own hardware cache
  - Maintain consistency of cached data
- Communication through shared Memory
  - The OS must provide mechanisms to protect data against simultaneous access (Semaphors, Monitors, etc.)
DOS - Multicomputer Operating Systems

- Each computer has its own Kernel to manage local resources
- Communication through message passing, or
- Distributed shared memory (virtual shared memory)
Software Concepts in DS(7)

Network Operating Systems

- Machine A
  - Network OS services
  - Kernel
- Machine B
  - Network OS services
  - Kernel
- Machine C
  - Network OS services
  - Kernel

Distributed applications

Network

e.g.: rlogin; remote copy; etc.
Network Operating Systems (cont.)

- Employ a client-server model

- **Pros:**
  - Minimal OS kernel
  - Easy to add new machines to the system (usually all is needed is to connect the machine and make it known to the network)

- **Cons:**
  - Lack transparency
  - Are more vulnerable to attacks
Software Concepts in DS(9)

Middleware Based Systems

- **Middleware**: A software layer placed between the application/user layer and the operating system layer. Allows users and applications to “ignore” the differences in lower layers (OS, hardware, etc.)
- Introduce transparency
Middleware Based Systems (cont.)

• 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.
Middleware Based Systems (cont.)

- Middleware was not “invented”: It appeared naturally as manufacturers started to build higher level application-independent services into their networked systems (e.g.: support for distributed transactions; advanced communication facilities)

- Middleware Models:
  - Everything is a file;
  - Distributed file systems;
  - Remote Procedure Calls;
  - Distributed Objects;
  - Distributed Documents.
# Software Concepts in DS(12)

## 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>Multicomp.</td>
<td></td>
</tr>
<tr>
<td><strong>Degree of transparency</strong></td>
<td>Very High</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Same OS on all nodes</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Number of copies of OS</strong></td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td><strong>Basis for communication</strong></td>
<td>Shared memory</td>
<td>Messages</td>
<td>Files</td>
</tr>
<tr>
<td><strong>Resource management</strong></td>
<td>Global, central</td>
<td>Global, distributed</td>
<td>Per node</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>No</td>
<td>Moderately</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Openness</strong></td>
<td>Closed</td>
<td>Closed</td>
<td>Open</td>
</tr>
</tbody>
</table>
The Client-Server Model

Client-Server

- Distributed Systems have a high degree of diversity. Consensus lies in the Client-Server model, or clients that request services from servers
  - **Server**: a process implementing a specific service (e.g.: a file system service; a database service)
  - **Client**: a process that requests a service from a server by sending a request and subsequently waiting for the server reply
An example Client and Server

- *header.h* (used by 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 */
};
```
The Client-Server Model (3)

An example Client and Server (cont.)

- 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 */
    }
}
```
The Client-Server Model (4)

An example Client and Server (cont.)

- A sample 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); /* return OK or error code */
}
```

The Client-Server Model (5)

How to distinguish between a Client and a Server?

- Application Layering:
  - The user-interface level
  - The processing level
  - The data level

- Alternative Client-Server organizations:
The Client-Server Model (6)

How to distinguish between a Client and a Server? (cont.)

- Example: organization of an Internet search engine into three different layers

```
User interface

Keyword expression

Query generator

HTML generator

Ranking component

Data level

Database with Web pages

Web page titles with meta-information

Ranked list of page titles

Processing level

HTML page containing list

User-interface level
```

J.P. Carvalho

Sistemas Distribuídos 2005/2006

37
Multitiered Architectures

- A server may sometimes also act as a client (three-tiered architecture)
The Client-Server Model(8)

Other Client-Server Architectures

- Horizontal server distribution

- Distributed clients acting as servers (peer-to-peer)