Distributed Computing
Course Instructor: Dr. Safdar Ali
Text Book:
BOOKS

- Reference Books:
Course Evaluation

- Quiz: 10%
- Assignments: 20%
- Mid Term: 20%
- Final Exam: 50%
INTRODUCTION
Developments in Computer Systems??
INTRODUCTION

 Computer systems are undergoing a revolution. From 1945, when the modem computer era began, until about 1985, computers were large and expensive. Even minicomputers cost at least tens of thousands of dollars each.

 As a result, most organizations had only a handful of computers, and for lack of a way to connect them, these operated independently from one another.
INTRODUCTION

The amount of improvement that has occurred in computer technology in the past half century is truly staggering and totally unprecedented in other industries.

From a machine that cost 10 million dollars and executed 1 instruction per second, we have come to machines that cost 1000 dollars and are able to execute 1 billion instructions per second.
INTRODUCTION

- If cars had improved at this rate in the same time period, a Rolls Royce would now cost 1 dollar and get a billion miles per gallon.

- The second development was the invention of high-speed computer networks.
LAN and WAN??
INTRODUCTION

- Local-area networks or LANs allow hundreds of machines within a building to be connected in such a way that small amounts of information can be transferred between machines in a few microseconds or so.

- Larger amounts of data can be moved between machines at rates of 100 million to 10 billion bits/sec.
INTRODUCTION

Wide-area networks or WANs allow millions of machines all over the earth to be connected at speeds varying from 64 Kbps (kilobits per second) to gigabits per second.
INTRODUCTION

- The result of these technologies is that it is now not only feasible, but easy, to put together computing systems composed of large numbers of computers connected by a high-speed network.
INTRODUCTION

High speed connectivity is the only advantage?

Resource Sharing
• Online banking
• File sharing
• Emails etc
What is Distributed System?
INTRODUCTION

“A distributed system is a collection of independent computers that appears to its users as a single coherent system”
INTRODUCTION

In principle, *Distributed systems should be easy to expand or scale*
INTRODUCTION

- This characteristic is a direct consequence of having **independent computers**, but at the same time, hiding how these computers actually take part in the system as a whole.
INTRODUCTION

- A distributed system will normally be continuously available, although perhaps some parts may be temporarily out of order.

- Users and applications should not notice that parts are being replaced or fixed, or that new parts are added to serve more users or applications.
GOALS
GOALS

- A distributed system should make resources easily accessible;
- It should reasonably hide the fact that resources are distributed across a network;
- It should be open; and
- It should be scalable.
Making Resources Accessible

The main goal of a distributed system is to make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way.

Resources can be just about anything, but typical examples include:

- Printers
- Computers
- Storage facilities
- Files
- Web pages
Why Remote Access and Sharing?
Making Resources Accessible

- There are many reasons for wanting to share resources.
- One obvious reason is that of economics.
- For example, it is cheaper to let a printer be shared by several users in a small office than having to buy and maintain a separate printer for each user.
Making Resources Accessible

- Connecting users and resources also makes it easier to collaborate and exchange information:
  - Exchanging files
  - Mails
  - Documents
  - Audio, and
  - Video.
The connectivity of the Internet is now leading to numerous virtual organizations in which geographically widely-dispersed groups of people work together by means of groupware, that is, software for collaborative editing, teleconferencing, and so on.

Likewise, the Internet connectivity has enabled electronic commerce allowing us to buy and sell all kinds of goods without actually having to go to a store or even leave home.
Making Resources Accessible

- However, as connectivity and sharing increase, **security** is becoming increasingly important.

- In current practice, systems provide little protection against **eavesdropping** or **intrusion** on communication.

- **Passwords** and other sensitive information are often sent as clear text (i.e., **unencrypted**) through the network, or stored at servers that we can only hope are trustworthy.
Making Resources Accessible

- It is currently possible to order goods by merely supplying a credit card number. Rarely is proof required that the customer owns the card.
An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers.

<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>
</tbody>
</table>
Is the Transparency always a good idea?
There are situations in which attempting to completely hide all distribution aspects from users is not a good idea.

An example is requesting your electronic newspaper to appear in your mailbox before 7 A.M. local time, as usual, while you are currently at the other end of the world living in a different time zone. Your morning paper will not be the morning paper you are used to.
Likewise, a wide-area distributed system that connects a process in San Francisco to a process in Amsterdam cannot be expected to hide the fact that Mother Nature will not allow it to send a message from one process to the other in less than about 35 milliseconds.

In practice it takes several hundreds of milliseconds using a computer network. Signal transmission is not only limited by the speed of light, but also by limited processing capacities of the intermediate switches.
As a simple example, consider an office worker who wants to print a file from her notebook computer. It is better to send the print job to a busy nearby printer, rather than to an idle one at corporate headquarters in a different country.
Scalability
Synchronous and Asynchronous Communication?
ASYNCHRONOUS MESSAGES

- Asynchronous messaging involves a client that does not wait for a message from the server.

- Asynchronous messaging means that, it is a one way communication and the flow of communication is one way only.
ASYNCHRONOUS MESSAGES

- Asynchronous communication works much like the postal system: An application creates a message and labels the message with a destination address.

- Now the sender application proceeds happily, without needing to wait for the message to be delivered.
**Asynchronous Tools**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Useful for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web logs (Blogs)</td>
<td>Sharing ideas and comments</td>
</tr>
<tr>
<td>Messaging (e-mail)</td>
<td>One-to-one or one-to-many communications</td>
</tr>
<tr>
<td>Streaming audio</td>
<td>Communicating or teaching</td>
</tr>
<tr>
<td>Streaming video</td>
<td>Communicating or teaching</td>
</tr>
<tr>
<td>&quot;Learning objects&quot; (Web-based training)</td>
<td>Teaching and training</td>
</tr>
<tr>
<td>Document libraries</td>
<td>Managing resources</td>
</tr>
</tbody>
</table>
Synchronous messaging involves a client that waits for the server to respond to a message.

Sender will not send another message until get reply from receiver.
Synchronous messages

- Synchronous communication works much like a phone call. The Receiver (callee) must be available, otherwise the conversation cannot occur.
# Synchronous Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Useful for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio conferencing</td>
<td>Discussions and dialogue</td>
</tr>
<tr>
<td>Web conferencing</td>
<td>Sharing presentations and information</td>
</tr>
<tr>
<td>Video conferencing</td>
<td>In-depth discussions with higher-touch interactions</td>
</tr>
<tr>
<td>Chat</td>
<td>Information sharing of low-complexity issues</td>
</tr>
</tbody>
</table>
SCALABILITY

- Scalability of a system can be measured along at least three different dimensions:
  - First, a system can be scalable with respect to its size, meaning that we can easily add more users and resources to the system.
  - Second, a geographically scalable system is one in which the users and resources may lie far apart.
  - Third, a system can be administratively scalable.
SCALABILITY PROBLEMS

- When a system needs to scale, very different types of problems need to be solved.

- Let us first consider scaling with respect to **size**. If **more users or resources need to be supported**, we are often confronted with the limitations of **centralized services**.
SCALABILITY PROBLEMS

- Unfortunately, using only a single server is sometimes unavoidable.

- Imagine that we have a service for managing highly confidential information such as medical records, bank accounts and so on.
SCALABILITY PROBLEMS

- In such cases, it may be best to implement that service by means of a single server in a highly secured separate room, and protected from other parts of the distributed system through special network components.

- Copying the server to several locations to enhance performance maybe out of the question as it would make the service less secure.
Scalability Problems

- Geographical scalability has its own problems.

- In synchronous communication, a party requesting service, generally referred to as a client, blocks until a reply is sent back.
This approach generally works fine in LANs where communication between two machines is generally at worst a few hundred microseconds.

However, in a wide-area system, we need to take into account that inter-process communication may be hundreds of milliseconds, three orders of magnitude slower. Building interactive applications using synchronous communication in wide-area systems requires a great deal of care (and not a little patience).
Finally, a difficult, and in many cases open question is how to scale a distributed system across multiple, independent administrative domains.

A major problem that needs to be solved is that of conflicting policies with respect to resource usage (and payment), management, and security.
SCALABILITY PROBLEMS

- Many components of a distributed system that reside within a single domain can often be trusted by users that operate within that same domain.

- In such cases, system administration may have tested and certified applications, and may have taken special measures to ensure that such components cannot be tampered with. In essence, the users trust their system administrators.

- However, this trust does not expand naturally across domain boundaries.
Scaling Techniques
Previous Lecture Slide
**Scalability Problems**

- Geographical scalability has its own problems.

- In synchronous communication, a party requesting service, generally referred to as a client, blocks until a reply is sent back.
Scaling Techniques

Having discussed some of the scalability problems brings us to the question of how those problems can generally be solved.

There are now basically only three techniques for scaling:

- Hiding communication latencies,
- Distribution, and
- Replication
HIDING COMMUNICATION LATENCIES

- Hiding communication latencies is important to achieve geographical scalability.

- The basic idea is simple:
  - Try to avoid waiting for responses to remote (and potentially distant) service requests as much as possible.

- For example, when a service has been requested at a remote machine, an alternative to waiting for a reply from the server is to do other useful work at the requester's side.
HIDING COMMUNICATION LATENCIES

- Essentially, what this means is constructing the requesting application in such a way that it uses only asynchronous communication.

- However, there are many applications that cannot make effective use of asynchronous communication.

- In such cases, a much better solution is to reduce the overall communication by moving part of the computation that is normally done at the server to the client process requesting the service.
HIDING COMMUNICATION LATENCIES

- A typical case where this approach works is accessing databases using forms. Filling in forms can be done by sending a separate message for each field, and waiting for an acknowledgment from the server.
A much better solution is to ship the code for filling in the form, and possibly checking the entries, to the client, and have the client return a completed form.
Another important scaling technique is distribution.

Distribution involves taking a component, splitting it into smaller parts, and subsequently spreading those parts across the system.

An excellent example of distribution is the Internet Domain Name System (DNS).

The DNS name space is hierarchically organized into a tree of domains.
The DNS tree has a single domain at the top of the structure called the root domain. A period or dot (.) is the designation for the root domain. Below the root domain are the top-level domains that divide the DNS hierarchy into segments.
### DISTRIBUTION

<table>
<thead>
<tr>
<th>Domain</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>.com</td>
<td>Commercial organizations, as in novell.com</td>
</tr>
<tr>
<td>.edu</td>
<td>Educational organizations, as in ucla.edu</td>
</tr>
<tr>
<td>.gov</td>
<td>Governmental agencies, as in whitehouse.gov</td>
</tr>
<tr>
<td>.mil</td>
<td>Military organizations, as in army.mil</td>
</tr>
<tr>
<td>.org</td>
<td>Nonprofit organizations, as in redcross.org</td>
</tr>
<tr>
<td>.net</td>
<td>Networking entities, as in nsf.net</td>
</tr>
<tr>
<td>.int</td>
<td>International organizations, as in nato.int</td>
</tr>
</tbody>
</table>
DISTRIBUTION
DISTRIBUTION

```
root
```

```
com
```

```
Company A
(companya.com)
```

```
chicago
(chicago.companya.com)
```

```
providence
(providence.companya.com)
```

```
washington
(washington.companya.com)
```

```
host1
(host1.washington.companya.com)
```
For example, Company A creates a domain called companya.com under the .com top-level domain. Company A has separate LANs for its locations in Chicago, Washington, and Providence. Therefore, the network administrator for Company A decides to create a separate sub-domain for each division, as shown in Domains and sub-domains.

Any domain in a sub-tree is considered part of all domains above it. Therefore, chicago.companya.com is part of the companya.com domain, and both are part of the .com domain.
DISTRIBUTION

Consider, for example, the name \texttt{nl. vu.cs.flits}. To resolve this name, it is first passed to the server of zone Z1 which returns the address of the server for zone Z2, to which the rest of name, \texttt{vu.cs.flits}, can be handed.

The server for Z2 will return the address of the server for zone Z3, which is capable of handling the last part of the name and will return the address of the associated host.
DISTRIBUTION

[Diagram showing a tree structure with nodes labeled by generic categories and specific domains such as int, com, edu, gov, mil, org, net, jp, us, nl, etc., with additional nodes for specific domains like sun, yale, cs, eng, ai, linda, robot, acm, ieee, ac, co, oce, vu, keio, nec, csl, cs, pc24, flits, fluit, Z1, Z2, Z3, and so on.]
It is generally a good idea to actually replicate components across a distributed system.

Replication not only increases availability, but also helps to balance the load between components leading to better performance.

Also, in geo-graphically widely-dispersed systems, having a copy nearby can hide much of the communication latency problems mentioned before.
REPLICATION

- There is one serious drawback to replication that may adversely affect scalability.
- Because we now have multiple copies of a resource, modifying one copy makes that copy different from the others.
- Consequently, replication leads to consistency problems.
System Models
Types of Models

- Physical Model
- Architectural Model

Physical models are the most explicit way in which to describe a system; they capture the hardware composition of a system in terms of the computers (and other devices, such as mobile phones) and their interconnecting networks.
**Physical models**

- **Early distributed systems**: Such systems emerged in the late 1970s and early 1980s in response to the emergence of local area networking technology, usually Ethernet.

- These systems typically consisted of between 10 and 100 nodes interconnected by a local area network, with limited Internet connectivity and supported a small range of services such as shared local printers.
**Physical models**

- **Internet-scale distributed systems**: Building on this foundation, larger-scale distributed systems started to emerge in the 1990s in response to the dramatic growth of the Internet during this time.

- In such systems, an extensible set of nodes interconnected by a *network of networks (the Internet)*.

- Such systems exploit the infrastructure offered by the Internet to become truly global.
The level of heterogeneity in such systems is significant in terms of networks, computer architecture, operating systems, languages employed and the development teams involved.

This has led to an increasing emphasis on open standards and associated middleware technologies such as CORBA.
ARCHITECTURAL MODELS

Architectural models describe a system in terms of the computational and communication tasks performed by its computational elements; the computational elements being individual computers or aggregates of them supported by appropriate network interconnections.
CLIENT-SERVER

- This is the architecture that is most often cited when distributed systems are discussed. It is historically the most important and remains the most widely employed.
CLIENT-SERVER

- Figure illustrates the simple structure in which processes take on the roles of being clients or servers.
CLIENT-SERVER

- Servers may in turn be clients of other servers, as the figure indicates. For example, a web server is often a client of a local file server that manages the files in which the web pages are stored.
Web servers and most other Internet services are clients of the DNS service, which translates Internet domain names to network addresses.
In this architecture all of the processes involved in a task, interacting cooperatively as peers without any distinction between client and server processes or the computers on which they run.
The hardware capacity and operating system functionality of today’s desktop computers exceeds that of yesterday’s servers, and the majority are equipped with always-on broadband network connections.

The aim of the peer-to-peer architecture is to exploit the resources (both data and hardware) in a large number of participating computers for the fulfilment of a given task or activity.
A large number of data objects are shared, an individual computer holds only a small part of the application database, and the storage, processing and communication loads for access to objects are distributed across many computers and network links.
**Peer-to-peer architecture**

- Each object is replicated in several computers to further distribute the load and to provide resilience in the event of disconnection of individual computers.