Lectures: Weeks 1-3

- Introduction, characteristics of distributed systems, design issues, h/s concepts, distributed programming models
  Reading list: Tanenbaum Chapter 1, pp 1-42.

- Communication issues and network technologies
  Reading list: Tanenbaum Chapter 2.1, pp 57-68, Layered Protocols

- Models of distributed computation: synchronous vs asynchronous systems, failure models
  Reading list: Mullender Chapter 5.1-5.2, pp 97-102,

- Distributed computations, global system states, consistent cuts and runs
  Reading list: Mullender Chapter 4.
Introduction to Distributed Systems
Distributed Systems

Three Technology Advances:
- Development of powerful microprocessors
- Development of high-speed networks
- Development of denser and cheaper memory/storage

Easy: put together large # of powerful processors connected by a high-speed network.

Hard: SOFTWARE! SOFTWARE! SOFTWARE!
What is a distributed system?

“You know you have one when the crash of a computer you’ve never heard of stops you from getting any work done.” Leslie Lamport

- A collection of (perhaps) heterogeneous nodes connected by one or more interconnection networks which provides access to system-wide shared resources and services.

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

Examples?
Characteristics of a distributed systems

- **Multiple Computers:**
  More than one physical computer, each consisting of CPUs, local memory, and possibly stable storage, and I/O paths to connect it with the environment.

- **Interconnections:**
  Mechanisms for communicating with other nodes via a network.

- **Shared State:**
  If a subset of nodes cooperate to provide a service, a shared state is maintained by these nodes. The shared state is distributed or replicated among the participants.
A distributed system organized as middleware. Note that the middleware layer extends over multiple machines.
Distributed vs. Centralized Systems

Why distribute?
- Resource sharing
- Device sharing
- Flexibility to spread load
- Incremental growth
- Cost/performance
- Reliability/Availability
- Inherent distribution
- Security?
Why NOT distribute?
- Software
- Network
- Security
- System management

Numerous sources of complexity including:
- Transparent/uniform access to data or resources
- Independent failures of processors (or processes)
- Dynamic membership of a system
- Unreliable/unsecured communication
- Overhead in message communication
- Distribution or replication of data (or meta-data)
- Lack of clean common interfaces
Design Goals & Issues

- Connecting users and resources is the primary goal
- Transparency: hide the fact that processes and resources are physically distributed
- Openness: offer services according to rules and interfaces that describe the syntax and semantics of those services
  - Interoperability and portability
  -- Separating policy from mechanism
- Scalability
- Performance
- Dependability
## 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.
How to achieve “single-system image”? How to hide distribution from users or programs?

Is it a good idea?

Sometimes requires trade off transparency for performance
Scalability

- The challenge is to build distributed systems that scale with the increase in the number of CPUs, users, and processes, larger databases, etc.

- Scalability along several dimensions: size, geography, administrative domains
### Scalability Problems

<table>
<thead>
<tr>
<th>Concept</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Centralized services</strong></td>
<td>A single server for all users</td>
</tr>
<tr>
<td><strong>Centralized data (tables)</strong></td>
<td>A single on-line telephone book</td>
</tr>
<tr>
<td><strong>Centralized algorithms</strong></td>
<td>Doing routing based on complete information</td>
</tr>
</tbody>
</table>

Examples of scalability limitations.
A very simple principle:
Avoid centralized services, centralized tables, and centralized algorithms

Characteristics of decentralized algorithms:
- No machine has complete information about the system state
- Machines make decisions based only on local information
- Failure of one machine does not ruin the algorithm
- There is no implicit assumption about a global clock
A few lessons from AFS:
- “Clients have cycles to burn.”
- “Cache whenever possible.”
- “Exploit usage properties.”
- “Minimize system-wide knowledge/change.”
- “Batch if possible.”
- Multicast often works!
Scaling Techniques (Tanenbaum’s Text)

- Hiding communication latencies
  - Asynchronous communication
  - Function shipping to clients

- Distribution of components
  - DNS name space

- Caching and Replication
  - Maintaining consistency
Scaling Techniques (1)

1.4 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.
Performance

Various performance metrics:
- response time
- throughput
- system utilization
- network capacity utilization

Key issue in parallelizing computations in a distributed system?

overhead of message communication
Performance

Trade off:
- More tasks → more parallelism → better performance
- More tasks → more communication → worse performance

Grain size affects # of messages:
- fine-grained parallelism vs. coarse-grained parallelism
- small computations vs. large computations
- high interaction rate vs. low interaction rate
Dependability

- Reliability: measured by the probability $R(t)$ that the system is up (and providing correct service) during the time interval $[0, t]$ assuming that it was operational at time $t$.
- Availability: measured by the probability $A(t)$ that the system is operational at the instant of time $t$. As $t \rightarrow \infty$, availability expresses the fraction of time that the system is usable.
- Timeliness: ability to meet timing constraints imposed on task execution or service delivery.
- Integrity: replicated copies of data must be kept consistent.
- Security: protection from unauthorized usage/access. Why more difficult in distributed systems?
Distributed Programming Paradigms

- Client/server model
- Remote procedure calls
- Distributed File Systems
- Group communication and multicasts
- Distributed transactions
- Distributed shared memory
- Distributed object-based systems
- Publish-subscribe model
- Peer-to-peer model
- The Web
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

(a) Crosspoint switch

(b) 2x2 switch
Homogeneous Multicomputer Systems

a) Grid
b) Hypercube
Heterogeneous Multicomputer Systems

- Most distributed systems today are built on top of heterogeneous multicomputers and interconnection networks
- No global system view
- Sophisticated software needed to support distributed applications
# 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 of
- **DOS** (Distributed Operating Systems)
- **NOS** (Network Operating Systems)
- **Middleware**
Uniprocessor Operating Systems

- Separating applications from operating system code through a microkernel.
Multiprocessor Operating Systems (1)

```cpp
monitor Counter {
    private:
        int count = 0;
    public:
        int value() { return count;}
        void incr () { count = count + 1;}
        void decr() { count = count - 1;}
}
```

- A monitor to protect an integer against concurrent access.
A monitor to protect an integer against concurrent access, but blocking a process.
Multicomputer Operating Systems (1)

- General structure of a multicomputer operating system

```
+----------------+     +----------------+     +----------------+
|                 |     |                 |     |                 |
| Machine A       |     | Machine B       |     | Machine C       |
|                 |     |                 |     |                 |
| Distributed applications | | Distributed operating system services | |
|                 |     |                 |     |                 |
| Kernel          |     | Kernel          |     | Kernel          |
+----------------+     +----------------+     +----------------+
```

Network
Multicomputer Operating Systems (2)

- Alternatives for blocking and buffering in message passing.
## Multicomputer Operating Systems (3)

<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.
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)

- 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.

Diagram:

- Machine A
  - Distributed applications
  - Middleware services
  - Network OS services
  - Kernel
- Machine B
  - Distributed applications
  - Middleware services
  - Network OS services
  - Kernel
- Machine C
  - Distributed applications
  - Middleware services
  - Network OS services
  - Kernel

Network
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>