CS4513
Distributed Computer Systems

Introduction
(Ch 1: 11-1.2, 1.4-1.5)

Outline
• Overview
• Goals
• Software
• Client Server

The Rise of Distributed Systems
• Computer hardware prices falling, power increasing
  - If cars the same, Rolls Royce would cost 1 dollar and get 1 billion miles per gallon (with 200 page manual to open the door)
• Network connectivity increasing
  - Everyone is connected with fat pipes
• It is easy to connect hardware together
• Definition: a distributed system is
  - A collection of independent computers that appears to its users as a single coherent system.

Definition of a Distributed System
A distributed system organized as middleware.
Note that the middleware layer extends over multiple machines.
Users can interact with the system in a consistent way, regardless of where the interaction takes place

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>Allocation</td>
<td>Hide that a resource may not be used when 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>Resistance</td>
<td>Hide whether a software resource is in memory or on disk</td>
</tr>
</tbody>
</table>

Scalability Problems
• As distributed systems grow, centralized solutions are limited
  - Consider LAN name resolution vs. WAN

<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>

• Sometimes, hard to avoid (consider a bank)
• Need to collect information in distributed fashion and distributed in a distributed fashion
• Challenges:
  - geography, ownership domains, time synchronization
Scaling Techniques: Hiding Communication Latency
- Especially important for interactive applications
- If possible, do asynchronous communication
  - Not always possible when client has nothing to do
  - Instead, can hide latencies

Scaling Techniques: Distribution
1. Example: DNS name space into zones
   - (nl.vu.cs.fluit – z1 gives address of vu gives address of cs)

Scaling Techniques: Replication
- Copy of information to increase availability and decrease centralized load
  - Example: P2P networks (Gnutella +)
  - distribute copies uniformly or in proportion to use
  - Example: akamai
  - Example: Caching is a replication decision made by client
- Issue: Consistency of replicated information
  - Example: Web Browser cache

Outline
- Overview (done)
- Goals (done)
- Software
  - Client Server

Software Concepts

<table>
<thead>
<tr>
<th>System</th>
<th>Description</th>
<th>Main Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOS</td>
<td>Lightly-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>

- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)
- Middleware

Uniprocessor Operating Systems
- Separating applications from operating system code through a microkernel
  - Can extend to multiple computers
Multicomputer Operating Systems

- But no longer have shared memory
  - Can try to provide distributed shared memory
  - Can provide message passing

Multicomputer Operating Systems

- Message passing primitives vary widely between systems
  - Example: consider buffering and synchronization

Distributed Shared Memory Systems

<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.
- These issues make synchronization harder. It was easier when we had shared memory.
  - So.. distributed shared memory

Distributed Shared Memory Systems

- Issue: how large should page sizes be? What are the tradeoffs?

Network Operating System

- OSes can be different (Windows or Linux)
- Typical services: rlogin, rcp
  - Fairly primitive way to share files
Network Operating System

- Can have one computer provide files transparently for others (NFS)
  - (try a "df" on the WPI hosts to see. Similar to a "mount network drive" in Windows)
  - Different clients may mount the servers in different places
  - Inconsistencies in view make NOSes harder, in general for users than DOSes.
    - But easier to scale by adding computers

Positioning Middleware

- Network OS not transparent. Distributed OS not independent computers.
  - Middleware can help
  - Much middleware built in-house to help use networked operating systems (distributed transactions, better comm, RPC)
  - Unfortunately, many different standards

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.
  - If different, compatibility issues
  - If incomplete, then users build their own or use lower-layer services (frowned upon)

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>Degree of transparency</td>
<td>High</td>
<td>Low</td>
<td>Low, High</td>
</tr>
<tr>
<td>Number of copies of OS</td>
<td>1</td>
<td>1</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>Local, central</td>
<td>Global, distributed</td>
<td>Per node, N node</td>
</tr>
<tr>
<td>Scalability</td>
<td>No</td>
<td>Moderately</td>
<td>Yes, Yes</td>
</tr>
<tr>
<td>Openness</td>
<td>Closed</td>
<td>Closed</td>
<td>Open, Open</td>
</tr>
</tbody>
</table>

- DOS most transparent, but closed and only moderately scalable
- NOS not so transparent, but open and scalable
- Middleware provides a bit more transparency than NOS
Clients and Servers

• Thus far, have not talked about organization of processes
  - Again, many choices but most agree upon client-server

• If can do so without connection, quite simple
• If underlying connection is unreliable, not trivial
  - Resend? What if receive twice

• Use TCP for reliable connection (apps on Internet)
  - Not always appropriate for high-speed LAN connection

Example Client and Server: Header

* Used by both the client and server.

Example Client and Server: Server

Example Client and Server: Client

* One issue, is how to clearly differentiate

Client-Server Implementation Levels

* Example of an Internet search engine
  - UI on client
  - Processing can be on client or server
  - Data level is server, keeps consistency

Multitiered Architectures

* Thin client (a) to Fat client (e)
  - (d) and (e) popular for NOS environments
Multitiered Architectures: 3 tiers

- Server may act as a client
  - Example would be transaction monitor across multiple databases

Modern Architectures: Horizontal

- Rather than vertical, distribute servers across nodes
  - Example of Web server “farm” for load balancing
  - Clients, too (peer-to-peer systems)