Business and Logistics

☐ Programming exercise
  ☐ Work on your own and challenge yourself
    ➢ if you get stuck, ask a classmate
  ☐ Some flexibility in what you implement
    ➢ might think of both a TCP and UDP approach
  ☐ Goal is to get your feet wet and learn

☐ Term paper assignment
  ☐ Begin thinking about your topic

☐ Survey
  ☐ Please fill out and send back by the end of today

☐ Term projects
  ☐ Please begin thinking about possible ideas
Last Lecture

- Course introduction
- Broad overview of what a distributed system is
  - Set of computers connected by a network
  - Present multiple challenges to design and development
- Transparency is a way to think about objectives
  - Separation of concerns and components
  - Access, location, failure, …
Lecture Objectives

- Provide conceptual models to motivate distributed system design problems and solutions
- Look at architectural models
  - High-level view of distribution of functionality
  - Structure of and relationships between components
  - Useful for evaluating properties of distributed systems
- Look at functional models
  - Vertical views or slices representing some key aspect
  - Sets of issues that must be addressed in design
- Emphasis on middleware to support development
- Synchronization, reliability, end-to-end argument
Distributed System Models

- Architecture model of distributed system concerned with placement of parts and their relationships
  - Defines (abstracts) the components of the system
  - Defines how they map to underlying network/computers
  - Looks at relationships (roles and patterns) between them
- Fundamental models are concerned with more formal descriptions of properties (correctness, reliability, …)
- Distributed system characteristics addressed by:
  - Interaction model
  - Failure model
  - Security model
  - Performance model
  - Workload model
Difficulties for / Threats to Distributed Systems

- Widely varying models of use
  - Components are subject to wide variations in workload
  - Some parts of system are poorly connected
  - Applications have different requirements
    - bandwidth and latency
- Wide range of system environments
  - Heterogeneous hardware, OS, and networks
  - Networks vary in performance
  - Systems have widely differing scales
- Internal problems - clocks, data, component failure
- External problems - attacks, data integrity, denial
Lamport’s Definition of a Distributed System

- Lamport once defined a distributed systems as:
  “One on which I cannot get any work done because some system I never heard of has crashed.”

- An example that leads to Lamport’s concern is NSF
  - How many of us have ever seen workstations freeze because the NSF server failed?
  - My Micro!@#$ applications routinely fail when I have remotely mounted files system

- Applications need to adapt gracefully in the face of partial failure
  - Distributed file systems are hard to implement
Architecture Models (Design)

- Goal is to ensure the *structure* will meet requirements
- First simplify and abstract functions of individual components of distributed system
- Then consider:
  - Placement across network of computers, seeking to define useful patterns for data distribution / workload
  - Inter-relationships between components, their functional roles and communication patterns
- Classification helps in simplification
  - Processes: server, client, peer
  - Identifies: responsibilities, behavior, workload, failure
  - Analysis used to specify placement to meet objectives
Software Layers

- Let’s first look at software layering in support of distributed systems models
- *Software architecture* refers to structuring of software
  - Layers (software modules)
  - Services (service layers) (server)
- Platform
  - Lowest-level hardware and software layers (e.g., OS)
- Middleware
  - Layer of software to mask heterogeneity and provide convenient *programming model* for applications
  - Provides useful building blocks to develop software
  - Raises level of communication activities through support for communication abstractions / mechanisms
  - Make distributed nature transparent!
Software and Hardware Services Layers

Applications, services

Middleware

Operating system

Computer and network hardware

Support for transparency in interfaces

Platform

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
Middleware Packages

- Remote procedure call (RPC)
- Group communication (Isis)
- Object-oriented
  - Common Object Request Broker Architecture (CORBA)
  - Java Remote Method Invocation (RMI)
  - Microsoft Distributed Common Object Model (DCOM)
- Packages provide higher-level applications services
  - Naming, security, transactions
  - Persistent storage, event notification
- Look for opportunities for “standardizing”
Middleware Limitations

☐ End-to-end argument (Saltzer, Reed, Clarke, 1984)

Some communication-related functions can be completely and reliably implemented only with the knowledge and help of the application standing at the endpoints of the communication system. Therefore, providing that function as a feature of the communication system itself is not always sensible.

☐ This runs counter to the view that all communication activities can be abstracted away by middleware layers

☐ Correct behavior in distributed programs depends upon error measures and security at all levels

❖ Example: fault-tolerant, reliable, end-to-end transfer
Examples Illustrating the End-to-End Argument
Functional View of Middleware

- Information exchange services
  - Fundamental message passing capability

- Application-specific services
  - Specialized services
  - Example: transactional/replication services for distributed database
  - Example: groupware services for collaborative applications

- Management and support services
  - Needed for locating distributed resources
    - name services, registries
  - Administering resources across the network
  - Monitoring performance and behavior in distributed system
Commercial (Production) Middleware

- These are encountered in the real world
- Single-service middleware components
  - HTTP for retrieving remote documents
  - SUN RPC for remote procedure call
  - SSL for security in socket communication
- Multi-service (integrated) middleware environments
  - Integrates multiple components in a coherent package
    - integrates RPC, security, directory, time, file services
  - Example: DCE, CORBA, DCOM, .NET, Java
- Compound middleware environments that combine many middleware environments into a single framework
  - Example: transaction management + RPC/RMI
System Architectures

- Concerned with division of responsibilities
  - Between system components (apps, servers, processes)
  - Placement on computers in the network
- Implications for performance, reliability, and security
- Types (focus on placement)
  - Client-server model (historically most important)
  - Services provided by multiple servers
  - Proxy servers and caches
  - Peer-to-peer processes
  - Mobile code / agents / spontaneous networking
  - Network computers / thin clients
Client-Server Model with Multiple Servers

- Most common distributed system architecture
- Defines interaction relationship
  - *Service*: task some machine can perform
  - *Server*: machine that performs the task (when requested)
  - *Client*: machine that is requesting the service
- Model allows chaining, hierarchy, exchange of roles
  - Servers may be clients of other servers
    - example: web server being a client of local file server
- Service types
  - Directory service, print service, file service, …
Services may be implemented by distributed processes

- May require distributed resources (e.g., WWW)
- May choose to partition and distribute for reliability
- Replication used to:
  - increase performance
  - increase availability
  - improve fault tolerance

Scalability issues

- Centralization or fixed allocation of service
- Management of services
Clients Invoke Individual Servers

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
A Service Provided by Multiple Servers

Benefits?

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
More on Client-Server Model

☐ Clients
  - Generally blocks until server responds or time outs
  - Typically invoked by end users when service is required
  - Interacts with users through a user interface
  - Interacts with client middleware using middleware API
    - raises abstraction of underlying network connectivity

☐ Server
  - Implements services
  - Usually waits for incoming requests
  - Usually a program with special privileges
  - Invoked by server middleware
  - Provide error recovery and failure handling services
Client-Server Middleware

Client

- Client processes
- Client middleware
- Local services
- Network services
- OS and H/W

Server

- Client processes
- Server middleware
- Local services
- Network services
- OS and H/W

exchange protocol (logical)

network protocol
Variations on Client-Server Model

- Derived from consideration of several factors
  - Use of multiple servers / caches
    - performance
    - resilience
  - Mobile code and mobile agents
  - Users’ need for low-cost computers
  - Requirements to add/remove mobile devices
- Services provided by multiple servers
- Proxy servers and caches
- Mobile code and agents
- Network computers and thin clients
Proxy Servers and Caches

- Proxy servers used to increase availability and performance of services by reducing the load on the network and servers
  - Separation of service functionality
  - Sharing of proxy server among clients

- A cache stores recently used data objects closer to users of the objects than the actual objects themselves
  - Cache is check first when data is required
  - If present in cache (hit), data is taken from there
  - Otherwise, fetched from actual
  - Cache aid in performance improvement
Web Proxy Server

Issues?

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
Applets

☑ Applets are well-known and widely used example
  ◆ Example: web applets

☑ Help to achieve better performance
  ◆ Does not suffer from delays or bandwidth variability associated with network communication

☑ Accessing services means running code that can invoke their operations
  ◆ Pull model: client initiates
  ◆ Push model: server initiates
Web Applets

a) client request results in the downloading of applet code

b) client interacts with the applet

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
Mobile Code and Mobile Agents

- Applets are an example
- Code to access specialize services
  - Dynamically depend on service type
  - Location (presence) dependent
- Mobile code moves to clients
- Mobile agent is a running program (code and data)
  - Travels from computer to computer performing tasks
    - example: collecting information
  - Can reduce communication costs and invocation time
  - Example: use mobile agent to install software
  - May be a security threat
Spontaneous Networking in a Hotel

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
Thin Clients

- Separates functionality between local and remote
- Bulk of processing done on the server
  - Little administration on client
  - Reduced hardware (e.g., sometime diskless)
- Client mostly responsible for user interactions /graphics
- X terminals are the classic example
  - X-11 windowing system, protocol
  - Client ran X server (interaction for remote program)
- Modern day equivalents are remote desktop services
  - VNC, Apple Remote Desktop, GotoMyPC
- Web browser as the thin client (app runs elsewhere)
Thin Clients and Compute Servers

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
Multi-Tier Client-Server Architectures

☐ Two-tier architecture
  ☐ Common from mid 1980’s to early 1990’s
  ☐ UI on user’s desktop
  ☐ Application services on server
  ☐ Performance deteriorates with large user communities
    ➢ server may spend a significant amount of time managing connections
    ➢ legacy services may have to run on environments poorly adapted for networking
    ➢ databases are performance hogs
Three-Tier Architecture

- **Client**
  - User interface
  - Some data validation and formatting

- **Middle tier**
  - Queueing and scheduling of user requests
  - Transaction processor
  - Connection management
  - Format conversion

- **Backend server**
  - Database
  - Legacy application processing
Peer-to-Peer (P2P)

- Cooperative interaction to perform distributed activity
  - Peer processes have equal status
  - No distinction between clients and servers
- Code in peer processes maintains consistency of application-level resources and synchronizes application-level actions when necessary
- No machines are dedicated to service others
  - Not strictly true
  - Some machines can have different roles (e.g., trackers)
- Examples:
  - File / Content sharing: BitTorrent, Gnutella, …
  - Chat programs
  - SETI@home computation
Distributed Application Based on Peer Processes

* Graphics from Distributed Systems: Concepts and Design, Couloris, Dollimore, and Kindberg
Distributed Peer-to-Peer Application

Pattern of communication depends on application requirements

Distributed state and distributed load

Sharable objects

Peers 5 .... N

Replication of data for reliability

Duplication of machines (peers) for performance

* Graphics from Distributed Systems: Concepts and Design, Couloris, Dollimore, and Kindberg
Process Peer Model

- What about idle or lightly loaded workstations?
- Sharing of computing resources (exploit resources)
  - Either let idle (lightly loaded) machines sit idle
  - Run jobs on unused computing resources
- Alternatively
  - Treat machines as a collection of CPUs and memory
  - Assign processes to run on resource on demand
  - Users won’t need heavy duty workstations locally
    - run GUI on local machine
    - use remote machine for heavy duty processing
- Plan 9 computation model (http://plan9.bell-labs.com)
Grid Computing

- Objective: make costly resources more accessible
- Provide users with seamless access to:
  - Storage capacity
  - Computing capacity
  - Network bandwidth between storage and computing
- Heterogeneous and geographically distributed
- On demand resource allocation
- Adaptive to variations in load and reliability
- Ubiquitous
- Popular for scientific computing (wax and wane)
Cloud Computing

- Transparent access to compute and storage resources
  - Many benefits: reliability, scalability, transparency
  - Metered cost model
  - Leverages hardware virtualization

- Driven by economic factors
  - Computing / Data center efficiencies
    - owner cost and power
  - User and administration productivity

- Example: Amazon’s Elastic Compute Cloud
  http://aws.amazon.com/ec2/
Design Requirements for Distributed Architectures

ië Performance issues

• Throughput
  - rate at which (computational) work is done
  - ability to perform work for all users (fairness)
  - combined workload and performance of all components
  - requirements: minimize bottlenecks

• Responsiveness
  - fast and consistent response to interaction
  - determined by performance of components along path
  - includes communications costs
  - requirements: few software layers and small data

• Balancing computational workload
  - requirement: reduce competitions for resources
Design Requirements (continued)

- Quality of service (QoS)
  - Non-functional properties (after function properties met)
    - reliability, security, performance
  - Adaptability to meet changing resource availability
  - Originally QoS reflected responsiveness and throughput
  - Redefined to reflect timeliness guarantees
    - time-critical has deadline requirements
    - batch has looser time requirements, but scheduled
  - QoS with respect to varying distributed system state

- Caching and replication
  - Access enhancement and validation

- Dependability
  - Fault tolerance and security
Design Requirements (continued)

- **Caching and replication**
  - Access enhancement and validation
  - Web caching
    - requirement of validation of version
    - requirement of detecting staleness (age)

- **Dependability**
  - Correctness
  - Fault tolerance
    - function correctly in the presence of faults (HW, SW, network)
    - reliability achieved through redundancy and state maintenance
  - Security
    - secure sensitive information to unauthorized access and attack
    - security services
Thinking about Design Requirements

- Requirements for a specific problem gives a basis for analyzing architecture appropriateness and suitability
- Consider quality of service for static web pages
  - They do not change frequently
  - Users want instant responses
  - What might you choose?
  - Replication?
  - Responsiveness?
  - Maybe proxies would be a good approach
  - When might it not be?
Fundamental Models

- Make assumptions about the system explicit
- Make generalizations about what is possible based on these assumptions
- Generalizations can be in the form of:
  - Algorithms
  - Formal proofs
  - Descriptions in a formal logical framework
- Generalizations are useful because they facilitate formal analysis to draw conclusions about:
  - Correctness, performance, security
- Capture aspects with respect to interactions, failures, and security
Fundamental models

- Interaction model
  - Performance of communication channels
  - Computer clocks and timing events
  - Two variants:
    - synchronous distributed systems
      - time to execute each step has known lower/upper bounds
      - each message transmitted is received in bounded time
      - each process has local clock with known relation to real
    - asynchronous distributed systems
      - no bounds on process execution speeds
      - no bounds on message transmission delays
      - no bounds on clock drift rates
Real-Time Ordering of Events

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
Processes and Channels

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
# Omission and Arbitrary Failures

<table>
<thead>
<tr>
<th>Class of failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail-stop</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may detect state.</td>
</tr>
<tr>
<td>Crash</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may not detect.</td>
</tr>
<tr>
<td>Omission</td>
<td>Channel</td>
<td>A message inserted in an outgoing buffer never arrives.</td>
</tr>
<tr>
<td>Send omission</td>
<td>Process</td>
<td>A process completes a send, but message not put in buffer.</td>
</tr>
<tr>
<td>Receive omission</td>
<td>Process</td>
<td>A message is put in incoming buffer, but never received</td>
</tr>
<tr>
<td>Arbitrary (Byzantine)</td>
<td>Process or Channel</td>
<td>Process/channel exhibits arbitrary behavior.</td>
</tr>
</tbody>
</table>

## Timing Failures

<table>
<thead>
<tr>
<th>Class of failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Process</td>
<td>Process’s local clock exceeds the bounds on its rate of drift from real time.</td>
</tr>
<tr>
<td>Performance</td>
<td>Process</td>
<td>Process exceeds the bounds on the interval between two steps.</td>
</tr>
<tr>
<td>Performance</td>
<td>Channel</td>
<td>A message’s transmission takes longer than the stated bound.</td>
</tr>
</tbody>
</table>
Objects and Principals

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg
The Enemy

* Graphics from Distributed Systems: Concepts and Design, Couloris, Dollimore, and Kindberg
Secure Channels

* Graphics from Distributed Systems: Concepts and Design, Coulouris, Dollimore, and Kindberg