Last time

- Broad overview of what a distributed system is
  - A set of computers connected by a network.
  - Servers provide services, clients use services.

- Design of distributed system software seeks to provide transparency for different aspects of the system
  - Access, location, failure, etc…

- Architectural models define the placement of the pieces of the system
  - Where parts are (hardware/software), and how they are connected (both physically and logically).
A software excursion

- Before continuing with architectural models, let’s briefly describe how software layering helps support the models.

- Middleware and layered software.
Software Layers

- **Software architectures** refers to the structuring of software
  - Layers and services (“service layers”).
  - We will see an instance of this soon with the networking middleware.

- **Platform**
  - Lowest-level hardware and software layers (e.g.: OS).

- **Middleware**
  - Layer of software that provides abstraction above potential heterogeneity via a convenient programming model.
  - Building blocks for building software.
  - Raises the level of communication activities through communication abstractions and mechanisms.
  - Makes distributed nature of system transparent.
Software and Hardware Layers

- Applications, services
- Middleware
- Operating system
- Computer and network hardware

Platform
Common middleware packages

- Remote procedure call (RPC)
- Group communication (Isis)
- Object-oriented
  - CORBA: Common Object Request Broker Architecture
  - Java RMI: Remote Method Invocation
  - Microsoft DCOM: Distributed Common Object Model
- Packages provide higher-level application services
  - Naming, security, transactions
  - Persistent storage, event notification
Middleware limitations

- End-to-end argument (Saltzer, Reed, Clarke, 1984)
  - Some communications-related functions can be completely and reliably implemented only with the knowledge and participation 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
End-to-end argument

- Can we think of any examples illustrating the end-to-end argument?
Functional View of Middleware

- **Information exchange services**
  - Message passing

- **Application-specific services**
  - Specialized services
    - Example: Transaction, replication services for distributed DB.
    - Example: Groupware services for collaborative applications.

- **Management and support services**
  - Name services and registries for locating distributed resources dynamically.
  - Administration of resources distributed over a network.
  - Monitoring performance and behavior of distributed set of resources.
Production Middleware

- These you will encounter in the real world.

- Single-service components
  - HTTP for retrieving documents remotely
  - Sun RPC for remote procedure call
  - SSL for secure socket layer

- Integrated middleware environments
  - Integrates multiple components into a single coherent package.
  - Examples: CORBA, DCOM, .NET, Java
Objectives

- Resume discussion of models
  - Talk a bit more about architectural models
  - Talk about fundamental models

- Begin our discussion of network technologies
  - Today: a birds-eye view of the issues and general concepts.
  - Next week: sockets and RMI to give you exposure to specific network technologies.
  - After next week, you will be given your first programming assignment for hands on work with RMI.
Let’s revisit architectural models and talk further about those beyond simple client/server.
Architectural model

- Recall:
  - The architectural model is concerned with:
    - What are the components of the system?
    - What are their roles?
    - Where are they located?

- From this, we can break distributed systems up into some coarse and familiar classes:
  - Client/server
  - Peer-to-peer
  - Multiple servers
  - Proxy servers
  - Etc…
Proxy Servers and Caches

- Proxy servers are 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 checked first when data requested.
  - If present in cache, data provided from there.
  - Otherwise, fetched from actual server.
  - Cache must provide facility to check if page is up-to-date, especially if server is unaware of presence of cache.
Example: Web Proxy Server
Peer-to-peer

- Unlike client/server, peer-to-peer is based on a set of processes with equal status in the distributed system who cooperatively perform some sequence of operations to achieve a goal.
  - No distinction between clients and servers.
- Code in peer processes maintains consistency of application-level resources and synchronizes application-level actions when necessary.
- No dedicated machines.
  - This may not be strictly true, especially considering popular P2P services like BitTorrent with trackers. These aid in finding peers initially though, and do not necessarily participate directly in the P2P activity.
- *Examples*: BitTorrent, Gnutella, “Bonjour” chat programs, file sharing.
Peer-based distributed application
Process Peer Model

- What about idle or lightly loaded workstations?
- Sharing of computing resources
  - Either let idle machine sit idle
  - Or run useful jobs on unused computing resources
- Alternatively
  - Treat machines as collection of CPUs and memory
  - Assign processes to run on resource on demand
  - Users won’t need heavy duty workstations locally
    - GUI locally
    - Remote machine for heavy processing
  - Computational model of Plan9
Thin clients

- Systems that run code remotely but provide user interactivity locally.
- X-terminals were a form of this. The thin client ran an X server, and displayed programs running remotely.
- We see this today in systems like remote desktop services such as VNC.
  - Examples: Apple Remote Desktop, “GotoMyPC.com”.
  - Today we see systems where the web browser is the thin client, with the app running elsewhere but displayed in an applet or service running in the browser.
Grid computing

- Addresses the issue of making costly resources available to a wide user base.
- Provide users seamless access to:
  - Storage capacity
  - Compute capacity
  - Network bandwidth between storage and computing
- Growing in popularity for scientific computing.
- On demand resource allocation
- Adaptive to variations in load and reliability
Cloud computing

- Transparent access for users to compute and storage resources for a metered cost.
  - Virtualization: Multiplex hardware resources transparently to multiple users.
  - Reliability: Through redundancy on cloud provider end.
    - (Outages may occur though)
  - Performance can be tuned dynamically.
  - Provider can add resources to the cloud to scale it.
  - Transparent access via network technology.
- Primarily driven by economic factors. You buy the cycles/space that you need, and the leftovers are used by others instead of wasted. Your cloud usage can grow as your load changes.
  - Example: Amazon cloud services.
A potential application area for a project would be to target a cloud system.

- Amazon
  - I think students can apply for free allocations.
- Download and install services to set up a local cloud.
Variations on Client/Server Model

- Mobile code moves between computer systems.
- Applets are a well-known and widely used example.
  - Example: Java applets, Flash apps.
- Helps to achieve better performance
  - Eliminates some delays and variability due to 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
Two-tier architecture

Common from 1980s to 1990s

UI on user desktop

Application services on server

Example: old text-based business applications. Does anyone remember those old telnet-based apps to login to a VMS system with text menus and interface?

Performance deteriorates with large user communities

Server can get overloaded managing connections.

Legacy services may end up running in environments poorly adapted for networking.

Databases are performance hogs.
Multi-Tier Client/Server Architectures

- Three-tiered architectures address issues with two-tier.
  - **Client**
    - User interface
    - Some data validation and formatting
  - **Middle tier**
    - Queuing and scheduling of user requests
    - Transaction processing
    - Connection management
    - Format conversion
  - **Backend server**
    - Database
    - Legacy application processing/interface
Architectures

- We have briefly seen the major, common system architectures that you will encounter in practice.

- So, when one is designing a distributed system to address some problem or provide a service, how do we determine which architectural model or pattern is best?

- **Analyze requirements that will drive the design.**
Design requirements

- **Performance**
  - Responsiveness (especially for interactive services)
  - Throughput
  - Load balance

- **Quality of service**
  - Reliability
  - Security
  - Performance
  - Adaptability to changes in resource availability.

- **Caching and replication**

- **Dependability**
  - Fault tolerance
  - Security
Design requirements

- Looking at the requirements of a specific problem, you can see how some architectures are more appropriate than others.

- Example case: Quality of service is required for serving up static web pages. They will not change frequently, and users want almost instant responses.
  - Infrequent changes: Replication is easy, as is caching in proxies closer to users.
  - Quality of service: Need fast response, so we want information close to users. Proxies again.

- So, sounds like a proxy server mirroring one or more servers is the right choice.
Fundamental models

- Make assumptions about the system explicit

- Make generalizations about what is possible given these assumptions.

- These generalizations can be in the form of
  - Algorithms
  - Formal proofs
  - Descriptions in a formal logical framework

- The generalizations are useful because they facilitate formal analysis to draw conclusions about:
  - Correctness
  - Performance
  - Security
We wish to capture the following aspects of distributed systems in fundamental models:

- Interactions in the presence of delays that occur in the real world.
- Failures
- Security