

Agenda

- **Last time**
 - Intro: milestones of interest, overview of class
- **This time**
 - System models (chpt 1)
- **Next time (Thurs)**
 - More system models (chpt 1 + 2), networks (chpt 3)s
- **No class Tuesday Jan 30 (Marty at conference)**
 - Will be made up Thurs Feb 8 / Fri Feb 9

CS451: Distributed Systems (Spring 2007)

Before we start

- **Concepts we will look at (with help from the textbook):**
 - Sys models, networks, IPC, dis objects, security, DFS, Name services, time + global states, coordination and agreement, transactions, P2P
- **How do we make-up classes? Thurs night AND/OR Friday afternoon?**
 - How about: Thurs night 7-8:15 AND Fri 3-4:15 ??
- **Office hours:**
 - How about Mon 2-4pm

CS451: Distributed Systems (Spring 2007)

Distributed System Definitions

- "A distributed system is a collection of **independent** computers that **appear to the users** of the system as a single computer." [Tanenbaum]
- "A distributed system, consists of discrete software agents that **must work together** to implement some intended functionality." [Web Services Architecture, W3C Working Draft 8 August 2003; <http://www.w3.org/TR/ws-arch/>]
- A system in which components located at **networked computers** communicate and coordinate their actions only by **passing messages**." (Coulouris et al. 2001 – our textbook)

CS451: Distributed Systems (Spring 2007)

Consequences of "Dis sys" definition

- **Concurrency**
 - Autonomous concurrent processes
 - A and B are concurrent if either A can happen before B, or B can happen before A
 - Synchronization and coordination by message passing
 - Enables resource sharing between users
 - Data, Services, Devices
 - Typical problems of concurrent systems
 - Deadlocks
 - Unreliable communication
- **No global clock**
 - "no shared, single idea of precisely "when" something occurred"
 - Coordination by asynchronous message passing
 - Limits precision of clock synchronization
- **No global state**
 - There is no single process in the distributed system that would have a knowledge of the current global state of the system
 - Due to concurrency and message passing communication

CS451: Distributed Systems (Spring 2007)

Consequences of Definition (cont.)

- **Independent failures**
 - Processes run autonomously, in isolation
 - Failures of individual processes may remain undetected
 - Individual processes may be unaware of failures in the system context
 - Failures more common than in a centralized system
 - New ways of failing
 - Network failures isolates processes and partitions the system

CS451: Distributed Systems (Spring 2007)

Why Distributed Systems?

- **Functional distribution: computers have different functional capabilities**
 - Client / server
 - Host / terminal
 - Data gathering / data processing
 - Sharing of resources with specific functionalities
- **Inherent distribution stemming from the application domain, e.g.**
 - Cash register and inventory systems for supermarket chains
 - Devices in the environment (e.g. "sensor nets")

CS451: Distributed Systems (Spring 2007)

Why Distributed Systems? (cont.)

- **Load distribution/balancing:**
 - assign tasks to processors such that the overall system performance is optimized.
- **Replication of processing power: independent processors working on the same task.**
 - Distributed systems consisting of collections of microcomputers may have processing powers that no supercomputers will ever achieve
 - 10000 CPUs, each running at 50 MIPS, yields 500000 MIPS -> instruction to be executed in 0.002 nsec -> equivalent to light distance of 0.6 mm -> any processor chip of that size would melt immediately

CS451: Distributed Systems (Spring 2007)

Why Distributed Systems? (cont.)

- **Physical separation: systems rely on the fact that computers are physically separated (e.g., to satisfy reliability requirements, disasters).**
- **Economics: collections of microprocessors offer a better price/performance ratio than large mainframes**
 - Mainframes: 10 times faster, 1000 times as expensive

CS451: Distributed Systems (Spring 2007)

Distributed System vs. Parallel System

- **Generally dependent:**
 - The *coupling* between the systems ("tight" or "loose")
 - The physical proximity
 - The programming model (e.g., threads not generally a model for distributed systems)

CS451: Distributed Systems (Spring 2007)

Examples of Distributed Systems

- The Internet/ WWW
- Intranets (e.g., "enterprise")
 - Firewalled, perhaps
- Wireless Information Devices
 - Mobile and ubiquitous computing; your fridge
- Distributed Multimedia-Systems
- Volvo S80
- Clusters (e.g., centurion) – perhaps
- Server farms
- Company LANs
- etc.

CS451: Distributed Systems (Spring 2007)

Challenges for Distributed Systems

- Application partitioning/distribution
- Heterogeneity
- Openness
- Security
- Scalability
- Failure handling/reliability
- Concurrency
- Transparency
- Management

CS451: Distributed Systems (Spring 2007)

Challenge: Application Partitioning/Distribution

- **Partitioning**
 - Dividing application into units of distribution
- **Configuration**
 - Associating units with each other
- **Allocation**
 - Binding/downloading modules on target system
 - Distributing load among processing elements (statically or dynamically)

CS451: Distributed Systems (Spring 2007)

Challenge: Heterogeneity

- **Heterogeneity** = variety and difference
- **Heterogeneity of**
 - underlying network infrastructure,
 - computer hardware and software (e.g., operating systems, compare UNIX sockets and Winsock calls),
 - programming languages (in particular, data representations).
- **Heterogeneity needs to be masked in most cases**
- **Interfaces and implementation may differ**
 - Underlying protocol stays the same
 - E.g., setting up a socket in Windows/Linux
 - System/library function calls differ
 - Both use the IP protocol
- **Standards are required (e.g., IETF, W3C, OASIS, etc.)**

CS451: Distributed Systems (Spring 2007)

Challenge: Heterogeneity (solution: middleware)

- **Middleware: a masking/abstracting software layer**
 - Allows heterogeneous nodes to communicate
 - Uniform computational model
 - Supports one or more programming languages
 - Provides support for distributed applications
 - Remote object invocation
 - Remote SQL access
 - Distributed transaction processing
- **Examples: CORBA, Java RMI, Microsoft DCOM (?), Globus, etc.**

CS451: Distributed Systems (Spring 2007)

Challenge: Heterogeneity (solution: VMs)

- **Mobile code: code designed to migrate between computers**
 - Needs to overcome heterogeneity in hardware (instruction sets)
- **Virtual machine can mask heterogeneity**
 - Compilers create byte code for VM
 - VM implemented for every type of hardware
 - E.g., JVM, .NET CLR
- **Alternative: Brute force**
 - Port code for every platform in the system
 - E.g., object is compiled for each instruction set

CS451: Distributed Systems (Spring 2007)

Challenge: Openness

- **How easily can third-party clients/services be added?**
- **Ensures extensibility and maintainability**
- **Possibility of re-implementation**
- **Ease of adding new resource-sharing devices**
- **Important factors:**
 - Specification
 - Documentation
 - Published interfaces (often bypassing standards organizations)
 - Conformance testing and verification of all devices

CS451: Distributed Systems (Spring 2007)

Challenge: Security

- **Three components:**
 - Confidentiality (protection against disclosure to unauthorized individuals)
 - Integrity (protection against alteration or corruption)
 - Availability (protection against interference with the means to access the resources)
- **The challenge: sending sensitive information in a network message in a secure manner efficiently**
 - Solution: crypto
- **Many Unsolved problems:**
 - Denial of service attacks
 - Disruption of a service by bombardment of messages
 - Security of mobile code
 - Unpredictable effects
 - Trojan horse behavior (need not be a virus)

CS451: Distributed Systems (Spring 2007)

Challenge: Scalability

- **Informal: "A distributed system is scalable if it remains effective as the number of users and/or resources increases"**
- **Challenges:**
 - Controlling resource costs
 - Controlling performance loss
 - Preventing resources from running out
 - Avoiding performance bottlenecks

CS451: Distributed Systems (Spring 2007)

Challenge: Scalability

- **Cost of physical resources**
 - Grows as the number of users increases
 - Guideline: Should not grow faster than $O(n)$, where n = number of users
- **Performance loss**
 - Increases with growing data sets (number of users)
 - Hierarchic structures have better search times than linear structures
 - Guideline: Search time should not grow faster than $O(\log n)$, where n = size of the data

CS451: Distributed Systems (Spring 2007)

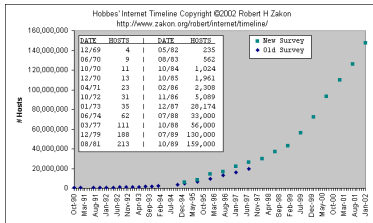
Challenge: Scalability

- **Limits on software resources**
 - E.g., IPv3 numbers (32 bits)
 - "640K should be enough for everyone"
 - Difficult to predict
 - Adjusting better than overcompensating?
 - Large IP addresses increase storage demands
- **Performance bottlenecks**
 - Some resources accessed frequently
 - Decentralize algorithms
 - E.g., Domain Name Service
 - Replication & caching

CS451: Distributed Systems (Spring 2007)

Challenge: Scalability

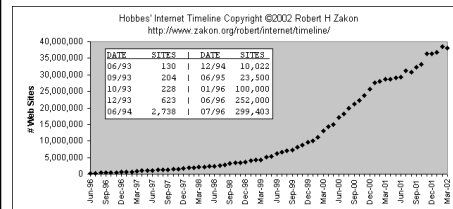
- **Internet hosts**
 - **Host:** A computer system with registered IP address
 - Exponential growth



CS451: Distributed Systems (Spring 2007)

Challenge: Scalability

- **World Wide Web**
 - Exponential growth or is it?



CS451: Distributed Systems (Spring 2007)

Challenge: Failure Handling

- **Detection, masking/hiding, tolerance, recovery, redundancy**
- **Detection (may be impossible)**
 - Some possible (e.g., using transmission errors via checksums)
 - Some impossible (crashed remote server vs. slow remote server)
 - Challenge: manage failures that cannot be detected, but suspected
- **Masking/hiding**
 - Some failures can be hidden or made less severe
 - Replication in space/time
 - Space: e.g., writing to multiple disks
 - Time: e.g., transmission of multiple messages
 - May not work in worst cases, e.g., all disks may have been corrupted
 - Not always ideal
 - E.g., bounds needed for real-time systems

CS451: Distributed Systems (Spring 2007)

Challenge: Failure Handling

- **Tolerance**
 - Sometimes not feasible to hide all failures
 - E.g., user has to tolerate if web service has failed rather than wait until service is up again
 - Only feasible for certain classes of applications/systems, e.g., DNS vs. Internet addresses
- **Recovery**
 - Restoring a correct system state
 - Roll back using log files
- **Redundancy**
 - Tolerate failures by using redundant components (provided through replication)
 - Can exist on many levels (Hardware, software and design)
 - E.g., redundant routes in network, replication of name tables in multiple domain name servers
- **Goal of failure handling: high availability**

CS451: Distributed Systems (Spring 2007)

Challenge: Transparency

- **Concealing the heterogeneous and distributed nature of the system so that it appears to the user like one system**
- **Eight types (ANSA/ISO)**
- **Access transparency:** enables local and remote resources to be accessed using **identical operations**. E.g., mapped network drives
- **Location transparency:** enables resources to be accessed without knowledge of their location. E.g., URL and e-mail addresses)
- **Concurrency transparency:** enables several processes to operate concurrently using shared resources without interference between them.
- **Replication transparency:** enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
- **Failure transparency:** enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components. E.g., retransmission of e-mail messages
- **Mobility transparency:** allows the **movement** of resources and clients within a system without affecting the operation of users or programs.
- **Performance transparency:** allows the system to be reconfigured to improve performance as loads vary.
- **Scaling transparency:** allows the system and applications to expand in scale without change to the system structure or the application algorithms.

CS451: Distributed Systems (Spring 2007)

Challenge: Transparency

- **[Arguably] Most important for distributed systems: Access and location transparency**
- Together referred to as **network transparency**
- Most strongly affects the utilization of distributed resources
- E.g., file directory
 - user does not know whether file is remote or local

CS451: Distributed Systems (Spring 2007)

Challenge: Management

- **Distributed resources have no central point for management**
- **Locally optimized may not be globally optimized**
 - Not possible in every case: the system might not be owned by one organization

CS451: Distributed Systems (Spring 2007)

Chapter 2: Distributed System Models

- **Architectural models**
 - How do components of the system interact?
 - How are these components map onto the underlying network of computers?
- **Fundamental models**
 - Formal description of system properties common in all architectural models
 - Interaction, failure, security models
 - Appear throughout the course, but not discussed in detail

CS451: Distributed Systems (Spring 2007)

Architectural Models

- **Architecture:** structure of separately specified components
- Overall goal: Structure should meet present and future demands on reliability, manageability, adaptability, and cost-effectiveness
- Functions of individual components not interesting
 - Abstracted away
- **Consider instead:**
 - **Placement (across network)**
 - **Patterns for data/workload distribution**
 - **Interrelationships (Functional roles, communication patterns)**

CS451: Distributed Systems (Spring 2007)

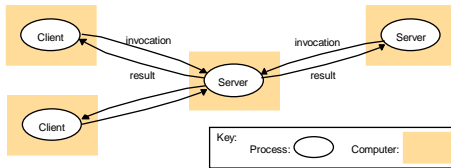
Architectural Models

- **Client/server vs. peer processes**
- **Architectural models can be used to determine placement**
- **More dynamic systems can be built as variations on the client-server model**
 - Moving code
 - Adding/removing nodes
 - Discovery and advertisement of services

CS451: Distributed Systems (Spring 2007)

System Architectures: Client-Server Model

- The most widely used



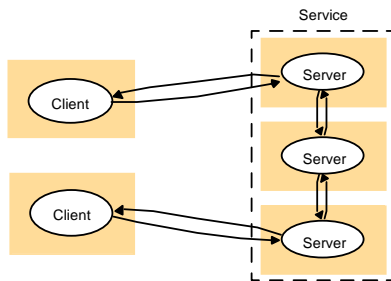
CS451: Distributed Systems (Spring 2007)

System Architectures: Client-Server Model

- Client: Process wishing to access data, use resources or perform operations on a different computer
- Server: Process managing data and all other shared resources amongst servers and clients, allows clients access to resource and performs computation
- Interaction: invocation / result message pairs
- Example
 - http server: client (browser) requests page, server delivers page

CS451: Distributed Systems (Spring 2007)

System Architectures: Multiple Servers Model



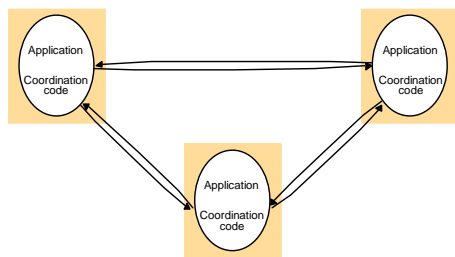
CS451: Distributed Systems (Spring 2007)

System Architectures: Multiple Servers Model

- Services may be provided by multiple servers
- Partitioned or replicated service-related objects
- Replication provides
 - Increased performance, availability and fault-tolerance
- But requires replica coordination / consistency preservation
- E.g. high availability web servers (portals, download centers), information services
- Servers maintain either replicated or distributed database

CS451: Distributed Systems (Spring 2007)

System Architectures: Peer-to-Peer Model



CS451: Distributed Systems (Spring 2007)

System Architectures: Peer-to-Peer Model

- Peer processes: processes that play similar roles
 - No absolute distinction between client/server
 - May still assume client/server roles from time to time
- Reduces inter-process communication delay for local object access
- Increased fault-tolerance and scalability
- Coordination difficult
- E.g. distributed search, routing, distributed computing, news servers (and of course MP3s)

CS451: Distributed Systems (Spring 2007)