CSCI 5105
Instructor: Abhishek Chandra

Today
- Distributed Systems Hardware Classification
- Distributed Architectures

Distributed Systems: Hardware Classification
- Multiprocessor system
- Homogeneous Multicomputer system
- Heterogeneous Multicomputer system

Multiprocessor Systems
- Hardware features
  - Memory: Typically shared with private cache
  - Interconnect: could be shared (bus-based) or switched
  - Tightly-coupled system and homogeneous
### Homogeneous Multicomputer Systems
- Hardware features:
  - Have private memory
  - Connected by LAN or an interconnect
  - Homogeneous hardware/OS
  - Less tightly-coupled and homogeneous
- Example: Compute Cluster

### Heterogeneous Multicomputer Systems
- Hardware features:
  - Each machine is autonomous
  - Has own hardware/OS
  - Connection through the Internet
  - Loosely-coupled system and heterogeneous
- Examples: Web, Grid

### Distributed System Architecture
- A distributed application runs across multiple machines
- How to organize the various pieces of the application?
  - Where is the user interface, computation, data?
  - How do different pieces interact with each other?

### Architectures
- Centralized: Most functionality is in a single machine
- Decentralized: Functionality is spread across symmetrical machines
- Hybrid: Combination of the two
Centralized Architecture
- Each component is responsible for different part of the application functionality
- Core functionality is in a single location

Client-Server Architecture
- Application is vertically distributed
  - Distribution along functionality
  - Logically different component at different place
  - E.g.: UI at client, computation & data at server

Component Distribution
- Could have variations on component distribution
  - Different amount of functionality between client-server
  - Only UI at client
  - UI+partial processing at client
  - UI+processing at client, data at server

Multi-tiered Servers
- Server may not be a single machine
- Multi-tiered architecture:
  - Front-end
  - Application server
  - Database
Decentralized Architecture

- Horizontal distribution of application
  - Each component is identical in functionality
  - Differ in the portion of data/state they operate on
  - E.g.: File-sharing, parallel processing

Server Clusters

- Replication of functionality across machines
  - Multiple front-ends, app servers, databases
  - Client requests are distributed among the servers
    - Load balancing
    - Content-aware forwarding

Peer-to-Peer Systems

- Each component is symmetric in functionality
  - Servent: Combination of server-client
  - How does a node find the other?
    - No “well-known” centralized server

Overlay Network

- A logical network consisting of participant components (processes/machines)
  - Built on top of physical network
  - Can be thought of as a graph
    - Nodes are processes/machines, links are communication channels (e.g., TCP connections)
Types of P2P Systems

- **Unstructured**: Built in a random manner
  - Each node can end up with any sets of neighbors, any part of application data
  - E.g.: Gnutella, Kazaa
- **Structured**: Built in a deterministic manner
  - Each node has well-defined set of neighbors, handles specific part of application data
  - E.g.: CAN, Chord, Pastry

Unstructured P2P Architectures

- Each node has a list of neighbors to which it is connected
- Communication to other nodes in the network happens through neighbors
- Neighbors are discovered in a random manner
- Exchange information with other nodes to maintain neighbor lists
- Application data is randomly spread across the nodes
- Flooding: To search for a specific item

Structured P2P Architectures

- Nodes and data are organized deterministically
- Distributed Hash Tables (DHT)
  - Each node has a well-defined ID
  - Each data item also has a key
  - A data item resides in the node with nearest key
- Each node has information about neighbors in the ID space
- Searching for a data item:
  - Routing through the DHT overlay network

Hierarchical Architecture

- Tree of nodes
- Centralized architecture between parent and children
- More scalable than a centralized architecture
  - Each node handles only part of the network
  - E.g.: DNS
**SuperPeers**
- Special peers that maintain an index
  - Of other peers
  - Of data items and their location
- Need for superpeers:
  - Efficient search: Avoid flooding
  - Location-awareness: Find "nearest" neighbors
  - Easy Join: Node can easily find a starting peer

**Hybrid Architecture**
- Combination of centralized and distributed architectures
  - Some parts of the system organized as client-servers
  - Other parts organized in decentralized manner

**Content Distribution Networks (CDNs)**
- Provide localized content to users
  - Decentralized set of content servers, may have P2P relationship
  - Client-Server relation to the users
  - E.g.: Akamai

**Collaborative Distributed Systems**
- Work by user collaboration
  - P2P in functionality
  - Starting up is done in a client-server manner
  - E.g.: Bittorrent, Napster
Centralized vs. Decentralized Control

- Decisions can be made in a centralized or decentralized manner
- Centralized: A single server maintains control, metadata, etc.
- Decentralized: All machines act autonomously, may exchange information

Architectural Styles

- How to implement a distributed application
  - How software components are organized and communicate with each other
- Component: Module with a well-defined interface
  - Would implement some part of the application functionality
- Connector: Communication mechanism
  - Will enable components to talk and coordinate

Architectural Styles

- Layered: Components are placed in multiple layers
  - Each layer interacts with those above and below
- Object-based: Each component is an object
  - Communication via (remote) method invocations
- Data-centered: Communication via a data repository
  - File system, Web pages, etc.
- Event-based: Communication through events
  - Pub-sub systems

Middleware

- A distributed layer between applications and low-level OS
  - Provides core functionality and services
  - Applications can use these for higher-level functionality
  - May rely on per-machine OS/software support