DISTRIBUTED COMPUTER SYSTEMS

ARCHITECTURES

Dr. Jack Lange
Computer Science Department
University of Pittsburgh

Fall 2015

Outline

- System Architectural Design Issues
  - Centralized Architectures
    - Application Layering and Multitiered Architecture
  - Decentralized Architectures
    - Vertical Distribution
    - Horizontal Distribution
- Client-Server Model
- Peer-to-Peer Model
Definitions

- **Software Architectures** – Describes the organization and interaction of software components
  - Focus is on logical organization of software – component interaction, etc.
- **System Architectures** - Describes the placement of software components on physical machines
- Realization of an architecture can be achieved in different ways
  - **Centralized** – most components are located on a single machine
  - **Decentralized** – most machines have approximately the same functionality,
  - **Hybrid** – a combination of both.

Architectural Styles

- An architectural style describes a particular way to configure a collection of components and connectors.
  - **Component** - a module with well-defined interfaces; reusable, replaceable
  - **Connector** – communication link between modules
- Architectures suitable for distributed systems:
  - **Layered** architectures
  - **Object-based** architectures
  - **Data-centered** architectures
  - **Event-based** architectures
Architectural Styles – Layered and Object-Based

Object based is less structured
Component = object
Connector = RPC Or RMI

Layered

Object-Based

Data-Centered Architectures

- Main purpose is data access and update
- Processes interact by reading and modifying data in some shared repository
  - Repository can be active or passive
    - Traditional data base – Passive repository responds to requests
    - Blackboard system – Active repository, where clients solve problems collaboratively and system updates clients when information changes.
- Web-based distributed systems are largely data centric
  - Processes communicate through shared Web-based data services
**Architectural Styles – Event-Based**

- Communication is via event propagation
- Often associated with Publish/Subscribe systems – register interest in market information and receive email updates
- Referential Decoupling
  - Loosely couples sender and receiver – space decoupling

Event-based arch. supports several communication styles:
- Publish-subscribe
- Broadcast
- Point-to-point

**Combined Event and Data-Centered Architectures**

- Shared Data Spaces combines event-based and data-centered architectures
  - In addition to space decoupling, processes are also decoupled in time
    - Processes need not be both active when communicating with each other
    - Data access can be achieved using description rather than explicit reference
Shared Data Space

System Architectures

- **Centralized** – Traditional client-server structure
  - Vertical or hierarchical organization of communication and control paths, as in layered software architectures
  - Logical separation of functions into clients and servers

- **Decentralized** – Peer-to-Peer structure
  - Horizontal rather than hierarchical communication and control
  - Communication paths are less structured; symmetric functionality

- **Hybrid**: combine elements of C/S and P2P
  - Edge-server systems
  - Collaborative distributed systems.

Classification of a system as centralized or decentralized refers to communication and control organization, primarily.
Centralized vs Decentralized Architectures

- **Vertical Distribution** – Traditional client-server architectures exhibit vertical distribution.
  - Each level serves a different purpose in the system.
  - Logically different components reside on different nodes

- **Horizontal distribution** – e.g., P2P architectures
  - Each node has roughly the same processing capabilities and stores and manages part of the total system data.
  - Better load balancing, more resistant to denial-of-service attacks, but harder to manage than C/S
  - Communication and control is not hierarchical, all nodes are peers with equal functionalities

System Architecture

**CLIENT-SERVER ARCHITECTURE**
**Traditional Client-Server**

- Processes are divided into two groups: Clients and Servers
- Synchronous communication
  - Request-reply protocol
- In LANs, often implemented with a connectionless protocol, typically unreliable protocols such as UDP
- In WANs, communication is typically connection-oriented TCP/IP, reliable
  - High likelihood of communication failures

---

**C/S Architectures**

Client and Server General Interaction Model
Transmission Failures

- With connectionless transmissions, failure of any sort means no reply
  - Request message was lost
  - Reply message was lost
  - Server failed either before, during or after performing the service
- Can the client tell which of the above errors took place?

Idempotency

- Retransmission, after timeout, is the typical response to lost request in connectionless communication
- Consider effect of re-sending a message such as “Increment X by 1000”
  - If first message was acted on, now the operation has been performed twice
- **Idempotent** operations can be performed multiple times without harm
  - “Return current value of X” and “Check on availability of a product” are idempotent
  - “Increment X”, “Order Product Y” are **non-idempotent**
A clear issue regarding the Client/Server model is how to draw a **distinction** between a client and a server

- Although still controversial, client-server model follows a layered architectural style.

**Layered “software” Architecture for Client-Server Systems**

- **User-interface level**: GUI’s, usually for interacting with end users
- **Processing level**: data processing applications – the core functionality
- **Data level**: interacts with database or file system
  - Data usually is **persistent**, and exists for next use even if no client is accessing it
  - In its simplest form, a data level is File System
  - It is more common to use a full-fledged database system
Examples

- Web search engine
  - Interface: type in a keyword string
  - Processing level: processes to generate DB queries, rank replies, format response
  - Data level: database of web pages
- Stock broker's decision support system
  - Interface: likely more complex than simple search
  - Processing: programs to analyze data; rely on statistics, AI perhaps, may require large simulations
  - Data level: DB of financial information
- Desktop “office suites”
  - Interface: access to various documents, data,
  - Processing: word processing, database queries, spreadsheets,…
  - Data: file systems and/or databases

Application Layering

Internet Search Engine – Simplified Organization Into Three Different Layers
System Architecture

- Distinction of Client/Server into three logical levels, leads to a number of possibilities for physically distributing Client/Server functionality across multiple machines
  - Performance, robustness and easy of management are important factors

System Architecture

- Mapping the software architecture to system hardware
  - Correspondence between logical software modules and actual computers
- Multi-tiered architectures
  - Layer and tier are roughly equivalent terms, but layer typically implies software and tier is more likely to refer to hardware.
  - Two-tier and three-tier are the most common
Two-tiered Client/Server Architectures

- **Thin-Client Architecture** – Server provides processing and data management and client provides simple graphical display
  - Perceived performance loss at client
  - Easier to manage, more reliable, client machines don’t need to be so large and powerful

- **Fat-Client Architecture** – At the other extreme, all application processing and some data resides at the client
  - Pro – Reduces work load at server; more scalable
  - Con – Harder to manage, and potentially less secure

Multitiered Architectures

Alternative Client-server Organizations
Three-tiered Architectures

- In some applications servers may also need to be clients, leading to a three level architecture
  - Distributed transaction processing
  - Web servers that interact with database servers
  - Distribute functionality across three levels of machines instead of two.
System Architecture

DECENTRALIZED ARCHITECTURE

Peer-to-Peer

- Nodes act as both client and server; interaction is symmetric
- Each node acts as a server for part of the total system data
- **Overlay networks** connect nodes in the P2P system
  - Nodes in the overlay use their own addressing system for storing and retrieving data in the system
  - Nodes can route requests to locations that may not be known by the requester.
Overlay Networks

- ONs are logical or virtual networks, built on top of a physical network
  - A link between two nodes in the overlay may consist of several physical links.
  - Messages in the overlay are sent to logical addresses, not physical (IP) addresses
  - Various approaches used to resolve logical addresses to physical.

Circles represent nodes in the network.
- Blue nodes are also part of the overlay network.
- Dotted lines represent virtual links.
- Actual routing is based on TCP/IP protocols.
Overlay Networks

- Each node in a P2P system knows how to contact several other nodes.
- The overlay network may be:
  - **Structured** – Nodes and content are connected according to some design that simplifies later lookups, or
  - **Unstructured** – Content is assigned to nodes without regard to the network topology

Decentralized Structured Architecture

**DHT Peer-To-Peer Network**
Structured P2P Architectures

- A common approach is to use a Distributed Hash Table (DHT) to organize the nodes.

- Traditional hash functions convert a key to a hash value, which can be used as an index into a hash table.
  - Keys are unique – Each represents an object to store in the table.
  - The hash function value is used to insert an object in the hash table and to retrieve it.

Structured P2P Architectures

- In a DHT, data objects and nodes are each assigned a key which hashes to a random number from a very large identifier space.
  - This is necessary to ensure uniqueness.

- A mapping function assigns objects to nodes, based on the hash function value.

- A lookup, also based on hash function value, returns the network address of the node that stores the requested object.
DHT Characteristics

- Scalable – to thousands, even millions of network nodes
  - Search time increases more slowly than size
    - Usually $O(\log(N))$
- Fault tolerant – able to re-organize itself when nodes fail
- Decentralized – no central coordinator
  - Decentralized algorithms

Chord Routing Algorithm
Structured P2P

- Nodes are logically arranged in a circle
- Nodes and data items have m-bit identifiers (keys) from a $2^m$ namespace.
  - For example, a node’s key is a hash of its IP address and a file’s key might be the hash of its name or of its content or other unique key.
  - The hash function is consistent – As a result, keys are distributed evenly across the nodes, with high probability.
Inserting Items in the DHT

- A data item with key value \( k \) is mapped to the node with the smallest identifier \( id \) such that \( id \geq k \pmod{2^m} \)
- This node is the successor of \( k \), or \( \text{succ}(k) \)
- Modular arithmetic is used

Structured Peer-to-Peer Architectures

The connections between nodes are logical connections, not necessarily physical connections

Mapping of Data items onto nodes in Chord for \( m = 4 \)
Finding Items in the DHT

- Each node in the network knows the location of some fraction of other nodes.
  - If the desired key is stored at one of these nodes, ask for it directly
  - Otherwise, ask one of the nodes you know to look in its set of known nodes.
  - The request will propagate through the overlay network until the desired key is located
  - Lookup time is $O(\log(N))$

Joining & Leaving the Network

- **Join**
  - Generate the node’s random identifier, id, using the distributed hash function
  - Use the lookup function to locate succ(id)
  - Contact succ(id) and its predecessor to insert self into ring.
  - Assume data items from succ(id)

- **Leave (Deliberate)**
  - Notify predecessor & successor;
  - Shift data to succ(id)

- **Leave (Due to Failure)**
  - Periodically, nodes can run “self-healing” algorithms
**Content Addressable Networks**  
**Structured P2P**

- A d-dimensional space is partitioned among all nodes.
- Each node and each data item is assigned a point in the space.
- Data lookup is equivalent to knowing region boundary points and the responsible node for each region.

---

**Structured Peer-to-Peer Architectures**

- 2-dim space $[0,1] \times [0,1]$ is divided among 6 nodes.
- Each node has an associated region.
- Every data item in CAN will be assigned a unique point in space.
- A node is responsible for all data elements mapped to its region.

- Mapping of data items onto nodes in Content Addressable Network (CAN).
Structured Peer-to-Peer Architectures

- To add a new region, split the region
- To remove an existing region, neighbor will take over

Splitting a region when a node joins

Decentralized Unstructured Architecture

PEER-TO-PEER NETWORK
Unstructured P2P

- Unstructured P2P organizes the overlay network as a random graph.
- Each node knows about a subset of nodes, its “neighbors”.
  - Neighbors are chosen in different ways – Physically close nodes, nodes that joined at about the same time, etc.
- Data items are randomly mapped to some node in the system and lookup is random, unlike the structured lookup in Chord.

Locating a Data Object by Flooding

- Send a request to all known neighbors
  - If not found, neighbors forward the request to their neighbors
- Works well in small to medium sized networks, doesn’t scale well
- “Time-to-live” counter can be used to control number of hops
- Example system: Gnutella & Freenet (Freenet uses a caching system to improve performance)
Comparison

- Structured networks typically guarantee that if an object is in the network it will be located in a bounded amount of time – usually $O(\log(N))$
- Unstructured networks offer no guarantees.
  - For example, some will only forward search requests a specific number of hops
  - Random graph approach means there may be loops
  - Graph may become disconnected

Superpeers

- Maintain indexes to some or all nodes in the system
- Supports resource discovery
- Act as servers to regular peer nodes, peers to other superpeers
- Improve scalability by controlling floods
- Can also monitor state of network
- Example: Napster
System Architecture

**HYBRID ARCHITECTURE**

Hybrid Architectures

- Combine client-server and P2P architectures
  - **Edge-server Systems** – ISPs, which act as servers to their clients, but cooperate with other edge servers to host shared content
  - Collaborative Distributed Systems – BitTorrent, which supports parallel downloading and uploading of chunks of a file.
    - First, interact with Client/Server system, then operate in decentralized manner.
Edge-Server Systems

Viewing The Internet as Consisting of a Collection Of Edge Servers

Collaborative Distributed Systems BitTorrent

- Clients contact a global directory (Web server) to locate a `.torrent` file with the information needed to locate a tracker;
  - A tracker is a server that can supply a list of active nodes that have chunks of the desired file.
- Using information from the tracker, clients can download the file in chunks from multiple sites in the network.
  - Clients must also provide file chunks to other users.
Collaborative Distributed Systems

BitTorrent Principal Working

BitTorrent - Justification

- Designed to force users of file-sharing systems to participate in sharing.
- Simplifies the process of publishing large files, e.g. games
  - When a user downloads your file, he becomes in turn a server who can upload the file to other requesters.
  - Share the load – doesn’t swamp your server
Architecture versus Middleware

- Where does middleware fit into an architecture?
- Middleware: the software layer between user applications and distributed platforms.
- Purpose: to provide distribution transparency
  - Applications can access programs running on remote nodes without understanding the remote environment

Middleware may also have an architecture
- For example, CORBA has an object-oriented style.
- Use of a specific architectural style can make it easier to develop applications, but it may also lead to a less flexible system.
- Possible solution – Develop middleware that can be customized as needed for different applications.
Interceptors

- Using interceptors to handle remote-object invocations.

General Approaches to Adaptive Software

- Three basic approaches to adaptive software:
  - Separation of concerns
  - Computational reflection
  - Component-based design
Summary – P2P v Client/Server

- P2P computing allows end users to communicate without a dedicated server.
- Communication is still usually synchronous (blocking)
- There is less likelihood of performance bottlenecks since communication is more distributed.
  - Data distribution leads to workload distribution.
- Resource discovery is more difficult than in centralized client-server computing & look-up/retrieval is slower
- P2P can be more fault tolerant, more resistant to denial of service attacks because network content is distributed.
  - Individual hosts may be unreliable, but overall, the system should maintain a consistent level of service

Summary – P2P v Client/Server

- Deterministic: If an item is in the system it will be found
- No need to know where an item is stored
- Lookup operations are relatively efficient
- DHT-based P2P systems scale well
- BitTorrent and Coral Content Distribution Network incorporate DHT elements
Conclusion

- Architectural Design Issues
  - Centralized Architectures
    - Application Layering and Multitiered Architecture
  - Decentralized Architectures
    - Vertical distribution
    - Horizontal distribution