Distributed Software and System Architectures

Yao Liu
Distributed architectures

• Software architecture
  • Logical organization of the collection of software components that make up a distributed application

• System architecture
  • Instantiation of a software architecture
  • i.e., physical placement of software components on computers
Software architectural styles

• Architectural style formulated in terms of software components and connectors
• Component – modular unit with well-defined interfaces that is replaceable within its environment
• Connector – a mechanism that mediates communication, coordination or cooperation between components
Architecture styles

- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures
Layered architectures

• e.g., network protocols
Object-based architectures
Data-centered architectures

• Main purpose: data access and update
• Processes interact by reading and modifying data in some shared repository
• Example: web-based distributed systems where communication is through web services
Event-based architectures

- Communication via event propagation, in distributed systems seen often in Publish/Subscribe
- e.g., get email updates, register interest in market info

![Diagram of event-based architectures]

- Component
- Component
- Event bus
- Event delivery
- Publish
Shared data-space architectures

- Combination of data-centered and event based architectures
- e.g., shared distributed file systems
System architectures

• Centralized architectures
  • Client-server applications

• Decentralized architectures
  • Peer-to-Peer applications

• Hybrid architectures
Centralized architectures

- Request-reply behavior

Client

Request

Wait for result

Reply

Provide service

Server

Time
Client-Server systems

- **Clients**
  - interacts with users through a user interface
  - performs application functions
  - interacts with client *middleware* using middleware API
  - receives response and displays it if needed

- **Servers**
  - implement services
  - waits for incoming requests
  - invoked by server *middleware*
  - provide error-recovery and failure-handling services
Application software architectures for client-server systems

- Many applications can be considered to be made up of three software components or logical layers
  - user interface
  - processing layer
  - data layer
Examples

• Web search engine
  • Interface: type in a keyword string
  • Processing: processes to generate DB queries, rank replies, format response
  • Data: 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, may also require large simulations
  • Data: 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

Keyword expression → Database queries → Database with Web pages

User interface

Query generator

HTML generator

Ranking algorithm

HTML page containing list → Ranked list of page titles → Web page titles with meta-information

User-interface level

Processing level

Data level
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 architecture

Client machine

Server machine

(a) Thin Client

(b) (c) (d) (e) Fat Client

User interface
Application
Database

User interface
Application
Database

User interface
Application
Database

User interface
Application
Database

User interface
Application
Database

User interface
Application
Database
Distributed presentation

• Example: Remote Desktop
Remote presentation

- Example: X Windows

![Diagram showing user interface, application, and database with dashed line between user interface and other components.](image-url)
Distributed programs

• Example: World Wide Web
Remote data

- Example: Network File Systems (NFS)
Distributed data

- Example: Browser with cache, Andrew File System (AFS), Coda
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.
Multi-tiered architectures
(3-tiered architecture)

• An example of a server acting as client
Motivation for multi-tier architectures

• Frees clients from dependencies on the exact implementation of the database
• It allows “business logic” to be concentrated in one place
  • Software updates are restricted to middle layer
• Performance improvements possible by batching requests from many clients to the database
• Database and business logic tiers could be implemented by multiple servers for scalability
Decentralized architectures

- A distributed system architecture:
  - No centralized control
  - Nodes are symmetric in function
- Nodes are unreliable
- Nodes form an overlay network
Centralized vs. decentralized architectures

• 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** (P2P): each node has roughly the same processing capabilities and stores/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 & control is not hierarchical; all about equal
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

- 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.
Peer-to-peer computing and networking
Peer-to-peer overlay network
Organization of nodes in P2P systems

- Centralized directory
  - Original Napster
- Unstructured P2P systems
  - Gnutella and its successors
- Structured P2P systems
  - Based upon Distributed Hash Tables (DHTs)
  - Chord, CAN, Tapestry, …
Centralized lookup (Napster)

Simple, but $O(N)$ state and a single point of failure
Unstructured P2P systems

• Overlay network resembles a random graph
• Searching for content based upon query flooding
  • Gnutella
• Each node knows about a subset of nodes, its “neighbors”
• Data items are randomly mapped to some node in the system & lookup is random
• Second generation P2P file sharing systems (e.g., Kazaa) introduced some structure in the form of superpeers
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
Flooded queries (Gnutella)

Robust, but worst case $O(N)$ messages per lookup
Flooded queries
Back propagation
Superpeers

- A hierarchical organization of nodes into a superpeer network (e.g., Kazaa)
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
Structured P2P systems

- A common approach is to use a **distributed hash table** (DHT) to organize the nodes
- Single-node hash table:
  - key = hash(name)
  - put(key, value)
  - get(key) \(\rightarrow\) value
- How do I do this across millions of hosts on the Internet?
  - DHT
What is a DHT?

• Distributed Hash Table:
  • key = hash(data)
  • lookup(key) → node IP address (Chord)
  • send-RPC(IP address, PUT, key, value)
  • send-RPC(IP address, GET, key) → value

• Possibly a first step towards truly large-scale distributed systems
  • a tuple in a global database engine
  • a data block in a global file system
  • rare.mp3 in a P2P file-sharing system
Characteristics of DHT

• 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 (example of decentralized algorithms)
Distributed Hash Table

- **Chord:**
  - Map node ID to a large circular space
  - Map keys (hash(data)) to the same circular space
  - Key $k$ belong to the first node whose identifier is equal to or follows $k$ in the identifier space (successor($k$))
Lookup in Chord

• Every node need to be aware of the next node on the ring

• May traverse all \( N \) nodes to find the Key

• \( O(N) \) steps
Lookup in Chord: finger table

- The $i^{th}$ entry in the table at node $n$ contains the identity of the first node, $s$, that succeeds $n$ by at least $2^{i-1}$ on the ID circle.
- $O(\log N)$ steps

A client contacts node 1 to find the node that succeeds key 8.
Join & leave Chord

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

• Leave (normally)
  • Notify predecessor & successor;
  • Shift data to \( \text{succ}(id) \)

• Leave (due to failure)
  • Periodically, nodes can run “self-healing” algorithms
Hybrid architectures

- Client-server combined with decentralized architectures
  - Collaborative distributed systems: e.g., BitTorrent, which supports parallel downloading and uploading of chunks of a file. First, interact with C/S system, then operate in decentralized manner.

- Edge-server systems: e.g., Content Delivery Network (CDN), edge servers at ISPs act as servers to their clients, but cooperate with other edge servers to host shared content
Hybrid architectures: BitTorrent

• Clients contact a global directory (Web server) to locate a *.torrent* file with the information needed to locate a tracker; 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.
Hybrid architectures: BitTorrent

Web Server

Tracker

Peer [Leech]
Downloader

Peer

Peer [Leech]

C
Peer [Seed]
Hybrid architectures: BitTorrent

Web Server

Tracker

Get-announce

A

Peer

[Leech]

Downloader

B

Peer

[Leech]

C

Peer

[Seed]
Hybrid architectures: BitTorrent

Web Server

Tracker

Response-peer list

A

Web page with link to .torrent

B

Peer [Leech]

C

Peer [Seed]

Downloader

Peer [Leech]
Hybrid architectures: BitTorrent

Web Server

Tracker

Web page with link to .torrent

Peer [Leech]
Downloader

Peer [Leech]

Peer [Seed]

A

B

C

Shake-hand

Shake-hand

Shake-hand
Hybrid architectures: BitTorrent

- Web Server
- Tracker
- B
  - Leech
  - Downloader
- Peer B
- Peer C [Seed]
- Peer A
  - Leech
Hybrid architectures: BitTorrent

Web Server

Tracker

Web page with link to .torrent

Peer [Leech]
Downloader

Peer
[Leech]

Peer [Seed]

A

B

C

pieces

pieces

pieces
Hybrid architectures: BitTorrent

Web Server

Tracker

Get-announce

Response-peer list

Peer [Leech]

Downloader

Web page with link to .torrent

Peer [Leech]

Peer [Seed]

C

A

B

pieces

pieces

pieces
Hybrid architectures: edge-server systems

- Content Delivery Network (CDN)
P2P vs. Client/Server

- P2P computing allows end users to communicate without a dedicated server.
- 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
Architecture vs. 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
Software and hardware service layers in distributed systems

From Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms © Prentice-Hall, Inc. 2002
Example Client/Server middleware

• Remote Procedure Call (RPC)

Remote Procedure Call (RPC)
- Uses the well-known procedure call semantics.
- The caller makes a procedure call and then waits. If it is a local procedure call, then it is handled normally; if it is a remote procedure, then it is handled as a remote procedure call.
- Caller semantics is blocked send; callee semantics is blocked receive to get the parameters and a nonblocked send at the end to transmit results.
Middleware

• Definitions
  • Middleware is a set of common business-unaware services that enable applications and end-users to interact with each other across a network
  • Distributed system services that have standard programming interfaces and protocols … services “sit in the middle” above OS and network software and below industry-specific applications
  • The “/” in client/server applications
Examples

• ftp, email
• Web browsers
• Database drivers and gateways
• OSF’s DCE (Distributed Computing Environment)
• OMG’s CORBA (Common Object Request Broker Architecture)
• Microsoft .NET
• Java RMI, JINI, Javaspaces, JMS
• Web services software – SOAP, REST
Functional view of middleware

- Information exchange services
- Application-specific services
  - specialized services, e.g. transactional services and replication services for distributed databases, groupware services for collaborative applications, specialized services for multimedia applications
  - business-unaware
- Management and support services
  - needed for locating distributed resources and administering resources across the network
Reading

• Chapter 2 of Tbook
• Articles on P2P systems (Blackboard or course website)