CSE 5306
Distributed Systems

Architectures

Jia Rao

http://ranger.uta.edu/~jrao/
Architecture

• Software architecture
  - How software components are organized,
  - And how they interact with each other

• System architecture
  - The instantiation of software architecture
    - Centralized architecture, client-server system
    - Decentralized architecture, peer-to-peer system
    - Hybrid architecture, edge computing
Architectural Style

• Component
  □ A modular unit with well-defined interfaces
  □ It is replaceable

• Connector
  □ Mediates communication, coordination, and cooperation among components
  □ Remote procedure calls, message passing, streaming data
Software architecture

• Layered architectures
  □ Widely adopted by the networking community

• Object-oriented architectures
  □ Each object corresponds to a component; interactions are through (remote) procedure calls

• Data-centered architectures
  □ Components communicate through a shared repository

• Event-based architectures
  □ Processes communicate through the propagation of events, which can also carry data
Layered architecture
THE 7 LAYERS OF OSI

PDU (Protocol Data Unit)
(units of data passed between layers)

Header      Data

Term for a unit of data at this layer

DATA

TRANSMIT

Application layer
(sometimes called "Layer 8")

Presentation layer

Session layer

Transport layer

Network layer

Data link layer

Physical layer

Physical Link, or Medium - sometimes called "Layer 0" - data unit is a "bit"

--From http://www.infocellar.com/networks/osi-model.htm
Object-oriented architecture

- Each object is an autonomous system that interacts with each other via RPC or RMI
- Example: client-server style
Event-based architecture

- Decoupled in space
  - Processes are loosely coupled, need not explicitly refer to each other
- Communication via propagation of events
  - Mostly publish/subscribe system
Shared data-space architecture

• Not only decoupled in space but also decouple in time
  - Processes need not both be active when communication takes place [More details in Chap 13]

• Examples of shared data-space architecture
  - Shared distributed file systems
System architecture

- Centralized architectures
  - Client-server model
  - Application layering
  - Multi-tiered architecture

- Decentralized architectures
  - Peer-to-peer architecture
  - Overlay networks

- Hybrid architectures
  - Edge-server systems
  - Collaborative distributed systems
The client-server model

General interaction between a client and a server.
An example client and server

```c
void *worker(void *arg) // worker thread
{
    unsigned int socket;
    socket = *(unsigned int *)arg;
    process (socket);
    pthread_exit(0);
}

int main (void) // main thread, or dispatcher thread
{
    unsigned int server_s, client_s, i=0;
    pthread_t threads[200];
    server_s = socket(AF_INET, SOCK_STREAM, 0);
    ......
    listen(server_s, PEND_CONNECTIONS);
    while(1){
        client_s = accept(server_s, ...);
        pthread_create(&threads[i++], &attr, worker, &client_s);
    }
}
```
Client-server communication

• Connectionless protocol
  □ Hard for a sender to detect if the message is successfully received
    • Retransmission may cause problems
  □ OK for idempotent operations
    • Operations that can be repeated many times without harm [More details in Chap 8]

• Connection-oriented protocol
  □ Often used for non-idempotent operations
  □ Problem: low performance and high cost (e.g., TCP/IP)
Application layering

• Many client-server system can be divided into three levels
  □ The user-interface level: display management
  □ The processing level: core functionality of applications
  □ The data level: actual data being acted on (database or file systems)
User-interface level

- Clients implement the user-interface level allowing end users to interact with applications.
  - A character-based screen: mainframe environment
  - A graphical display: X-Windows, Windows, Apple Mac
  - A graphical window: exchange data through user actions
Processing level

• Example: Internet search engine
Data level

• Data level contains the programs that maintain the actual data on which the application operate.
  □ Data are often persistent.
    • Even if no application is running, data will be stored somewhere for next use.
  □ Keeping data consistent across different applications.
Two-tiered architecture

- The simplest organization is to have only two types of machines:
  - A client that only containing (part of) the user-interface level
  - A server containing the rest (processing level and data level)
Three-tiered architecture

• The server tier in two-tiered architecture becomes more and more distributed
  - A single server is no longer adequate for modern information systems

• The three-tiered architecture

![Diagram of three-tiered architecture: User interface (presentation) → Application server → Database server]
Decentralized architecture

• Multi-tiered architecture is vertical distribution
  □ Placing logically different components on different machines

• An alternative is horizontal distribution (P2P systems)
  □ A collection of logically equivalent parts
  □ Each part operates on its own share of the complete data set, thus balancing the load

• The main question for peer-to-peer system is
  □ How to organize the processes in an overlay network
    • A network in which the nodes are formed by the processes and the links represent the possible communication channels.
  □ Two types: structured and unstructured
Structured P2P architectures

• Structured: the overlay network is constructed in a deterministic procedure
  □ Most popular: distributed hash table (DHT)

• Key questions
  □ How to map data item to nodes
  □ How to find the network address of the node responsible for the needed data item

• Two examples
  □ Chord and content addressable network (CAN) [More details in Chap. 5]
Chord System

![Diagram of the Chord system showing nodes and associated data keys.](image)

- Actual node
- Associated data keys

Nodes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
Content addressable network

- 2-dim space \([0,1] \times [0,1]\) is divided among 6 nodes
- Each node has an associated region
- Every data item in CAN is assigned a unique point in space
- The node owning the region is responsible for the data item
Unstructured P2P architectures

• Largely relying on randomized algorithm to construct the overlay network
  □ Each node has a list of neighbors, which is more or less constructed in a random way

• One challenge is how to efficiently locate a needed data item
  □ Flood the network

• Many systems try to construct an overlay network that resembles a random graph [More details in Chap. 5]

• Each node maintains a partial view, i.e., a set of live nodes randomly chosen from the current set of nodes
Super-peers

- In unstructured peer-to-peer systems, locating relevant data items can become problematic as the network grows.
  - Super-peers: Make use of special nodes that maintain indexes of data items.

- How to select the nodes that are eligible to as the super-peer?
Hybrid Forms: Edge-Server System

• Servers are placed “at the edge” of the network.
  - The edge is formed by the boundary between enterprise networks and the actual Internet.

• Edge server’s main purpose is to serve content
  - Web-based solutions [More details in Chap 12]
Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.
Collaborative Distributed Systems

- BitTorrent file-sharing system.
  - The basic idea is when an end user is looking for a file, he downloads chunks of the file from other active users.
- The design goal is to ensure collaboration.

Figure 2-14. The principal working of BitTorrent [adapted with permission from Pouwelse et al. (2004)].
Collaborative Content Distributed Systems

• End users provide enhanced web servers that are capable of collaborating in the replication of Web pages.

• Each server has the following components:
  - A component that can redirect client requests to other servers
  - A component for analyzing access patterns
  - A component for managing the replication of web pages
Architectures Versus Middleware

- Middleware forms a layer between applications and distributed platforms, the purpose is to provide a degree of distribution of transparency.

- Middleware systems usually follow a specific architecture style.
  - Object-based architecture style: CORBA, OMG, and 2004a
  - Event-base architecture style: TIB/Rendezvous
    - Benefits: designing applications become simpler
    - Drawbacks: Adding other interaction patterns is difficult

- Solutions should be adaptable to applications requirements
  - Make several versions of a middleware system
  - Configure, adapt, and customize the middleware as needed by applications
Interceptors: adapt the middleware

- Environment in which distributed applications are executed changes continuously.
  - Mobility, variance in the quality-of-service of networks
  - Failing hardware, battery drainage

- An interceptor is nothing but a software construct that will break the usual flow of control and allow other application specific code to be executed.
Figure 2-15. Using interceptors to handle remote-object invocations.
Adaptive Software

• Three techniques about software adaption:
  □ Separation of concerns
    • Separate the parts that implement functionality from those that take care of reliability, performance, and security
  □ Computation reflection
    • The ability of a program to inspect itself and if necessary, adapt its behavior
  □ Computation-based design
    • Automatically selection of the best implementation of a component during runtime
Discussion about Middleware

• Middleware are usually bulky and complex:
  □ Provide distribution transparency
  □ Distributed applications have extra-functional requirements which conflicts with the aim at achieving this transparency
  □ Necessary to adapt the applications?
    • Environment changes: faulty hardware, security attacks
    • Distributed system cannot be shut down
Self-management in Distributed Systems

• Distributed systems should:
  - Support as many applications as possible
    - shielding undesirable features inherent to network
  - Support application-specific solutions
    - full distributed transparency is not what most applications want
  - Adapting their execution behavior not to modify the software components they comprise

• Autonomic computing
  - High-level feedback control systems allowing automatic adaptations to changes
The Feedback Control Model

• Adaptions take place with feedback control loops
  □ Uncontrollable parameters come from:
    • The environment distributed systems is executing
    • Unanticipated component interaction
  □ Metric estimation component
    • Monitor the running distributed systems
  □ Feedback analysis component (core component)
    • Analyzes the measurements and compares them to reference values
  □ Adjustment component
    • Change the behavior of the system: scheduling priorities, switching services, moving data, redirecting requests etc.
Uncontrollable parameters (disturbance / noise)

Initial configuration → Corrections → Core of distributed system → Observed output

Adjustment measures

Reference input → Analysis → Metric estimation

Adjustment triggers → Measured output

Figure 2-16. The logical organization of a feedback control system.
System Monitoring with Astrolabe

• General monitoring of large distributed systems
  - Organizing a large collections of hosts into a hierarchy of zones
  - The lowest-level zones consists of a single host
  - The top-level zone covers all hosts

• Each host runs an Astrolabe process called agent
  - Agent collects information of the hosts in each zone
    - Local information of each host is stored in a set of attributes:
      - Only the attributes of the lowest-level are writable
  - Agent communicates with each other
Figure 2-17. Data collection and information aggregation in Astrolabe.
Automatic Component Repair Management in Jade

• Detecting component failures and replacing them automatically during runtime

• Repair Management Domain:
  - A number of nodes
    - Each node represents a server
    - Equipped with failure detectors
  - A node manager
    - Adding and removing nodes from domain

• Crucial data has been lost?
Summary

- Software architecture and System architecture
  - Software architecture: logical organization
  - System architecture: implementation

- Architecture Styles:
  - Layered architecture
  - Object orientation architecture
  - Event orientation and Data-space architecture

- Centralized and Decentralized architectures
  - Centralized: client-server architectures
  - Decentralized: peer-to-peer architectures: structured and unstructured

- Self-managing distributed systems
  - Feedback-control loops