CSE 5306
Distributed Systems

Architectures
Architecture

- **Software architecture**
  - How software components are organized,
  - How software components interact

- **System architecture**
  - Instantiation and placement of software components on real machines
    - Centralized architecture, client-server system
    - Decentralized architecture, peer-to-peer system
    - Hybrid architecture
Software Architecture

• **Layered architecture**
  - widely adopted by the networking community

• **Object-based architecture**
  - E.g., client-server style (ftp)

• **Data-centered architecture**
  - Communicate through a common repository

• **Event-based architecture**
  - Communicate through the propagation of events, e.g., publish/subscribe systems
Layered Architecture

Layer N

Layer N-1

Layer 2

Layer 1

Request flow

Response flow

(a)
Object-Based Architecture

(b)
Event-Based Architecture

- Decoupled in space (referentially decoupled)
  - Processes are loosely coupled, need not explicitly refer to each other
- Communication via propagation of events
  - Mostly publish/subscribe, e.g., clients register in market info.

(a)
Shared Data-Space Architecture

- Not only decoupled in space but also decoupled in time
  - Processes need not both be active when communication takes place
- Example of shared data-space architecture
  - Shared distributed file systems, web-based distributed systems
System Architecture

• Centralized architectures
  - Application layering (logical software layering)
  - Multi-tiered architectures (system architecture)

• Decentralized architectures
  - Structured P2P (peer-to-peer) architectures
  - Unstructured P2P architectures
  - Topology management of overlay networks
  - Superpeers

• Hybrid architectures
  - Edge-server systems
  - Collaborative distributed systems
Centralized Architecture

General interaction between a client and a server.
Client-Server Communication

• **Connectionless protocol**
  - It is hard for a sender to detect if the message is successfully received
    • Retransmission may cause problems
  - Usually ok for idempotent operations
    • Operations can be repeated many times without harm, e.g., get a quotes on stock, search on web

• **Connection-oriented protocols**
  - Often used for non-idempotent operations
    • E.g., buying stock
  - Problem: low performance in local-area networks
Application Layering

- Many client-server system can be divided into three levels
  - The user-interface level
  - The processing level
  - The data level

- Example: the internet search engine
Two-tiered Architectures

- The simplest way to place a client-server application is
  - A client machine that only implements (part of) the user-interface level
  - A server machine implementing the rest, i.e., the processing and data levels
  - This is so called the two-tiered architecture

- Thin-client model and fat-client model
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
- This leads to three-tiered architecture
  - Server may acting as a client
Decentralized Architecture

• Multi-tiered architectures can be considered as vertical distribution
  - Placing logically different components on different machines

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

• The main question for peer-to-peer system is
  - How to organize the processes in an overlay network
  - 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)
Chord
Content Addressable Network (1)

• 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
• That node is responsible for that data element.
• To add a new region, split the region
• To remove an existing region, neighbor will take over
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 overly network that resembles a random graph
  - Each node maintains a partial view, i.e., a set of live nodes randomly chosen from the current set of nodes
Random Graph

- Given N nodes, we build a random graph by
  - having an edge between any two nodes at a probability p
- The random graph theory is to answer
  - what value should p have such that it is “almost certainly true” that the graph is fully connected
Partial View Construction

- A framework by Jelasity et al. in 2004
- Nodes exchange entries from their partial view regularly
  - Each entry is associated with an age tag
- Consists of an active thread and a passive thread
  - The active thread initiate the communication with a selected peer for partial view propagation
  - The passive thread waits for response from another peer and update its partial view accordingly
  - A node can be in **PUSH** mode or **PULL** mode
The Active Thread

Actions by active thread (periodically repeated):

select a peer P from the current partial view;
if PUSH_MODE {
    mybuffer = [(MyAddress, 0)];
    permute partial view;
    move H oldest entries to the end;
    append first c/2 entries to mybuffer;
    send mybuffer to P;
} else {
    send trigger to P;
}
if PULL_MODE {
    receive P’s buffer;
}
construct a new partial view from the current one and P’s buffer;
increment the age of every entry in the new partial view;

(a)
The Passive Thread

Actions by passive thread:

receive buffer from any process Q;
if PULL_MODE {
    mybuffer = [(MyAddress, 0)];
    permute partial view;
    move H oldest entries to the end;
    append first c/2 entries to mybuffer;
    send mybuffer to P;
}
construct a new partial view from the current one and P’s buffer;
increment the age of every entry in the new partial view;
(b)
Topology Management

- Some specific topologies may benefit the applications in a given P2P system
  - E.g., only including nearest peers in the partial view may reduce the latency of data delivery

- Question:
  - How to constructing a specific topology from a unstructured P2P systems

- Solution: two-layered approach
  - Lower layer: unstructured P2P outputs a random graph
  - Higher layer: carefully exchange and selecting entries to build a desired topology
Two-Layered Approach

Structured overlay

Protocol for specific overlay

Random peer

Random overlay

Protocol for randomized view

Links to topology-specific other nodes

Links to randomly chosen other nodes
Example of Two-Layer Approach

Generating a specific overlay network using a two-layered unstructured peer-to-peer system (Jelasity and Babaoglu, 2005).
Finding Data Items

• This is quite challenging in unstructured P2P systems
  - Assume a data item is randomly placed

• Solution 1: Flood the network with a search query

• Solution 2: A randomized algorithm
  - Let us first assume that
    • Each node knows the IDs of \( k \) other randomly selected nodes
    • The ID of the hosting node is kept at \( m \) randomly picked nodes
  - The search is done as follows
    • Contact \( k \) direct neighbors for data items
    • Ask your neighbors to help if none of them knows
  - What is the probability of finding the answer directly?
Superpeers

• Used to address the following question
  – How to find data items in unstructured P2P systems
  – Flood the network with a search query?

• An alternative is using superpeers
  – Nodes such as those maintaining an index or acting as a broker are generally referred to as superpeers
  – They hold index of info. from its associated peers (i.e. selected representative of some of the peers)

• Remaining question: how to pick the superpeers
An Example of Superpeer Networks

A hierarchical organization of nodes into a superpeer network.

Regular peer

Superpeer

Superpeer network
Hybrid Architectures

• Many real distributed systems combine architectural features
  - E.g., the superpeer networks -- combine client-server architecture (centralized) with peer-to-peer architecture (decentralized)

• Two examples of hybrid architectures
  - Edge-server systems
  - Collaborative distributed systems
Edge-Server Systems

- Deployed on the Internet where servers are “at the edge” of the network (i.e. first entry to network)
- Each client connects to the Internet by means of an edge server
Collaborative Distributed Systems

• A hybrid distributed model that is based on mutual collaboration of various systems
  - Client-server scheme is deployed at the beginning
  - Fully decentralized scheme is used for collaboration after joining the system

• Examples of Collaborative Distributed System:
  - BitTorrent: is a P2P File downloading system. It allows download of various chunks of a file from other users until the entire file is downloaded
  - Globule: A Collaborative content distribution network. It allows replication of web pages by various web servers
BitTorrent

The principal working of BitTorrent (Pouwelse et al. 2004).

Information needed to download a specific file

Many trackers, one per file, tracker holds which node holds which chunk of the file

Client node

K out of N nodes

A BitTorrent Web page

Ref. to file server

Web server

Lookup(F)

.torrent file for F

File server

Ref. to tracker

List of nodes storing F

Tracker

Node 1

Node 2

Node N
Globule

- Collaborative content distribution network:
  - Similar to edge-server systems
  - Enhanced web servers from various users that replicates web pages

- 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.

- Has a centralized component for registering the servers and make these servers known to others
Benefits of Globule

- **Example:**
  - Alice has a web server; Bob has a web server
  - Alice’s server can have replicated contents of the Bob’s server and vice versa

- **Good if your server goes down**
- **Good if too much traffic that your server can not handle or gets too slow**
- **Better Geographic diversity**
  - Allow users to get quick response from the nearest server with the replicated page
Architectures v.s. Middleware

• A middleware layer between application and the distributed platforms for distribution transparency

• The question is:
  - Given the software and system architecture, where the middleware fits in?

• Many middleware follows a specific architecture style
  - Object-based style, event-based style
  - Benefits: simpler to design application
  - Limitations: the solution may not be optimal

• Should be adaptable to application requirements
  - Separating policies from mechanism
Supporting Technology: Interceptors

• An Interceptor is a software that
  – breaks the usual flow of control and
  – allows other (application specific) code to be executed

• It makes middleware more adaptable to
  – application requirements and changing environment

• Interceptors are good for
  – providing transparent replication and
  – improving performance
Interceptors

May want to send to many other B’s (i.e. replicated)

May want to break a large message for better performance

Using interceptors to handle remote-object invocations
General Approaches for Adaptability

• Separation of concerns:
  - Modularizing the system and separate security from functionality
  - However, the problem is that a lot of things you cannot easily separate, e.g., security

• Computational reflection
  - Ability to inspect itself, and if necessary, adapt its behavior
  - Reflective middleware has yet to proof itself as a powerful tool to manage the complexity of distribute systems

• Component-based design (stand-alone)
  - However, components are less independent than one may think
  - Replacement of one component may have huge impact on others
Self-Management in Distributed Sys.

- Distributed systems are often required to adapt to environmental changes by
  - switching policies for allocation resources
- The algorithms to make the changes are often already in the components
  - But the challenge is how to make such change without human intervention
- A strong interplay between software architectures and system architectures
  - Organize components in a way that monitoring and adjustment can be done easily
  - Decide where the processes to be executed to do the adaption
The Feedback Control Systems

- Allow automatic adaption to changes by means of one or more feedback control loops
  - self-managing, self-healing, self-configuration, self-optimization, etc.

Logical organization of a feedback control system (the physical organization could be very different)
System Monitoring with Astrolabe

- **A tool for observing system behavior in a distributed system**
  - One component of the feedback control system
- Every host runs an Astrolabe process, called an agent
  - Agents are organized as a hierarchy

<table>
<thead>
<tr>
<th></th>
<th>avg_load</th>
<th>avg_mem</th>
<th>avg_procs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.06</td>
<td>0.55</td>
<td>47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IP-addr</th>
<th>load</th>
<th>mem</th>
<th>procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.2</td>
<td>0.03</td>
<td>0.80</td>
<td>43</td>
</tr>
<tr>
<td>192.168.1.3</td>
<td>0.05</td>
<td>0.50</td>
<td>20</td>
</tr>
<tr>
<td>192.168.1.4</td>
<td>0.10</td>
<td>0.35</td>
<td>78</td>
</tr>
</tbody>
</table>
Replication Strategy in Globule

• When enough requests for a page is collected,
  - Globule does a “what-if analysis” to evaluate the replication policies and select the best policy

• The evaluation is done using a trace-driven simulation
Replication Strategy in Globule

- How many requests (i.e., trace length) are needed for evaluation?

The dependency between prediction accuracy and trace length.
Automatic Component Repair in Jade

• Jade: A Java implementation framework that allows components to be added and removed at runtime

• Steps in a simple auto-repair example
  – Terminate every binding between a component on a non-faulty node, and a component on the node that just failed.
  – Request the node manager to start and add a new node to the domain.
  – Configure the new node with exactly the same components as those on the crashed node.
  – Re-establish all the bindings (between client & server interfaces) that were previously terminated.

• Done via a repair management server (can be replicated)