Distributed Systems (COMP9243)

Lecture 3: System Architecture

Slide 1
1. System Architectures
   - Client-server (and multi-tier)
   - Peer to peer
   - Hybrid architectures
2. Processes & Server Architecture

Slide 2

Architecture

Building a Distributed System

Slide 3
Two questions:
1. Where to place the hardware?
2. Where to place the software?

Slide 4
System Architecture:
- placement of machines
- placement of software on machines
Where to place?:
- processing capacity, load balancing
- communication capacity
- locality
Mapping of services to servers:
- Partitioning
- Replication
- Caching
Slide 5  ARCHITECTURAL PATTERNS

Slide 6  CLIENT-SERVER

Slide 7  CLIENT-SERVER from another perspective:

Slide 8  Example client-server code in C:

```c
client(void) {
    struct sockaddr_in cin;
    char buffer[bufsize];
    int sd;

    sd = socket(AF_INET,SOCK_STREAM,0);
    connect(sd,(void *)&cin,sizeof(cin));
    send(sd,buffer,strlen(buffer),0);
    recv(sd,buffer,bufsize,0);
    close (sd);
}
```
server(void) {
  struct sockaddr_in cin, sin;
  int sd, sd_client;

  sd = socket(AF_INET, SOCK_STREAM, 0);
  bind(sd, (struct sockaddr *)&sin, sizeof(sin));
  listen(sd, queuesize);
  while (true) {
    sd_client = accept(sd, (struct sockaddr *)&cin, &addrlen);
    recv(sd_client, buffer, sizeof(buffer), 0);
    DoService(buffer);
    send(sd_client, buffer, strlen(buffer), 0);
    close(sd_client);
  }
  close(sd);
}

Example client-server code in Erlang:
% Client code using the increment server
client (Server) ->
  Server ! {self (), 10},
  receive
    {From, Reply} -> io:format ("Result: ~w~n", [Reply])
  end.
% Server loop for increment server
loop () ->
  receive
    {From, Msg} -> From ! {self (), Msg + 1},
    loop ();
    stop -> true
  end.
% Initiate the server
start_server() -> spawn (fun () -> loop () end).

Which is the best approach?

Vertical Distribution (Multi-tier)

Three ‘layers’ of functionality:
- User interface
- Processing/Application logic
- Data

Logically different components on different machines

Leads to Service-Oriented architectures (e.g. microservices).
Vertical Distribution from another perspective:

Slide 13

User interface (presentation)
Application server
Database server

Request operation
Time
Wait for result
Wait for data
Return result
Return data

How scalable is this?

Horizontal Distribution

Slide 14

Front end handling incoming requests
Requests handled in round-robin fashion
Replicated Web servers each containing the same Web pages
Disks

Internet

→ Logically equivalent components replicated on different machines

How scalable is this?

Note: Scaling Up vs Scaling Out?
Horizontal and Vertical Distribution not the same as Horizontal and Vertical Scaling.

Vertical Scaling: Scaling UP
Increasing the resources of a single machine

Horizontal Scaling: Scaling OUT
Adding more machines.
Horizontal and Vertical Distribution are both examples of this.

Service Oriented Architecture (SOA)

Slide 16

Auction Service
Stock Service
Bank Service
Photo Service

HTTP add_photo delete_photo
search
get_auction buy
manage_auction bid
sell
update_photo
balance transfer

HTTP

XML-RPC

SOAP

MICROSERVICES
**MICROSERVICES**

‘Extreme’ vertical distribution

- split application logic into many (reusable) services
- services limited in scope: single-purpose, do one thing really well
- orchestrate execution of services

---

**PEER TO PEER**

- All processes have client and server roles: servant

Why is this special?

---

**PEER TO PEER AND OVERLAY NETWORKS**

How do peers keep track of all other peers?

- static structure: you already know
- dynamic structure: Overlay Network
  - structured
  - unstructured

Overlay Network:

- Application-specific network
- Addressing
- Routing
- Specialised features (e.g., encryption, multicast, etc.)

---

**Example:**
**Unstructured Overlay**

- Data stored at random nodes
- Partial view: node's list of neighbours
- Exchange partial views with neighbours to update

What's a problem with this?

**Structured Overlay**

Distributed Hash Table:

- Nodes haveidentifier and range, Data has identifier
- Node is responsible for data that falls in its range
- Search is routed to appropriate node
- Examples: Chord, Pastry, Kademlia

What's a problem with this?

**Hybrid Architectures**

Combination of architectures.

Examples:
- Superpeer networks
- Collaborative distributed systems
- Edge-server systems

**Superpeer Networks:**
- Regular peers are clients of superpeers
- Superpeers are servers for regular peers
- Superpeers are peers among themselves
- Superpeers may maintain large index, or act as brokers
- Example: Skype

What are potential issues?
Collaborative Distributed Systems:

Example: BitTorrent

- Node downloads chunks of file from many other nodes
- Node provides downloaded chunks to other nodes
- Tracker keeps track of active nodes that have chunks of file
- Enforce collaboration by penalising selfish nodes

What problems does BitTorrent face?

Edge-Server Networks:

- Servers placed at the edge of the network
- Servers replicate content
- Mostly used for content and application distribution
- Content Distribution Networks: Akamai, CloudFront, CoralCDN

What are the challenges?

<table>
<thead>
<tr>
<th>Model</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-threaded process</td>
<td>No parallelism, blocking system calls</td>
</tr>
<tr>
<td>Threads</td>
<td>Parallelism, blocking system calls</td>
</tr>
<tr>
<td>Finite-state machine</td>
<td>Parallelism, non-blocking system calls</td>
</tr>
</tbody>
</table>

Stateful vs Stateless Servers

Stateful:

- Keeps persistent information about clients
  ✓ Improved performance
  x Expensive crash recovery
  x Must track clients

Stateless:

- Does not keep state of clients
  ✓ soft state design: limited client state
  ✓ Can change own state without informing clients
  ✓ No cleanup after crash
  ✓ Easy to replicate
  x Increased communication

Note: Session state vs. Permanent state
**Clustered Servers**

Logical switch (possibly multiple)

Application/Compute servers

Distributed file/database system

Slide 33

**Request Switching**

Transport layer switch:

- Logically a single TCP connection

Response

Request forwarded on

Server

Client Request Switch

Slide 34

**Virtualisation**

Virtual Machines:

- Server
- Guest OS

Host OS

Hardware

What are the benefits?

Slide 35

**Containerisation**

Contemporary Applications

- Docker
-Host Operating System
- Infrastructure

What are the benefits?

What are the drawbacks?

(From: https://www.docker.com/resources/what-container)

Slide 36
**Serverless**

Serverless does use servers!
- You don’t maintain them yourself
- You only provide functions to run
- Transparently run on servers
- Functions as a Service (FaaS)
  - code components have a short lifecycle (per request)
  - environment manages loading, starting, stopping code
  - client-side management of control-flow, application logic

**Code Mobility**

Why move code?
- Optimise computation (load balancing)
- Optimise communication

Weak vs Strong Mobility:
- **Weak** transfer only code
- **Strong** transfer code and execution segment

Sender vs Receiver Initiated migration:
- **Sender** Send program to compute server
- **Receiver** Download applets

Examples: Java, JavaScript, Virtual Machines, Mobile Agents

What are the challenges of code mobility?

**Homework**

Client Server:
- Do Exercise Client server exercise (Erlang) Part A

Hacker’s Edition: Client-Server vs Ring:
- Do Exercise Client-Server vs. Ring (Erlang)