Outline

- Recap
- Single network server
  - Concurrent network servers
  - Operational analysis
- Multiple network servers
  - Basic issues
  - Load direction
    - DNS
    - Load balancer (smart switch)
Admin

- Assignment three
Recap: High-Performance Network Servers

- Avoid blocking (so that we can reach bottleneck throughput)
  - threads
- Limit unlimited thread overhead
  - thread pool, async io
- Coordinating data access
  - synchronization (lock, synchronized)
- Coordinating behavior: avoid busy-wait
  - wait/notify; FSM
- Extensibility/robustness
  - language support/design for interfaces
Recap: Operational Laws

- Utilization law: $U = XS$
- Forced flow law: $Xi = Vi X$
- Bottleneck device: largest $Di = Vi Si$
- Little’s Law: $Qi = Xi Ri$
Interactive Response Time Law

- **System setup**
  - Closed system with \(N\) users
  - Each user sends in a request, after response, think time, and then sends next request

- **Notation**
  - \(Z = \text{user think-time}, R = \text{Response time}\)

- The total cycle time of a user request is \(R + Z\)

In duration \(T\), \#requests generated by each user: \(\frac{T}{R+Z}\) requests
Interactive Response Time Law

- If \( N \) users and flow balanced:

System Throughput \( X = \operatorname{Toal\# req./T} \)

\[\begin{align*}
X &= \frac{N}{R+Z} \\
\Rightarrow \quad R &= \frac{N}{X} - Z
\end{align*}\]
Bottleneck Analysis

\[ X(N) \leq \min \left\{ \frac{1}{D_{\text{max}}} , \frac{N}{D+Z} \right\} \]

\[ R(N) \geq \max \left\{ D, ND_{\text{max}} - Z \right\} \]

- Here D is the sum of Di
Proof

\[ X(N) \leq \min\left\{ \frac{1}{D_{\text{max}}} , \frac{N}{D+Z} \right\} \]

\[ R(N) \geq \max\{D, ND_{\text{max}} - Z\} \]

- We know

\[ X \leq \frac{1}{D_{\text{max}}} \quad R(N) \geq D \]

Using interactive response time law:

\[ R = \frac{N}{X} - Z \quad \Rightarrow \quad R \geq ND_{\text{max}} - Z \]

\[ X = \frac{N}{R+Z} \quad \Rightarrow \quad X \leq \frac{N}{D+Z} \]
In Practice: Common Bottlenecks

- No more file descriptors
- Sockets stuck in TIME_WAIT
- High memory use (swapping)
- CPU overload
- Interrupt (IRQ) overload
Summary: Operational Laws

- Utilization law: $U = XS$
- Forced flow law: $X_i = V_i X$
- Bottleneck device: largest $D_i = V_i S_i$
- Little’s Law: $Q_i = X_i R_i$
- Bottleneck bound of interactive response:

$$X(N) \leq \min\left\{\frac{1}{D_{\text{max}}}, \frac{N}{D+Z}\right\}$$

$$R(N) \geq \max \{D, ND_{\text{max}} - Z\}$$
Summary: High-Perf. Network Server

- Avoid blocking (so that we can reach bottleneck throughput)
  - Introduce threads
- Limit unlimited thread overhead
  - Thread pool, async io
- Shared variables
  - Synchronization (lock, synchronized)
- Avoid busy-wait
  - Wait/notify; FSM
- Extensibility/robustness
  - Language support/Design for interfaces
- System modeling and measurements
  - Queueing analysis, operational analysis
Outline

- Recap
- Single network server
- Multiple network servers
  - Basic issues
Why Multiple Servers?

- Scalability
  - Scaling beyond single server capability
    - There is a fundamental limit on what a single server can
      - process (CPU/bw/disk throughput)
      - store (disk/memory)
  - Scaling beyond geographical location capability
    - There is a limit on the speed of light
    - Network detour and delay further increase the delay
Why Multiple Servers?

- Redundancy and fault tolerance
  - Administration/Maintenance (e.g., incremental upgrade)
  - Redundancy (e.g., to handle failures)
Why Multiple Servers?

- System/software architecture
  - Resources may be naturally distributed at different machines (e.g., run a single copy of a database server due to single license; access to resource from third party)
  - Security (e.g., front end, business logic, and database)
Problems Caused by Multiple Servers

- **Scalability**
  - Load direction

- **Redundancy and fault tolerance**
  - Consistency
Outline

- Recap
- Single network server
- Multiple network servers
  - Basic issues
  - Load direction
Load Direction: Overview

- Global direction: select a server site for each request
- Local direction: select a specific server at the chosen site
Load Direction: Overview

https://splash.riverbed.com/docs/DOC-1705
Load Direction: Basic Architecture

- Major components
  - Server state monitoring
    - Load (incl. failed or not): what requests it can serve
  - Network path properties between clients and servers
    - E.g., bw, delay, loss, network cost
  - Server selection alg.
  - Client direction mechanism
  - Server routing/adaptation
Load Direction

- Server state
- Net path property between servers/clients
- Notify client about selection (direction mech.)
- Server selection algorithm
- Specific request of a client
- Server routing
**Network Path Properties**

- **Why is the problem difficult?**
  - **Scalability:** if do measurements, complete measurements grow with $N \times M$, where
    - $N$ is # of clients
    - $M$ is # of servers
  - **Complexity/feasibility in computing path metrics**
Network Path Properties: Improve Scalability

- **Aggregation:**
  - merge a set of IP addresses (reduce \( N \) and \( M \))
    - E.g., when computing path properties, Akamai aggregates all clients sharing the same local DNS server

- **Sampling and prediction**
  - Instead of measuring \( N \times M \) entries, we measure a subset and **predict** the unmeasured paths
  - We will cover it later in the course
Server Selection

- Why is the problem difficult?
  - What are potential problems of just sending each new client to the lightest load server?
Client Direction Mechanisms

- **Application layer**
  - App/user is given a list of candidate server names
  - HTTP redirector

- **DNS:** name resolution gives a list of server addresses

- **IP layer:** Same IP address represents multiple physical servers
  - IP anycast: Same IP address shared by multiple servers and announced at different parts of the Internet. Network directs different clients to different servers (e.g., Limelight)
  - Load balancer (smart switch) indirection: a server IP address may be a virtual IP address for a cluster of physical servers
Direction Mechanisms are Often Combined

Cluster1 in US East
  Load balancer

Cluster2 in US West
  Load balancer

Cluster2 in Europe
  Load balancer

DNS name1
  IP1

DNS name2
  IP2

IPn

----- proxy -----

Load balancer

servers
Example: Wikipedia Architecture

Example: Netflix

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.netflix.com">www.netflix.com</a></td>
<td>Netflix</td>
</tr>
<tr>
<td>signup.netflix.com</td>
<td>Amazon</td>
</tr>
<tr>
<td>movies.netflix.com</td>
<td>Amazon</td>
</tr>
<tr>
<td>agmoviecontrol.netflix.com</td>
<td>Amazon</td>
</tr>
<tr>
<td>nflx.i.87f50a04.x.lcdn.nflximg.com</td>
<td>Level 3</td>
</tr>
<tr>
<td>netflix-753.vo.llnwd.net</td>
<td>Limelight</td>
</tr>
<tr>
<td>netflix753.as.nflximg.com.com.edgesuite.net</td>
<td>Akamai</td>
</tr>
</tbody>
</table>

Multi-cloud approach: Netflix data center, Amazon cloud, and CDNs.
Example: Netflix Manifest File

- Client player authenticates and then downloads manifest file from servers at Amazon Cloud
Example: Netflix Manifest File

```xml
<nccp:bitrate>560</nccp:bitrate>
<nccp:videoprofile>
  playready-h264mpl130-dash
</nccp:videoprofile>
<nccp:resolution>
  <nccp:width>512</nccp:width>
  <nccp:height>384</nccp:height>
</nccp:resolution>
<nccp:pixelsaspect>
  <nccp:width>4</nccp:width>
  <nccp:height>3</nccp:height>
</nccp:pixelsaspect>
<nccp:downloadurls>
  <nccp:downloadurl>
    <nccp:expiration>131xxx</nccp:expiration>
    <nccp:cdnid>6</nccp:cdnid>
    <nccp:url>http://nflx.i.../...</nccp:url>
  </nccp:downloadurl>
  <nccp:downloadurl>
    <nccp:expiration>131xxx</nccp:expiration>
    <nccp:cdnid>4</nccp:cdnid>
    <nccp:url>http://netflix.../...</nccp:url>
  </nccp:downloadurl>
  <nccp:downloadurl>
    <nccp:expiration>131xxx</nccp:expiration>
    <nccp:cdnid>9</nccp:cdnid>
    <nccp:url>http://netflix.../...</nccp:url>
  </nccp:downloadurl>
</nccp:downloadurls>
```
Outline

- Recap
- Single network server
- Multiple network servers
  - Basic issues
  - Load direction
    - DNS
    - Load balancer (smart switch)
DNS Indirection and Rotation

IP address of cnn.com
157.166.226.25
157.166.226.26

DNS server for cnn.com
157.166.226.26
157.166.226.25

157.166.226.25
157.166.226.26
157.166.255.18
Example: Amazon Elastic Cloud 2 (EC2) Elastic Load Balancing

- Use the `elb-create-lb` command to create an Elastic Load Balancer.
- Use the `elb-register-instances-with-lb` command to register the Amazon EC2 instances that you want to load balance with the Elastic Load Balancer.
- Elastic Load Balancing automatically checks the health of your load balancing Amazon EC2 instances. You can optionally customize the health checks by using the `elb-configure-healthcheck` command.
- Traffic to the DNS name provided by the Elastic Load Balancer is automatically distributed across healthy Amazon EC2 instances.

http://aws.amazon.com/documentation/elasticloadbalancing/
1. **Call** `CreateLoadBalancer` **with the following parameters:**
   - **AvailabilityZones** = us-east-1a
   - **Listeners**
     - `Protocol` = HTTP
     - `InstancePort` = 8080
     - `LoadBalancerPort` = 80
     - `LoadBalancerName` = MyLoadBalancer

   The operation returns the DNS name of your LoadBalancer. You can then map that to any other domain name (such as www.mywebsite.com) (how?)

   **PROMPT**>
   ```bash
   elb-create-lb MyLoadBalancer --headers --
   listener "lb-port=80,instance-port=8080,protocol=HTTP"
   --availability-zones us-east-1a
   ```

   **Result:**
   ```
   DNS-NAME DNS-NAME
   DNS-NAME MyLoadBalancer-2111276808.us-east-1.elb.amazonaws.com
   ```

Details: Step 2

2. Call `ConfigureHealthCheck` with the following parameters:
   - `LoadBalancerName = MyLoadBalancer`
   - `Target = http:8080/ping`
     - Note: Make sure your instances respond to /ping on port 8080 with an HTTP 200 status code.
   - `Interval = 30`
   - `Timeout = 3`
   - `HealthyThreshold = 2`
   - `UnhealthyThreshold = 2`

PROMPT: `elb-configure-healthcheck MyLoadBalancer --headers --target "HTTP:8080/ping" --interval 30 --timeout 3 --unhealthy-threshold 2 --healthy-threshold 2`

Result:
```
HEALTH-CHECK TARGET INTERVAL TIMEOUT HEALTHY-THRESHOLD UNHEALTHY-THRESHOLD
HEALTH-CHECK HTTP:8080/ping 30 3 2 2
```
Details: Step 3

3. Call `RegisterInstancesWithLoadBalancer` with the following parameters:
   - `LoadBalancerName = MyLoadBalancer`
   - `Instances = [ i-4f8cf126, i-0bb7ca62 ]`

   ```command
   PROMPT> elb-register-instances-with-lb MyLoadBalancer --headers --instances i-4f8cf126,i-0bb7ca62
   ```

   Result:
   ```
   INSTANCE INSTANCE-ID
   INSTANCE i-4f8cf126
   INSTANCE i-0bb7ca62
   ```
Facebook DNS Load Direction

- A system named Cartographer (written in Python) processes measurement data and configures the DNS maps of individual DNS servers (open source tinydns)
Discussion

- Advantages and disadvantages of using DNS
Outline

- Recap
- Single network server
- Multiple network servers
  - Basic issues
  - Load direction
    - DNS
    - Load balancer (smart switch)
Clustering with VIP: Basic Idea

- Clients get a single service IP address, called virtual IP address (VIP)
- A virtual server (also referred to as load balancer, vserver or smart switch) listens at VIP address and port
- A virtual server is bound to a number of physical servers running in a server farm
- A client sends a request to the virtual server, which in turn selects a physical server in the server farm and directs this request to the selected physical server
Big Picture
VIP Clustering

L4: TCP
L7: HTTP
SSL etc.

Goals
server load balancing
failure detection
access control filtering
priorities/QoS
request locality
transparent caching

What to switch/filter on?
L3 source IP and/or VIP
L4 (TCP) ports etc.
L7 URLs and/or cookies
L7 SSL session IDs
Load Balancer (LB): Basic Structure

Problem of the basic structure?
Problem

- Client to server packet has VIP as destination address, but real servers use RIPs
  - if LB just forwards the packet from client to a real server, the real server drops the packet
  - reply from real server to client has real server IP as source -> client will drop the packet

```
state: listening
address: {*.*.6789, *:*}
completed connection queue: C1; C2
sendbuf: recvbuf:

state: established
address: {128.36.232.5:6789, 198.69.10.10.1500}
sendbuf: recvbuf:
...

state: established
address: {128.36.232.5:6789, 198.69.10.10.1500}
sendbuf: recvbuf:
...

Real Server TCP socket space
```
Solution 1: Network Address Translation (NAT)

- LB does rewriting/translation
- Thus, the LB is similar to a typical NAT gateway with an additional scheduling function
Example Virtual Server via NAT

Table 1: An example of virtual server rules

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Virtual IP Address</th>
<th>Port</th>
<th>Real IP Address</th>
<th>Port</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>202.103.106.5</td>
<td>80</td>
<td>172.16.0.2</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>TCP</td>
<td>202.103.106.5</td>
<td>8000</td>
<td>172.16.0.3</td>
<td>8000</td>
<td>2</td>
</tr>
<tr>
<td>TCP</td>
<td>202.103.106.5</td>
<td>21</td>
<td>172.16.0.3</td>
<td>21</td>
<td>1</td>
</tr>
</tbody>
</table>
LB/NAT Flow
LB/NAT Flow

- Request packet:
  - Source IP: 206.183.42.40
  - Destination IP: 192.168.0.100
  - Source IP: 192.168.0.100
  - Destination IP: 206.183.42.40

- Response packet:
  - VIP: 192.168.0.100
  - RIP: 192.168.0.201
  - RIP: 192.168.0.205
  - RIP: 192.168.0.206

- Server 1
- Server n

- Active load balancer
- Backup load balancer

Primary heartbeat
Backup heartbeat
LB/NAT Advantages and Disadvantages

- Advantages:
  - Only one public IP address is needed for the load balancer; real servers can use private IP addresses
  - Real servers need no change and are not aware of load balancing

- Problem
  - The load balancer must be on the critical path and hence may become the bottleneck due to load to rewrite request and response packets
    - Typically, rewriting responses has more load because there are more response packets
LB with Direct Reply

Direct reply  Each real server uses VIP as its IP address
LB/DR Architecture

The diagram illustrates a Load Balancer (LB) architecture connected by a single switch. The User sends requests through the Internet/Intranet to the load balancer. The load balancer connects to the real servers (Real Server 1, Real Server 2, Real Server n) via the internal network. The replies from the real servers go directly to the user.
Why IP Address Matters?

- Each network interface card listens to an assigned MAC address
- A router is configured with the range of IP addresses connected to each interface (NIC)
- To send to a device with a given IP, the router needs to translate IP to MAC (device) address
- The translation is done by the Address Resolution Protocol (ARP)
ARP Protocol

ARP is “plug-and-play”:
- nodes create their ARP tables without intervention from net administrator

A broadcast protocol:
- Router broadcasts query frame, containing queried IP address
  - all machines on LAN receive ARP query
- Node with queried IP receives ARP frame, replies its MAC address
ARP in Action

- Router broadcasts ARP broadcast query: who has VIP?
- ARP reply from LB: I have VIP; my MAC is $MAC_{LB}$
- Data packet from R to LB: destination MAC = $MAC_{LB}$
ARP and race condition:
• When router R gets a packet with dest. address VIP, it broadcasts an Address Resolution Protocol (ARP) request: who has VIP?
• One of the real servers may reply before load balancer

Solution: configure real servers to not respond to ARP request
LB via Direct Routing

- The virtual IP address is shared by real servers and the load balancer.

- Each real server has a non-ARPing, loopback alias interface configured with the virtual IP address, and the load balancer has an interface configured with the virtual IP address to accept incoming packets.

- The workflow of LB/DR is similar to that of LB/NAT:
  - the load balancer directly routes a packet to the selected server
    - the load balancer simply changes the MAC address of the data frame to that of the server and retransmits it on the LAN (how to know the real server’s MAC?)
  - When the server receives the forwarded packet, the server determines that the packet is for the address on its loopback alias interface, processes the request, and finally returns the result directly to the user.
Advantages:
- Real servers send response packets to clients directly, avoiding LB as bottleneck

Disadvantages:
- Servers must have non-arp alias interface
- The load balancer and server must have one of their interfaces in the same LAN segment
Example Implementation of LB

- An example open source implementation is Linux virtual server (linux-vs.org)
  - Used by
    - www.linux.com
    - sourceforge.net
    - wikipedia.org
    - Many commercial LB servers from F5, Cisco, ...

- More details please read chapter 2 of Load Balancing Servers, Firewalls, and Caches
Example: Wikipedia Architecture

A major remaining problem is that the LB becomes a single point of failure (SPOF).
Solutions

- **Redundant load balancers**
  - E.g., two load balancers

- **Fully distributed load balancing**
  - e.g., Microsoft Network Load Balancing (NLB)
Microsoft NLB

- No dedicated load balancer
- All servers in the cluster receive all packets
- All servers within the cluster simultaneously run a mapping algorithm to determine which server should handle the packet. Those servers not required to service the packet simply discard it.

- Mapping (ranking) algorithm: computing the “winning” server according to host priorities, multicast or unicast mode, port rules, affinity, load percentage distribution, client IP address, client port number, other internal load information.

Discussion

- Compare the design of using Load Balancer vs Microsoft NLB
Summary: Example Direction Mechanisms

- DNS name1
  - IP1
    - Cluster1 in US East
      - Load balancer
    - Cluster2 in US West
      - Load balancer
    - Cluster2 in Europe
      - Load balancer
    - proxy
      - Load balancer
        - servers
  - IP2
  - IPn

- DNS name2

- Rewrite
- Direct reply
- Fault tolerance