CS 188/219

Scalable Internet Services

Andrew Mutz

October 8, 2015

For Today

About PTEs

- Empty spots were given out
- If more spots open up, I will issue more PTEs

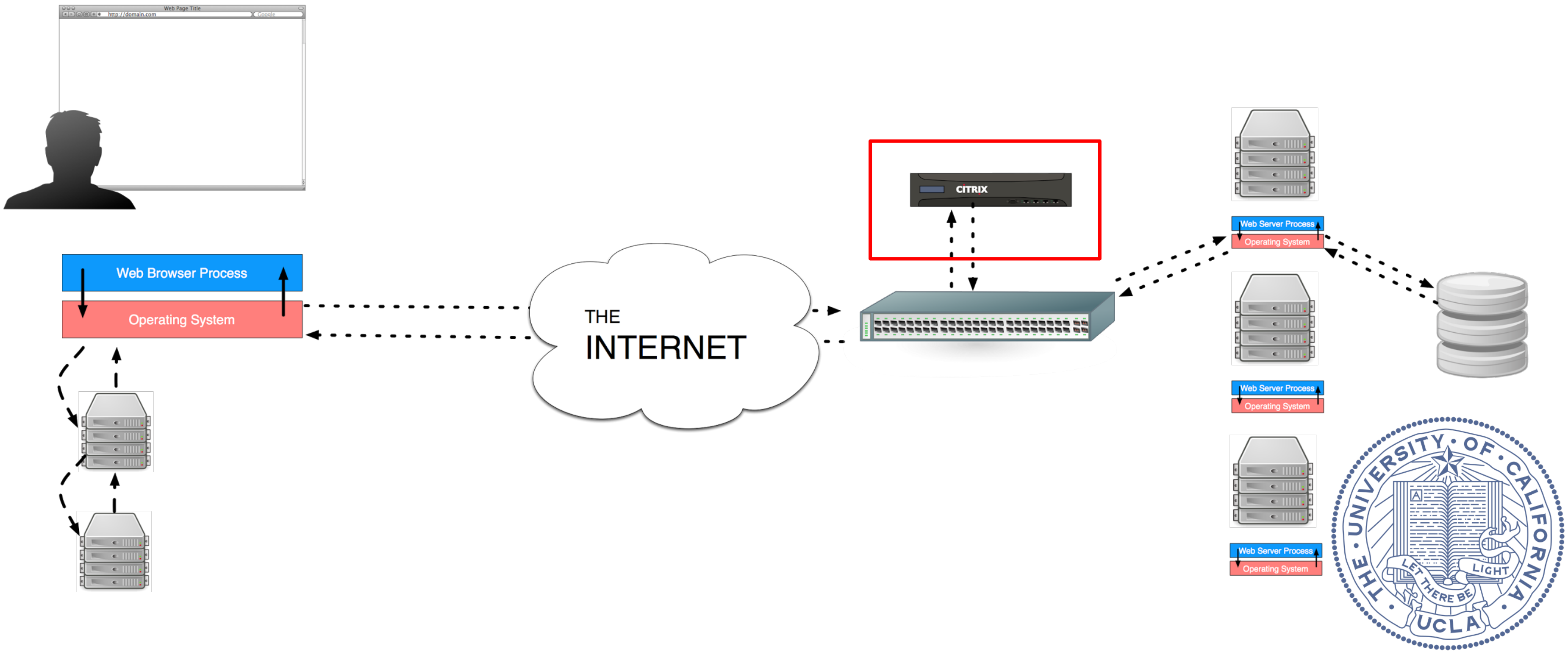
You must have a group by today. More detail at end of this lecture.

If you still need a group, come up front during halftime.

For Today

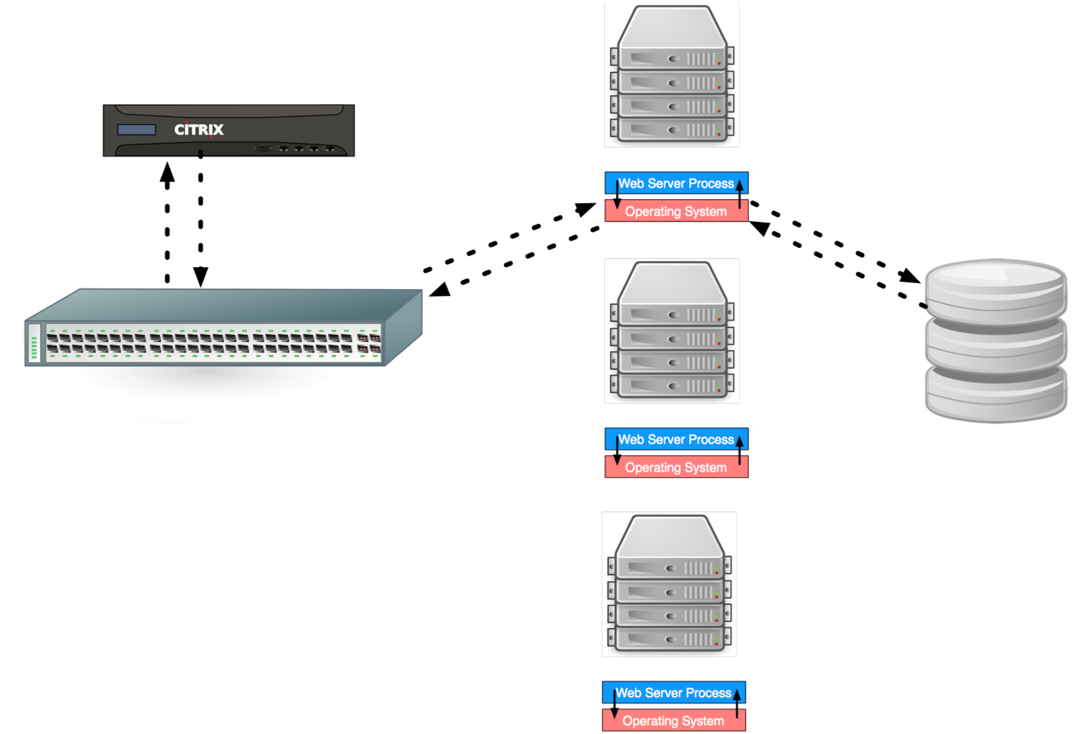
- Motivation
- Vertical Scaling
- Horizontal Scaling
- For Next Time...

Motivation



Web Page Title
http://domain.com
Google
Web Browser Process
Operating System
THE
INTERNET
CITRIX
Web Server Process
Operating System
Web Server Process
Operating System
Web Server Process
Operating System

Motivation



After today's lecture you will understand what load balancing is, and and why the architecture depicted here allows cost effective and smooth scaling paths.

Motivation

Suppose you've built something the world is excited about.

What do you do when your popularity doubles?

- ...and doubles again. and again.

Motivation

We've mentioned buying "bigger" hardware and scaling up

- This is called vertical scaling
- This is easy to do, but hits limits

We've mentioned buying more servers and scaling out

- This is called horizontal scaling
- This is harder to do, but doesn't really have limits

Today we will look at both.

Vertical Scaling

Vertical scaling

- Use a server with more memory, more/faster cores, higher bandwidth, etc.
- On EC2, there are a few scaling paths
  - T and M series: balanced mix of memory and CPU
  - C series: emphasis on CPU
  - R series: emphasis on memory

How much bang do you get for your buck?

- Let's take a look...

Vertical Scaling

We will use the same testing script from the last lecture...

1. Going to the homepage
2. Waiting for up to 2 seconds
3. Requesting a form to create a new community
4. Waiting for up to 2 seconds
5. Submitting the new community
6. Requesting a form to create a new link submission
7. Waiting for up to 2 seconds
8. Submitting the new link
9. Waiting for up to 2 seconds
10. Delete the link
11. Waiting for up to 2 seconds
12. Delete the community

Vertical Scaling

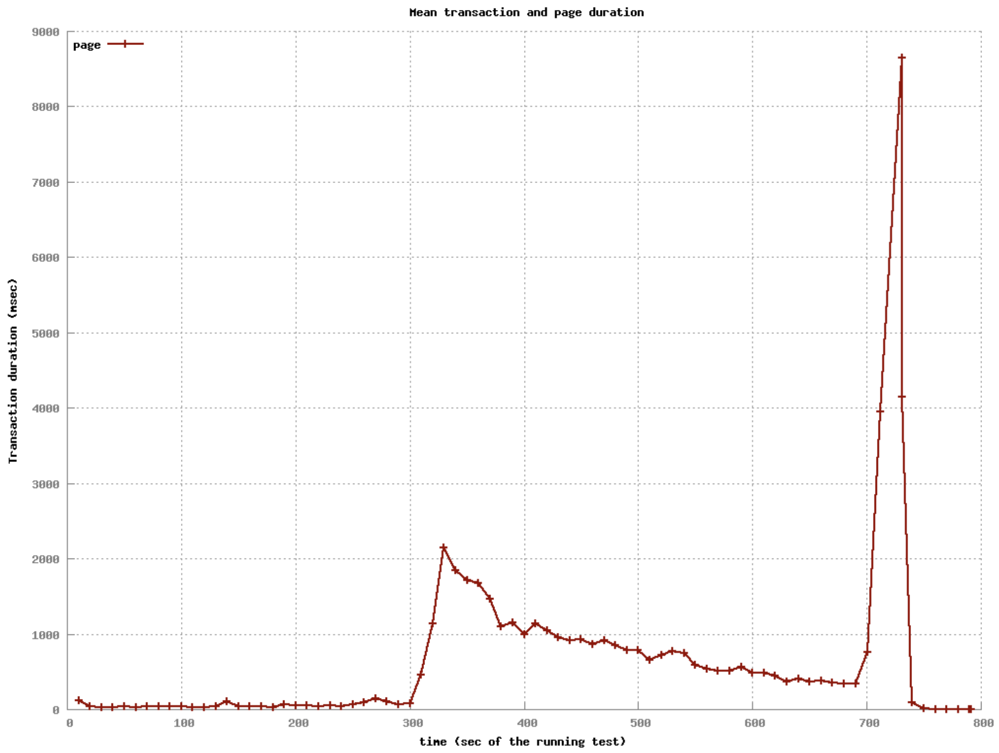
When we test we will have 12 phases, of 60 seconds each:

Phase	1	2	3	4	5	6	7	8	9	10	11	12
Users/sec	1	1.5	2	4	6	10	16	20	25	35	45	55

Remember these are users, not requests. There will be many requests per user, and users stay for 5-10s.

Deployed using nginx/passenger.

Vertical Scaling



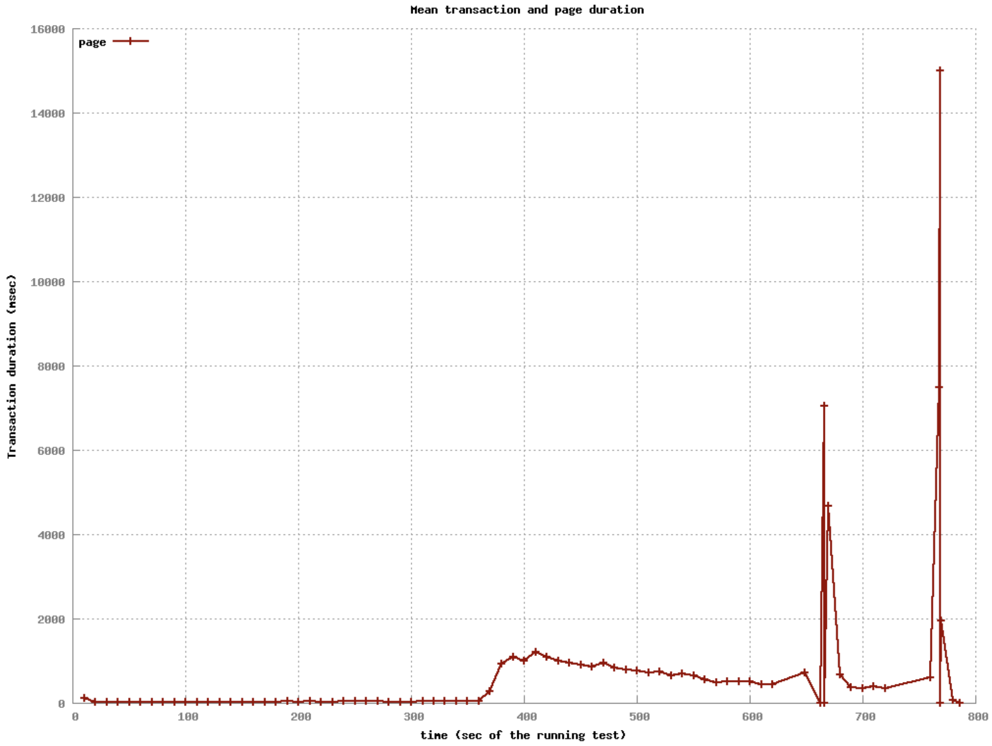
M3 Large Instance

- 2 vCPUs
- 7.5 GB Memory
- SSD storage

Responds well with 6 new users per second, but fails with 10

~$100 a month

Vertical Scaling



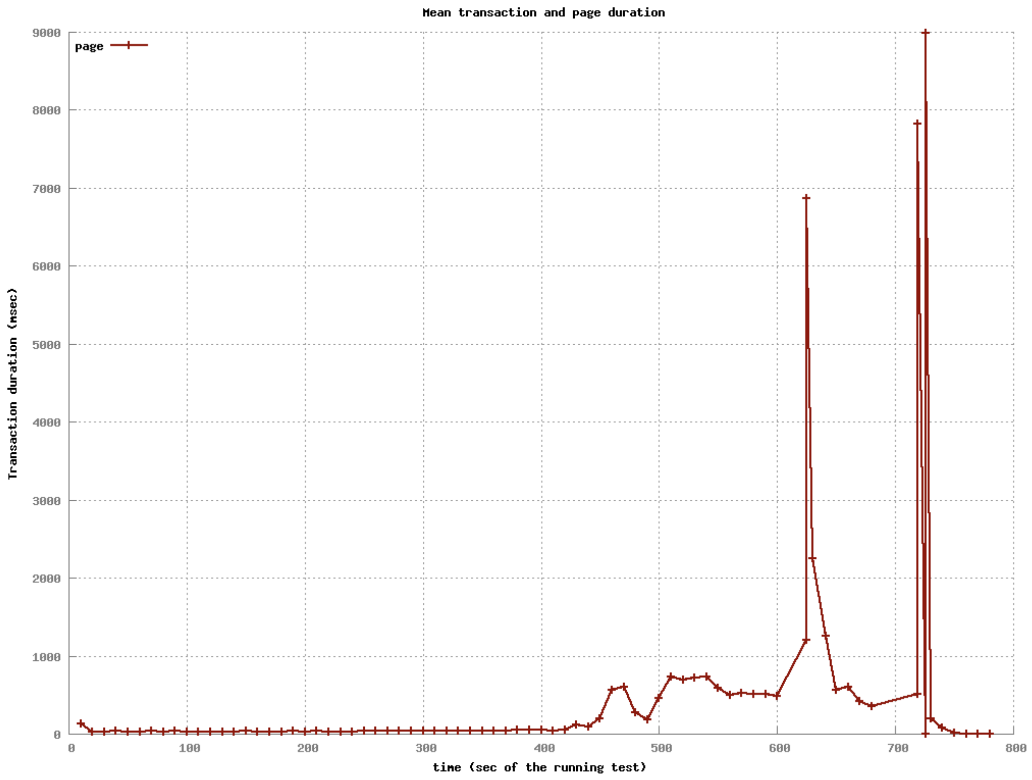
M3 X-Large Instance

- 4 vCPUs
- 15 GB Memory
- SSD storage

Responds well with 10 new users per second, but fails with 16

~$200 a month

Vertical Scaling



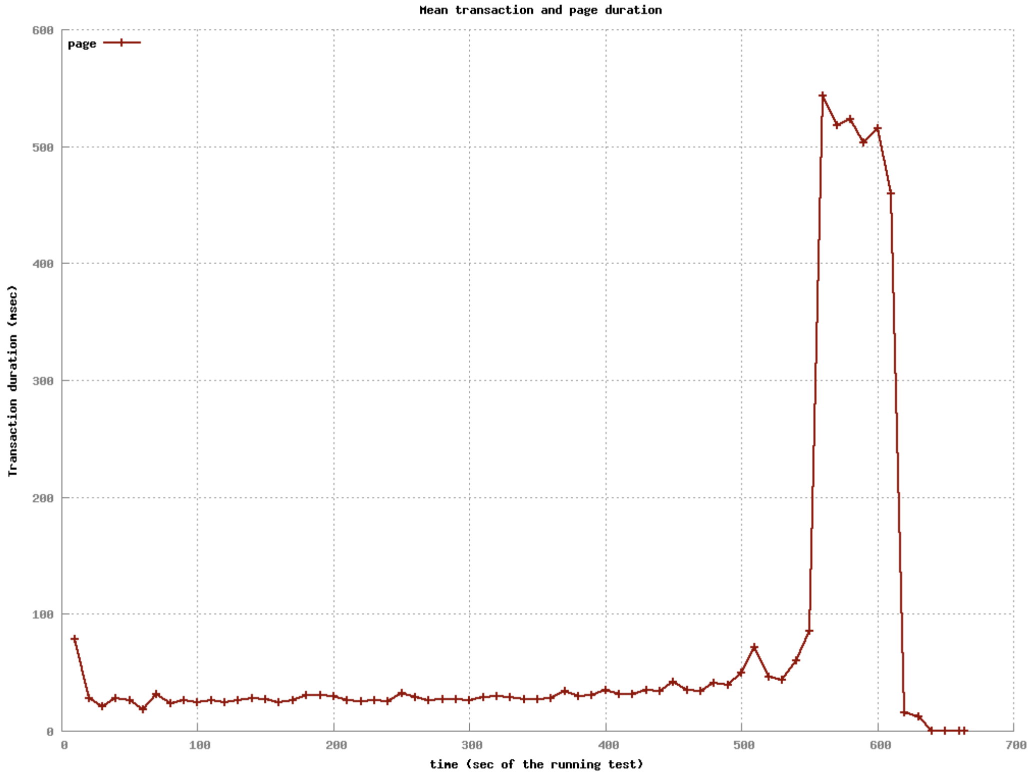
M3 XX-Large Instance

- 8 vCPUs
- 30 GB Memory
- SSD storage

Responds well with 16 new users per second, but fails with 20

~$400 a month, end of the M series.

Vertical Scaling



C3 4X-Large Instance

- 16 vCPUs
- 30 GB Memory
- SSD storage

Responds well with 20 new users per second, but fails with 25

~$600 a month,

Load Balancing

Vertical Scaling has its place, but horizontal scaling is generally preferable.

- For HTTP, this technique is referred to as load balancing.

Basic idea: have many servers that can serve clients, and make them appear as a single endpoint to the outside world

- Users experience is the same, regardless of which machine ultimately serves the request.

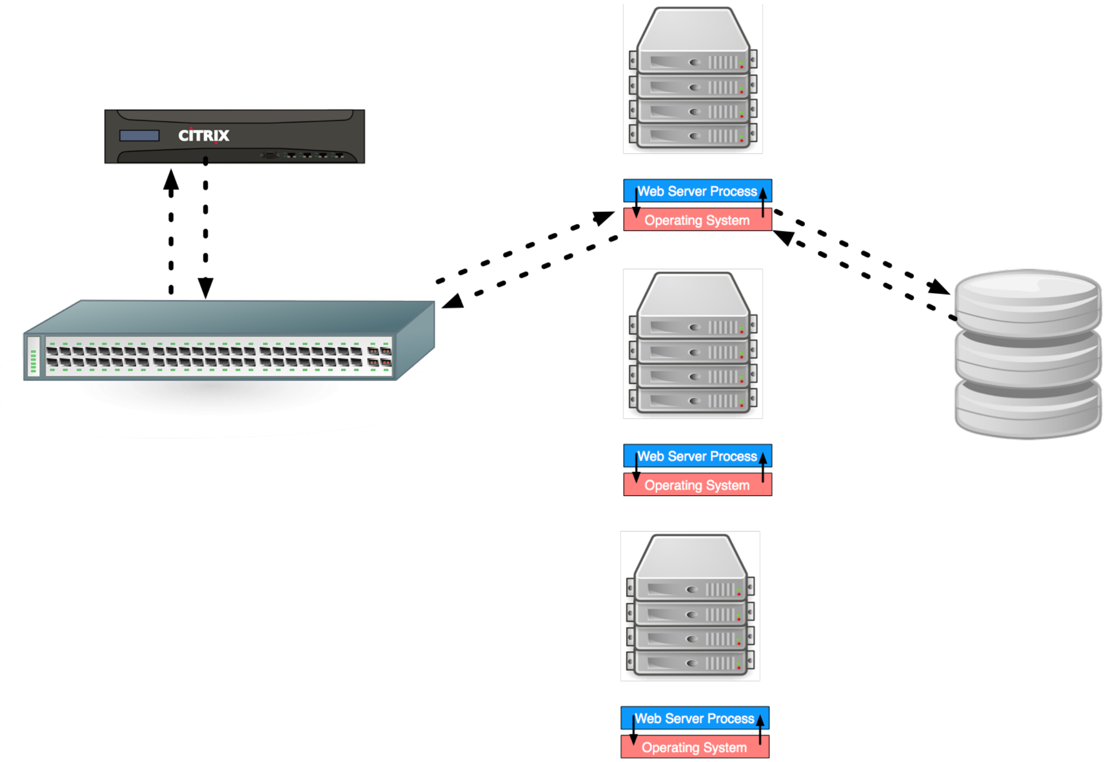
Load Balancing

Question: how can we make many servers able to serve clients?

- What if I do the following actions:
  - `GET /products`
  - `POST /products`
  - `GET /products`

If these requests are handled by three different servers, will the third request show the new product?

Load Balancing



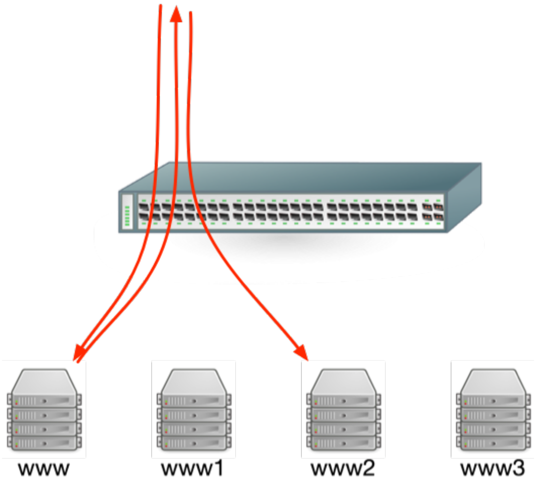
In order for load balancing to work, these servers must be stateless.

- This is generally accomplished by keeping the persistence layer separate from the application layer

Load Balancing on the Web

Idea #1: HTTP Redirects

- Discussed in section 6.1 of the reading
- Implemented using multiple web servers with different domain names
- www.domain.com uses http status 301 or 302 to redirect users to a pool of possible hosts.

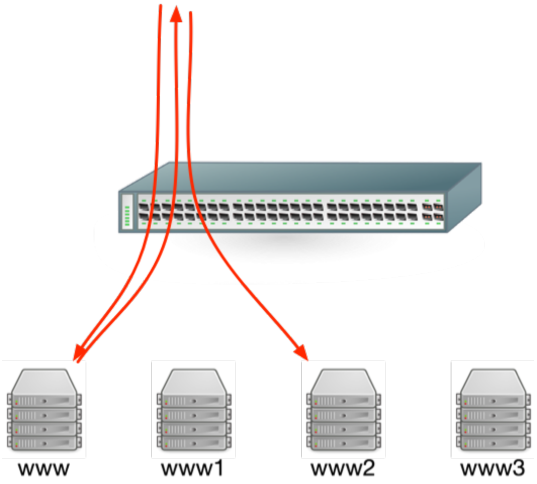


Load Balancing on the Web

```
% nc www.domain.com 80
GET / HTTP/1.1
host: www.domain.com

HTTP/1.1 301 Moved Permanently
Date: Wed, 15 Oct 2014 21:08:22 GMT
Server: Apache/2.2.22 (Ubuntu)
Location: http://www2.domain.com/
Content-Type: text/html; charset=iso-8859-1
...
```

Load Balancing on the Web

Strengths? Weaknesses?



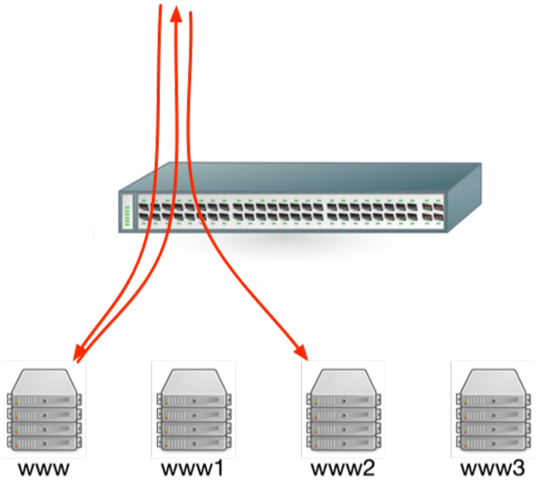
Load Balancing on the Web

Strengths:

- Simple
- Complete control over load balancing algorithm.
- Location independent.

Weaknesses:

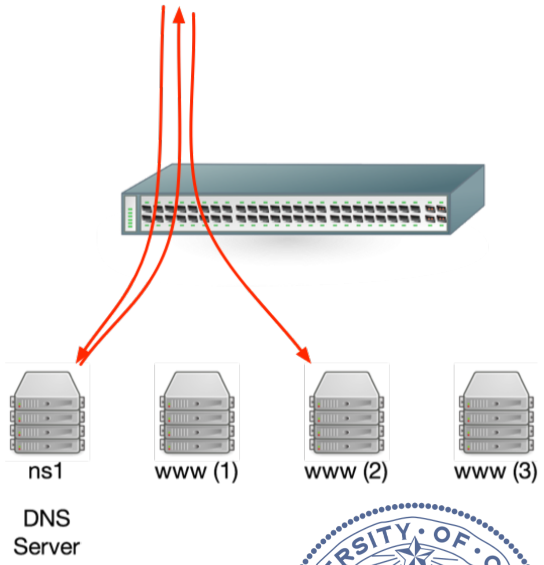
- Visible to user: URL bar, bookmarks, etc.
- www will see a lot of load.



Load Balancing on the Web

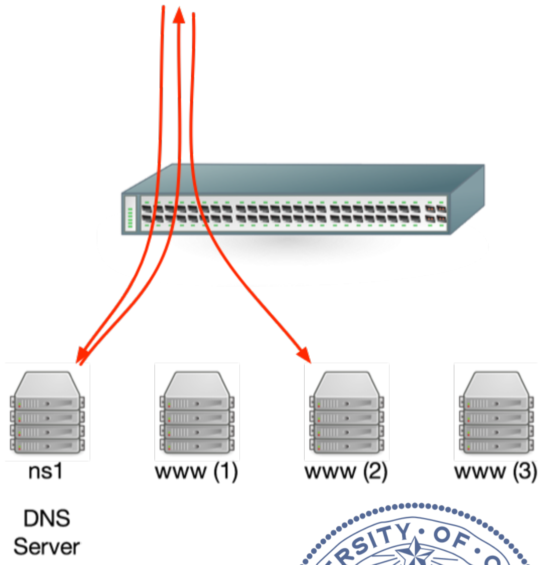
Idea #2: Round Robin DNS

- Discussed in section 4.1 of the reading
- When user's browser queries DNS for www.example.com, a list of IPs is returned.
- User's browser chooses which IP to connect to (generally the first listed)



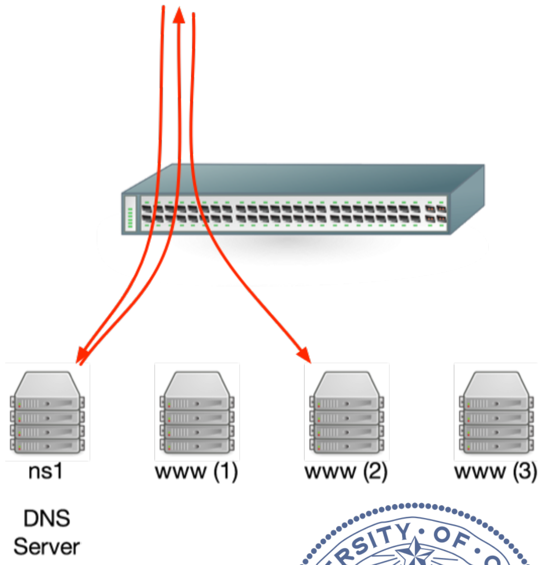
Load Balancing on the Web

```
% host www.google.com
www.google.com has address 74.125.224.48
www.google.com has address 74.125.224.51
www.google.com has address 74.125.224.52
www.google.com has address 74.125.224.49
www.google.com has address 74.125.224.50

% host www.google.com
www.google.com has address 74.125.224.49
www.google.com has address 74.125.224.52
www.google.com has address 74.125.224.51
www.google.com has address 74.125.224.50
www.google.com has address 74.125.224.48
```



Load Balancing on the Web

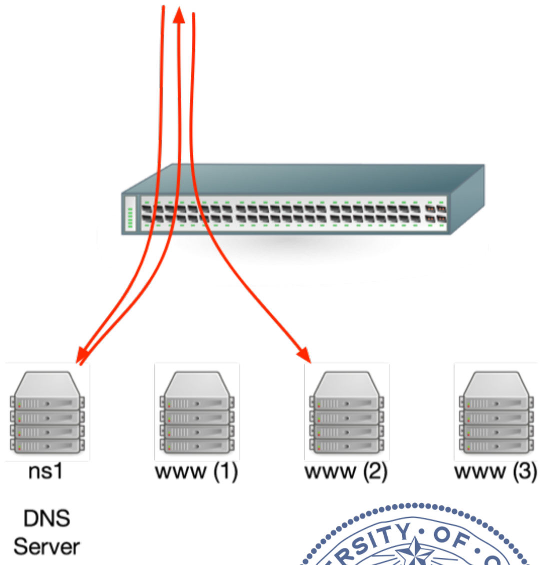
Strengths? Weaknesses?



Load Balancing on the Web

Strengths:

- Easy
- Cheap
- Simple



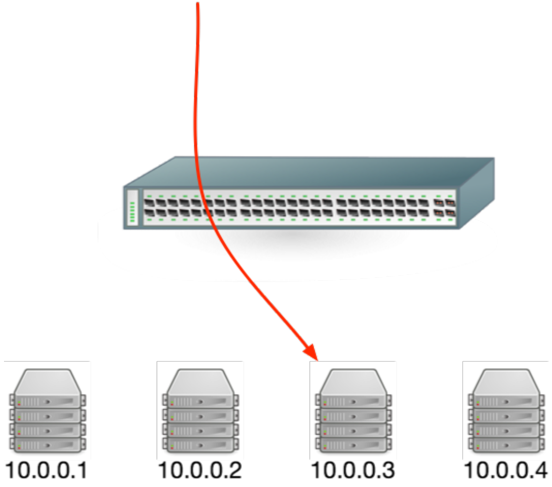
Weaknesses:

- Less control over balancing
- Modifying the list is inhibited by caching in the browser and proxies

Load Balancing on the Web

Idea #3: Load Balancing Switch

- Discussed in section 4.1 of the reading
- AKA: “TCP Load Balancing”
- The idea: rewrite TCP packets to send them to the correct server
  - Addresses, sequence numbers and checksums need to be rewritten on the fly
  - Constructed using hardware (ASICs) to perform this fast
- Commercial products available:
  - Cisco Content Services Switch
  - Citrix Netscaler
  - F5 Big IP



Load Balancing on the Web

```
__ client -> switch : 216.64.159.149 -> 208.50.157.136
IP D=208.50.157.136 S=216.64.159.149 LEN=60, ID=48397 2.94950 216.64.159.149 -> 208.50.157.136
TCP D=80 S=1421 Syn Seq=899863543 Len=0 Win=32120 …

__ switch -> client : 208.50.157.136 -> 216.64.159.149
IP D=216.64.159.149 S=208.50.157.136 LEN=48, ID=26291 2.95125 208.50.157.136 -> 216.64.159.149
TCP D=1421 S=80 Syn Ack=899863544 Seq=1908949446 Len=0 ...

__ client -> switch : 216.64.159.149 -> 208.50.157.136
IP D=208.50.157.136 S=216.64.159.149 LEN=40, ID=48400 2.98324 216.64.159.149 -> 208.50.157.136
TCP D=80 S=1421 Ack=1908949447 Seq=899863544 Len=0 ...

__ client -> switch : 216.64.159.149 -> 208.50.157.136
IP D=208.50.157.136 S=216.64.159.149 LEN=154, ID=48401 2.98395 216.64.159.149 -> 208.50.157.136
TCP D=80 S=1421 Ack=1908949447 Seq=899863544 Len=114 ... 2.98395 216.64.159.149 -> 208.50.157.136
HTTP GET /eb/images/ec_home_logo_tag.gif HTTP/1.0
```

Load Balancing on the Web

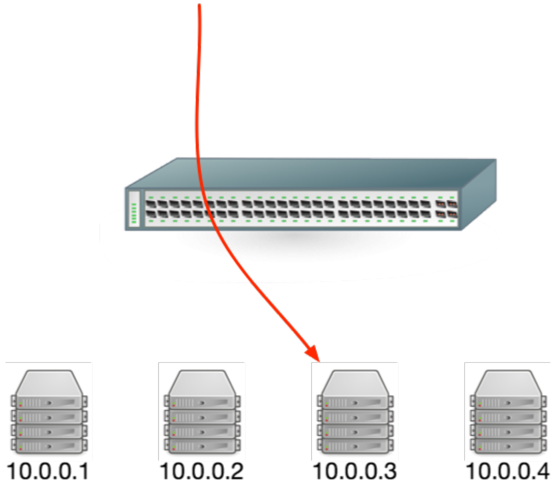
```
__ switch -> server : 216.64.159.149 -> 10.16.100.121
IP D=10.16.100.121 S=216.64.159.149 LEN=48, ID=26292 0.00000 216.64.159.149 -> 10.16.100.121
TCP D=80 S=1421 Syn Seq=899863543 Len=0 Win=32120 Options...

__ server -> switch : 10.16.100.121 -> 216.64.159.149
IP D=216.64.159.149 S=10.16.100.121 LEN=44, ID=22235 0.00001 10.16.100.121 -> 216.64.159.149
TCP D=1421 S=80 Syn Ack=899863544 Seq=2156657894 Len=0 ...

__ switch -> server : 216.64.159.149 -> 10.16.100.121
IP D=10.16.100.121 S=216.64.159.149 LEN=154, ID=48401 0.00131 216.64.159.149 -> 10.16.100.121
TCP D=80 S=1421 Ack=2156657895 Seq=899863544 Len=114 ... 0.00131 216.64.159.149 -> 10.16.100.121
HTTP GET /eb/images/ec_home_logo_tag.gif HTTP/1.0
```

Load Balancing on the Web

```
__ server -> switch : 10.16.100.121 -> 216.64.159.149
IP D=216.64.159.149 S=10.16.100.121 LEN=40, ID=22236 0.00134 10.16.100.121 -> 216.64.159.149
TCP D=1421 S=80 Ack=899863658 Seq=2156657895 Len=0 …

__ switch -> client : 208.50.157.136 -> 216.64.159.149
IP D=216.64.159.149 S=208.50.157.136 LEN=40, ID=22236 2.98619 208.50.157.136 -> 216.64.159.149
TCP D=1421 S=80 Ack=899863658 Seq=1908949447 Len=0 ...

__ server -> switch : 10.16.100.121 -> 216.64.159.149
IP D=216.64.159.149 S=10.16.100.121 LEN=1500, ID=22237 0.00298 10.16.100.121 -> 216.64.159.149
TCP D=1421 S=80 Ack=899863658 Seq=2156657895 Len=1460 ... 0.00298 10.16.100.121 -> 216.64.159.149
HTTP HTTP/1.1 200 OK

__ switch -> client : 208.50.157.136 -> 216.64.159.149
IP D=216.64.159.149 S=208.50.157.136 LEN=1500, ID=22237 2.98828 208.50.157.136 -> 216.64.159.149
TCP D=1421 S=80 Ack=899863658 Seq=1908949447 Len=1460 ... 2.98828 208.50.157.136 -> 216.64.159.149
HTTP HTTP/1.1 200 OK
```

Load Balancing on the Web

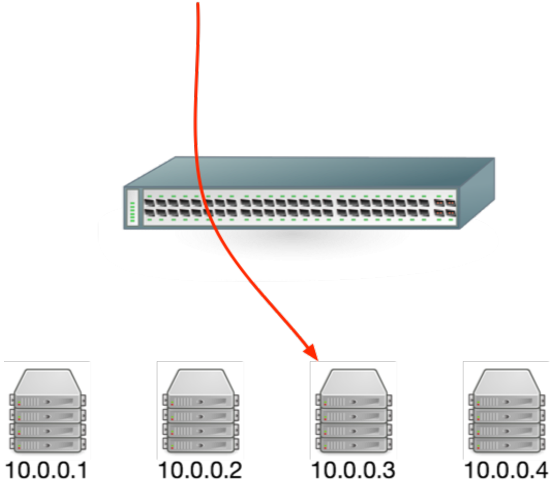
Strengths? Weaknesses?



Load Balancing on the Web

Strengths:

- More control over which requests go to which servers
- Works fine with HTTPS



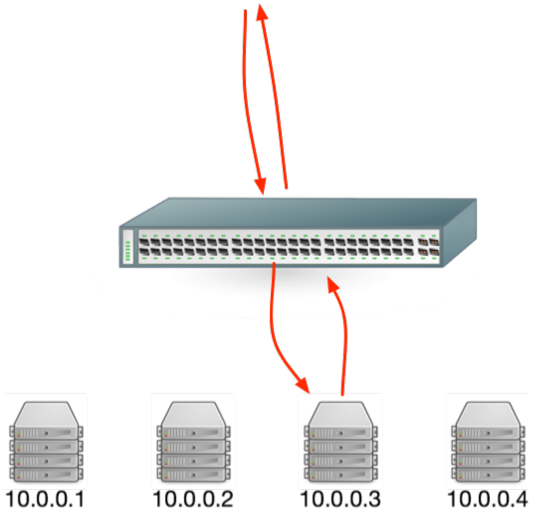
Weaknesses:

- Constrains where the servers are
- Complicated

Load Balancing on the Web

Idea #4: Load Balancing Proxy

- Also known as "Layer 7 load balancing"
- Terminate HTTP requests: act like a web server
- Issue "back-end" HTTP requests to real web servers to get responses.
- Many hardware products available:
  - Citrix Netscaler
  - BigIP F5
- Most HTTP servers have modules to do this as well.



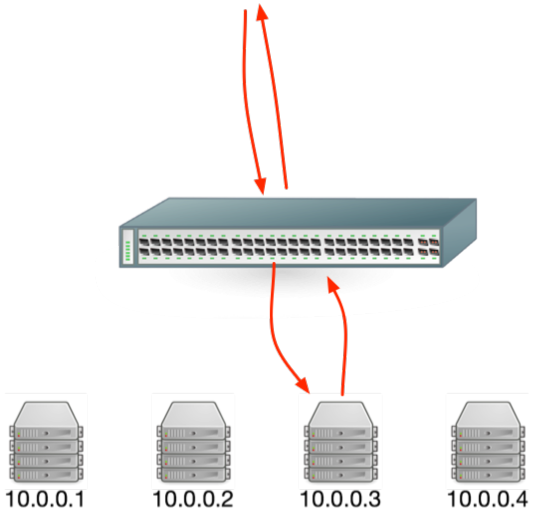
Load Balancing on the Web

Strengths:

- Cleaner implementation than packet rewriting

Weaknesses:

- Load balancer needs to be doing more work
- SSL is more complicated



This is the most popular technique in use today.

- Technique #3 also used as well

Load Balancing on the Web

Let's say you are designing a load balancing proxy…

- You see every request go in and out.
- What algorithm would you choose for balancing between machines?

Load Balancing on the Web

Let's say you are designing a load balancing proxy...

- You see every request go in and out.
- What algorithm would you choose for balancing between machines?
  - Random
  - Round robin
  - Least number of connections
  - Fastest response time
  - Bandwidth per Server
  - Based on URI (e.g. /images, or /cgi-bin)

- ELB uses Round Robin with cookie-based stickiness

Load Balancing on the Web

Some interesting challenges come up when designing a load balancing proxy…

- Detecting Server Failures
- Session persistence & affinity
- Connection pooling

Load Balancing on the Web

Detecting Server Failures

How do we know when a member of our pool has died?

- We can observe traffic: are requests being serviced? Some requests just take a long time.
- We can probe the server
  - Various protocols
    - ICMP ping: test network and kernel
    - TCP connection set up: process is running
    - HTTP HEAD: it’s serving pages
    - SNMP: server load

Load Balancing on the Web

Session persistence & affinity

Can we redirect users back to the same web server they used before?

- This can get us caching improvements
- A few options:
  - Affinity based on client IP address
    - Client IPs change & IPs can be shared
  - HTTP Cookie
    - Works, but needs proxy configuration
  - Session ID in URL
    - Hard to parse out in a load balancer (http://..../.../.../...?...&...&SID=01234&...)
    - Your sessions are in your URL??

Load Balancing on the Web

Connection Pooling

- We’re used to the idea that one client reuses a TCP connection for many HTTP requests.
- We can reuse this mechanism for distinct clients
- Saves on repeated TCP setup
- Reduce idle waiting on server for reads and writes

Load Balancing on the Web

Common Load Balancers

Software

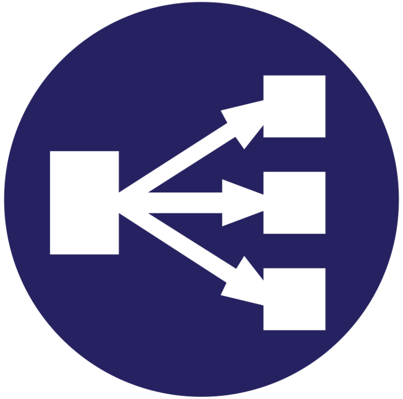
- Apache
- HAProxy
- Varnish
- Pound
- Squid
- Nginx
- ....

Hardware

- F5 Big IP
- Citrix Netscaler
- Cisco

Load Balancing on the Web

Amazon Elastic Load Balancer



Load balancing as a service

- $0.025 per hour plus $0.008 per GB processed
- Works with EC2 autoscaling
- Supports SSL termination

Can load balance between Availability Zones within a Region

Can't load balance between regions. Rely on Route 53 DNS for region failover.

Horizontal Scaling

We will use the same testing script from before...

1. Going to the homepage
2. Waiting for up to 2 seconds
3. Requesting a form to create a new community
4. Waiting for up to 2 seconds
5. Submitting the new community
6. Requesting a form to create a new link submission
7. Waiting for up to 2 seconds
8. Submitting the new link
9. Waiting for up to 2 seconds
10. Delete the link
11. Waiting for up to 2 seconds
12. Delete the community

Horizontal Scaling

When we test we will have 12 phases, of 60 seconds each:

Phase	1	2	3	4	5	6	7	8	9	10	11	12
Users/sec	1	1.5	2	4	6	10	16	20	25	35	45	55

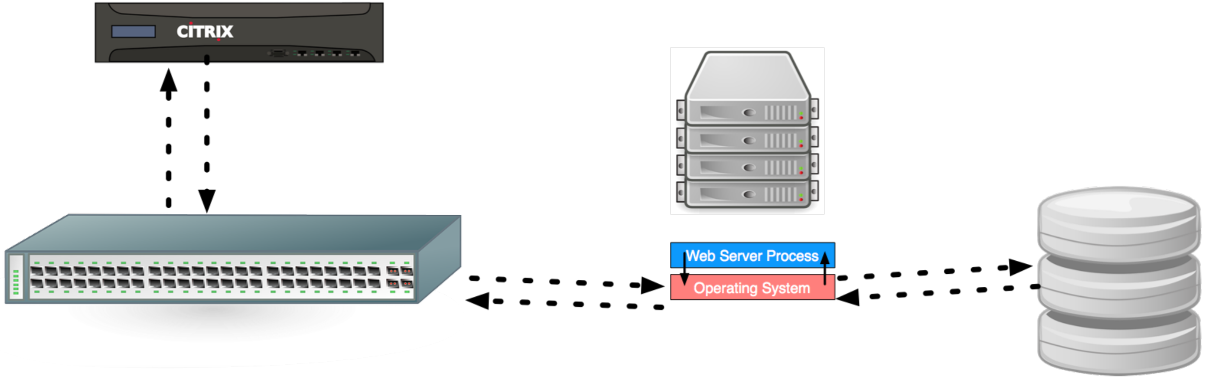
Remember these are users, not requests. There will be many requests per user, and users stay for 5-10s.

Deployed using nginx & passenger.

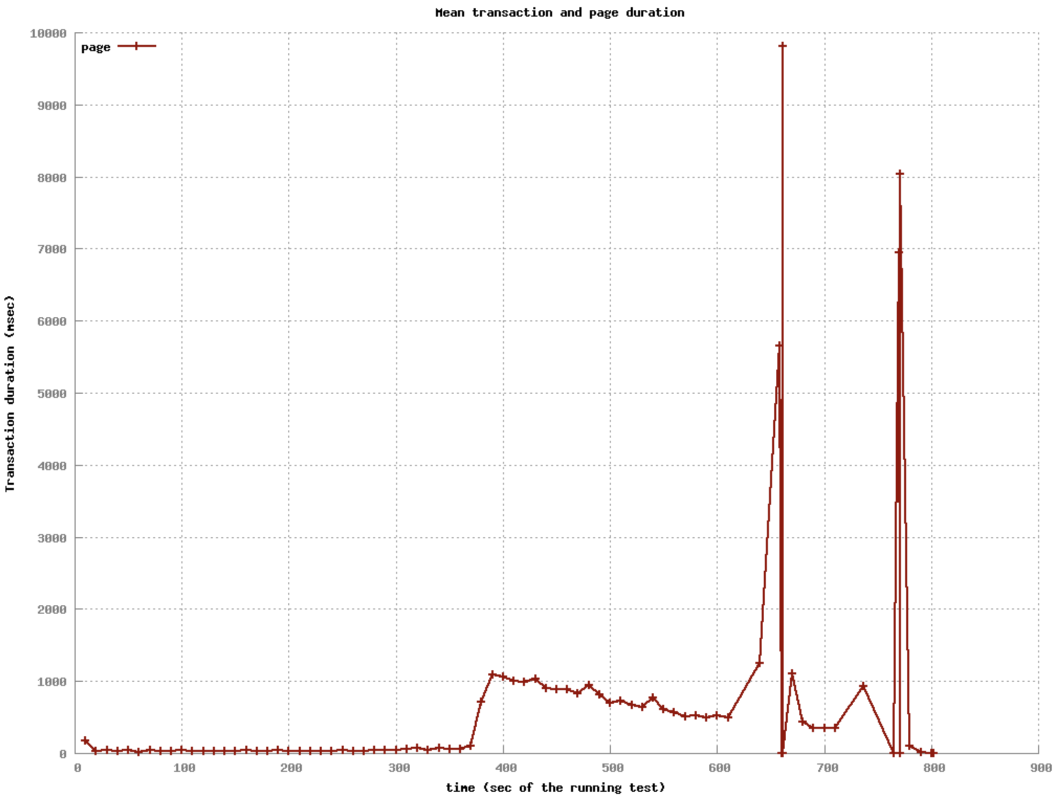
Horizontal Scaling

First we will look at breaking our database away from our single instance.

This isn't horizontal scaling yet, but this change enables us to scale up the number of app servers



Horizontal Scaling



1 M3 Large Instance App server

- 2 vCPUs
- 7.5 GB Memory
- SSD storage

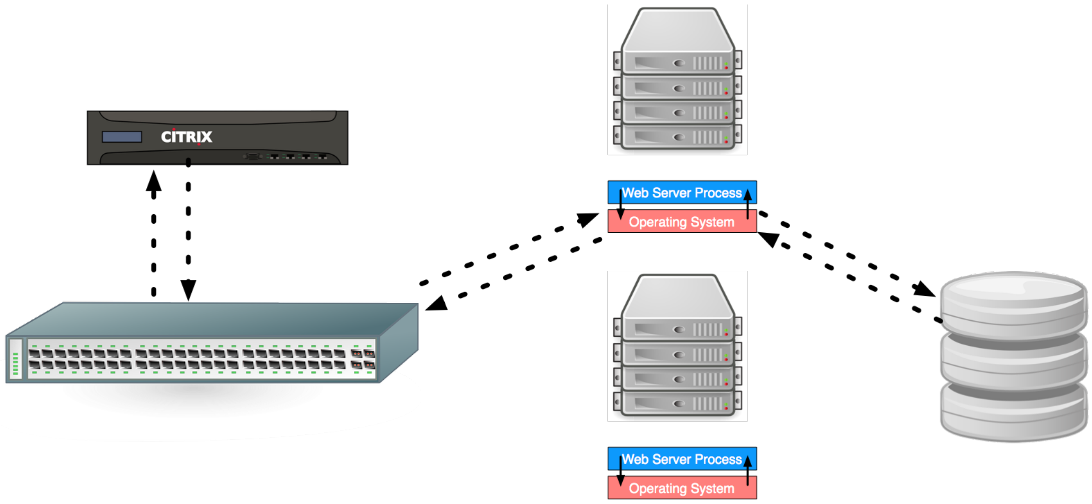
DB is also M3 Large.

Responds well with 10 new users per second, but fails with 16

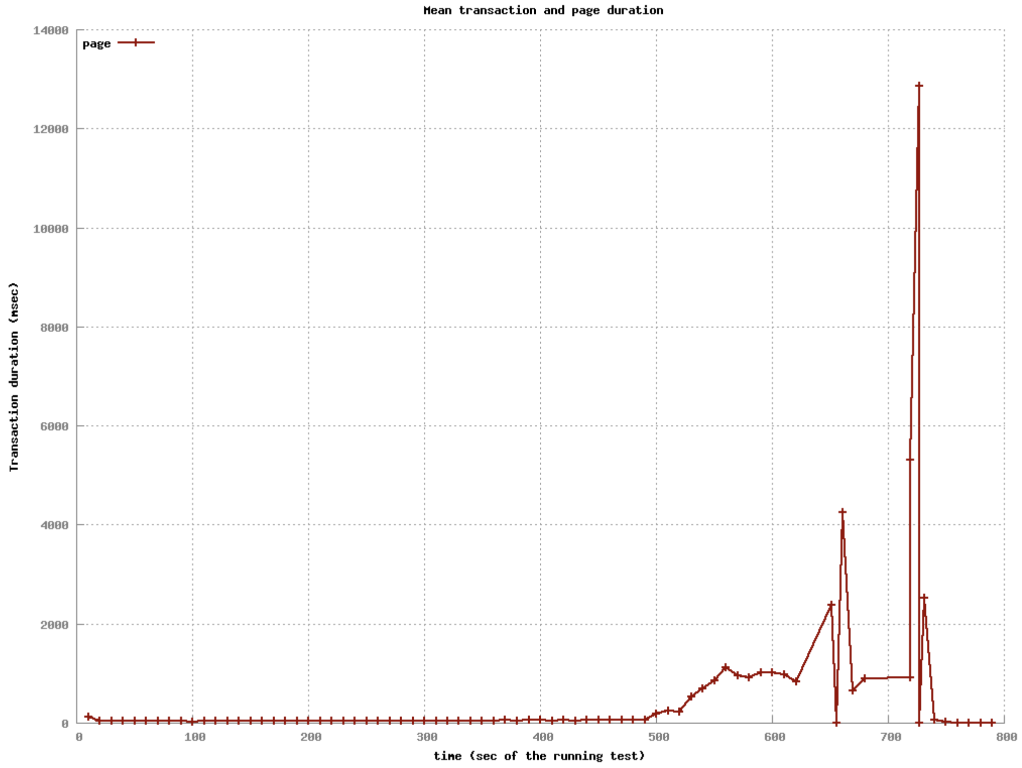
~$200 a month

Horizontal Scaling

Let's horizontally scale our app server by adding a second app server instance and see the effect on system performance



Horizontal Scaling



2 M3 Large Instance App server

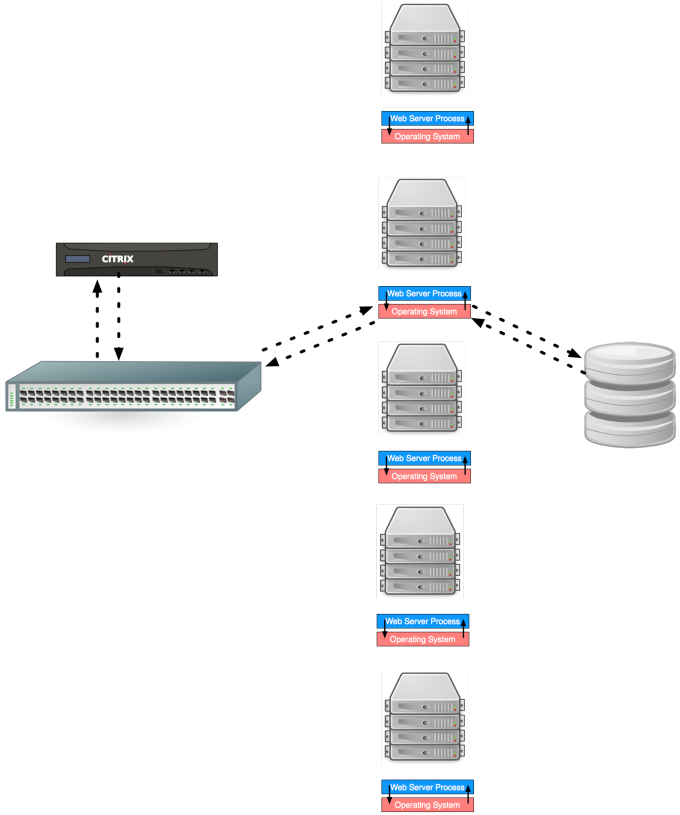
- 2 vCPUs
- 7.5 GB Memory
- SSD storage

DB is also M3 Large.

Responds well with 20 new users per second, but fails with 25

~$300 a month

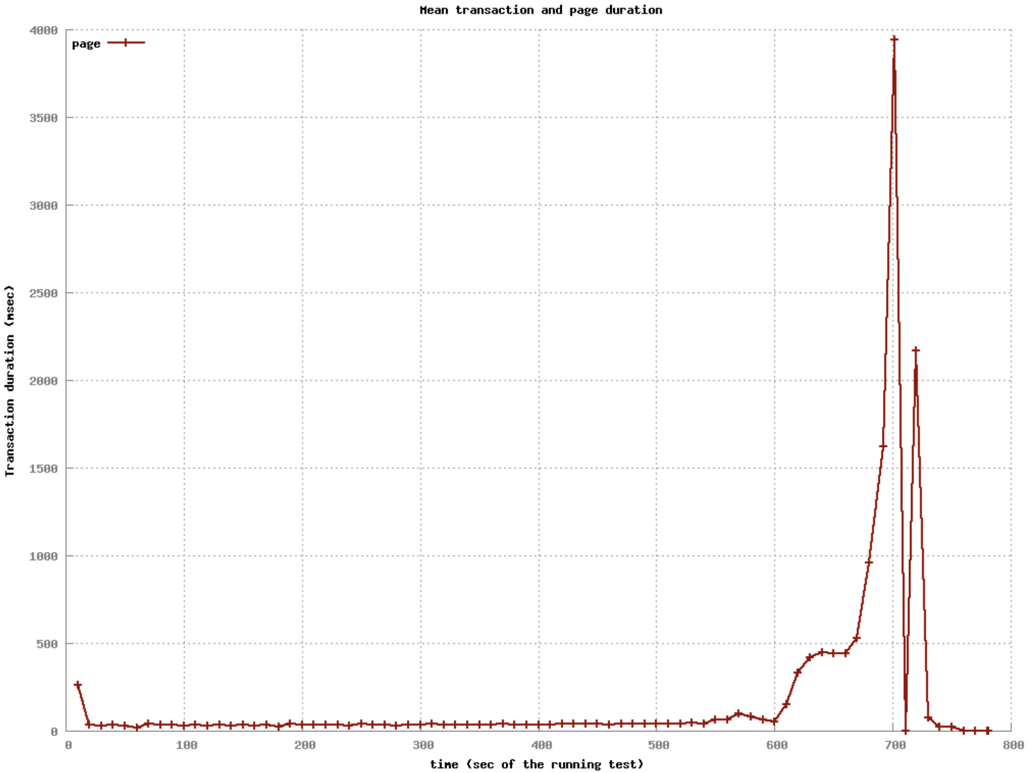
Horizontal Scaling



Let's see how the system performs if we scale out to 5 app servers.

Total dollar cost will be comparable to the most expensive single instance we tested

Horizontal Scaling



2 M3 Large Instance App server

- 2 vCPUs
- 7.5 GB Memory
- SSD storage

DB is also M3 Large.

Responds well with 35 new users per second, does okay with 45, fails with 55

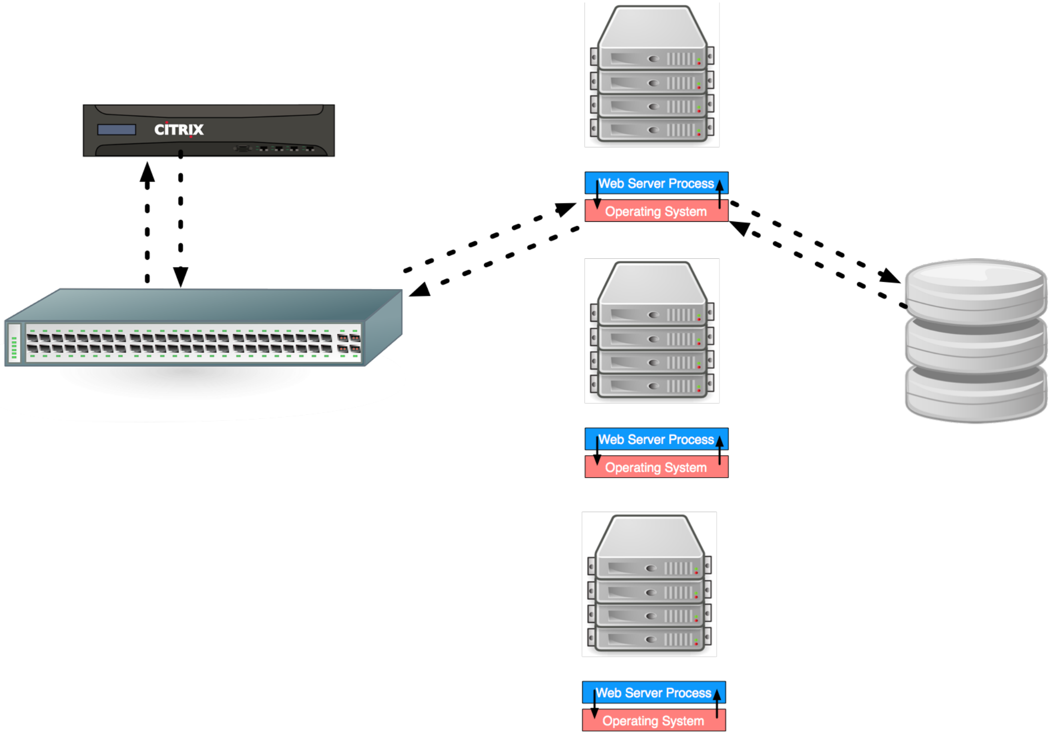
~$600 a month

Horizontal Scaling

Vertical scaling handled 20 new users per second for ~$600 per month, and left us with a nonexistent scaling path.

Horizontal scaling handled 45 new users per second for ~$600 per month and left us with a clear path to increase scale

Motivation



After today's lecture you should understand what load balancing is, and and why the architecture depicted here allows cost effective and smooth scaling paths.

For Next Time...

Email me your group. Required info:

- Team name
- Each person's name, email, github acct.

Read/do chapters 9 through 17 in AWDR by next Tuesday.

Lab is tomorrow, I will meet with each group to discuss projects.

If you need technical help, come to Franz 1260 Friday's noon to 2pm.