Scalability of web applications
Overview

• Scalability questions
  – What's important in order to build scalable web sites?

• High availability vs. load balancing

• Approaches to scaling
  – Performance tuning, horizontal scaling, vertical scaling

• Multiple web servers
  – DNS based sharing, hardware/software load balancing

• State management

• Database scaling
  – Replication
  – Splitting things up
Scalability related questions

• Where is your session state being stored? Why?
• How are you generating dynamic content? Why?
• Are you regenerating things that could be cached?
• What is being stored in the database? Why?
• Could you be lazier?
  – Do you need exact answers?
    • e.g. “page 1/2063” versus “page 1 of many”
  – Queue up work if it doesn't need to be done right now
    • e.g. Does user really need a video thumbnail right away?
• What do you care about?
  – Time to market, money, user experience, uptime, power efficiency, bug density, ...
High availability / load balancing

• **High availability**
  – Stay up despite failure of components
  – May involve load-balancing, but not necessarily
    • Hot standby = switched to automatically if primary fails
    • Warm standby = switched to by engineer if primary fails
  – Easy component updates
    • e.g. Avoid maintenance windows in the middle of the night

• **Load balancing**
  – Combining resources from multiple systems
  – Send request to somebody else if a certain system fails
  – May provide high availability, but not necessarily
    • e.g. Adding a single-point of failure load balancing appliance
### Availability 9s

<table>
<thead>
<tr>
<th>Availability %</th>
<th>Downtime per year</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>&quot;one nine&quot;</td>
</tr>
<tr>
<td>99%</td>
<td>&quot;two nines&quot;</td>
</tr>
<tr>
<td>99.9%</td>
<td>&quot;three nines&quot;</td>
</tr>
<tr>
<td>99.99%</td>
<td>&quot;four nines&quot;</td>
</tr>
<tr>
<td>99.999%</td>
<td>&quot;five nines&quot;</td>
</tr>
<tr>
<td>&quot;carrier grade&quot;</td>
<td></td>
</tr>
<tr>
<td>99.9999%</td>
<td>&quot;six nines&quot;</td>
</tr>
<tr>
<td>99.99999%</td>
<td>&quot;seven nines&quot;</td>
</tr>
</tbody>
</table>

99.99% Uptime

We provide a 99.99% uptime SLA around network, power and virtual server availability. If we fail to deliver, we'll credit you based on the amount of time that service was unavailable.

GET STARTED

https://www.digitalocean.com/features/reliability/
Approaches to scaling

• **Make existing infrastructure go further**
  – Classic **performance tuning**:
    • Find the bottleneck
    • Make faster (if you can)
    • Find the new bottleneck, iterate
  – How are you generating **dynamic content**? Why?
  – Where is your **session state** being stored? Why?
  – What is being stored in the **database**? Why?
  – Can you **be lazier**?
    • Do you need exact answers?
      – e.g. “page 1/2063” versus “page 1 of many”
    • Add work to a queue if it doesn't need to be done right now
      – e.g. Does user really need a video thumbnail right away?
Approaches to scaling

• **Vertical scaling (scale up)**
  - Buy more memory, faster CPU, more cores, SSD disks
  - A quick fix: uses existing software/network architecture
  - But there are performance limits
    - Also a price premium for high end kit

**ABMX server, 1u**
1 core @3.1 Ghz, 1GB memory, 80GB disk
$397

**Oracle Exadata X2-8, 42u**
160 cores @ 2.4Ghz, 4TB memory
14 storage servers, 168 cores, 336TB
1.5M database I/O ops/sec
$1,650,000
Approaches to scaling

• **Horizontal scaling (scale out)**
  – Buy more servers
  – Well understood for many parts
    • Application servers (e.g. web servers)
    • But may require software and/or network changes
  – Not so easy for other parts
    • Databases

http://www.flickr.com/photos/intelfreepress/6722296265/
One web site: many servers

• How does the user arrive at a particular server?
  – Does the session need to “stick” to same web server?
    • Very important depending on how app manages state
    • e.g. using PHP file-based session state
  – What happens if a web server crashes?
  – Users would prefer a geographically nearby server
Round robin DNS

- Round robin DNS
  - Multiple IP addresses assigned to a single domain name
  - Client's networking stack chooses which to connect to

```
Queries
- cnn.com: type A, class IN
  Name: cnn.com
  Type: A (Host address)
  Class: IN (0x0001)

Answers
- cnn.com: type A, class IN, addr 157.166.226.25
- cnn.com: type A, class IN, addr 157.166.226.26
- cnn.com: type A, class IN, addr 157.166.255.18
```
Round robin DNS

- **Round robin DNS**
  - Simple and cheap to implement
    - No specialized hardware, using existing DNS infrastructure
  - Problems:
    - DNS has **no visibility into server load or availability**
    - In simplest configuration, each web server requires an IP address
    - Users may end up being sent to a **distant server with high latency**

Browser A

Browser B

Web 1 157.166.226.25

Web 2 157.166.226.26

Web 3 157.166.255.18

DB
Anycast + DNS

• **Goal:** Get users to the "closest" server

• **Anycast = multiple servers with same IP address**
  
  – Routing protocols determine best route to shared IP
  
  – Best suited for connectionless protocols
    
    • e.g. UDP
Anycast + DNS

- **Multiple clusters**
  - Place a DNS server next to each web cluster
    - Each DNS server has same IP address via IP Anycast
    - A particular DNS server gives out IPs in its local cluster
  - Anycast routes client to closest DNS server
    - DNS servers routes client to "closest" server farm
Load balancers

- **Load balancers (web switches)**
  - Hardware or software (e.g. mod_proxy_balancer, Varnish)
  - Like a NAT device in reverse
    - People hit a single public IP to get to multiple private IP addresses
  - Introduces a new single point of failure
    - But we can introduce a backup balancer
    - Load balancers monitor each other via a heartbeat
- How to distribute load?
  - Round robin, least connections, predictive, available resources, random, weighted random
Load balanced, no single point of failure

Internet

router

switch

web switch

switch

www 1

router

switch

web switch

switch

www 2
Load balancer, some features

- **Session persistence**
  - Getting user back to same server (e.g. via cookie/client IP)
- **Asymmetric load**
  - Some servers can take more load than others
- **SSL offload**
  - Load balancer terminates the SSL connection
- **HTTP compression**
  - Reduce bandwidth using gzip compression on traffic
- **Caching content**
- **Intrusion/DDoS protection**
Software load balancer

- Apache server running mod_proxy_balancer
  - One server answers user requests
  - Distributes to two or more other servers

```xml
<Proxy balancer://mycluster>
  BalancerMember http://192.168.1.50:80
  BalancerMember http://192.168.1.51:80
</Proxy>
ProxyPass /test balancer://mycluster

Example configuration without sticky sessions.

```xml
Header add Set-Cookie
"ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/"
env=BALANCER_ROUTE_CHANGED
<Proxy balancer://mycluster>
  BalancerMember http://192.168.1.50:80 route=1
  BalancerMember http://192.168.1.51:80 route=2
  ProxySet stickysession=ROUTEID
</Proxy>
ProxyPass /test balancer://mycluster

Example configuration with sticky sessions.
State management

• HTTP is *stateless*, but user interactions often *stateful*

• Store session state somewhere:
  – Local to web server
  – Centralized across servers
  – Stored in the client
  – Or some combination
    • Centralized but cached at closer level(s)
Local sessions

• Stored on disk
  – PHP temp file somewhere

• Stored in memory
  – Faster
  – PHP:
    • Compile with --with-mm
    • session.save_handler=mm in php.ini

• Problems:
  – User can't move between servers
    • Load balancer must always send user to same physical server
  – User's session won't survive a server failure
    • Switching to new server results in loss of client's state
Centralized sessions

• User can move freely between servers
  – But always need to pull info from central store

• Web servers can crash
  – User gets routed to another web server

• Approaches
  – Shared file system
  – Store in a database
  – Store in an in-memory cache
    • e.g. Memcached
No sessions

• Put all information in the cookie
• Ultimate in horizontal scalability
  – Browser "nodes" scale with your users
  – Free!
• Concerns:
  – User may delete cookie
  – User may modify cookie
    • But you can encrypt and digitally sign
  – Limits on amount of data
  – Local to the browser, user may use multiple browsers
Database scaling

- **Scaling databases is hard**
  - Distribute among many servers to maintain performance
  - DB must obey **ACID** principles:
    - **Atomicity** - transactions are all or none
    - **Consistency** - transactions go from one valid state to another
    - **Isolation** - no transaction can interfere with another one
    - **Durability** - on failure, information must be accurate up to the last committed transaction
  - **ACID isn't too hard/expensive on a single machine:**
    - Using: shared memory, interthread/interprocess synch, shared file system
    - Facilities are fast and reliable
  - Distribute over a LAN or WAN, big performance problems!
Database replication

• **Multimaster replication**
  – The “holy grail” of distributed databases
  – Group of DBs, updates can occur on any DB
  – BUT: doing this without loosening ACID, very expensive
    • Two-phase commit between all the nodes
      – Node attempting transaction notifies peers it is about to commit
      – Peer prepare transaction and notify node they are ready to commit
      – If everybody ready, node informs peers to commit

• **Master-master replication**
  – For high-availability, not scalability
  – Two servers connected via a low latency network

• **Master-slave replication**
  – Mods only occur on master, changes propagate to slaves
  – Can offer read-intensive applications linear speedups
Database example

Remote cluster 1

Core web app

Slave DB

Content web app

Remote cluster 1

Core web app

Slave DB

Content web app
Other database options

• **Horizontal partitioning**
  – Separate rows into separate tables
  – Spreads read/writes, improves cache locality

• **Vertical partitioning**
  – Split rows into multiple tables with fewer columns
  – Allows queries to scan less data
    • Unless you end up needing to do a join across tables

• **Sharding**
  – Separate rows onto separate databases
    • e.g. All customers west of the Mississippi
  – Must determine which shard customer belongs to
  – Trouble for queries/transactions involving multiple shards
Summary

• **Scaling web sites**
  – High availability != load balancing
  – Scale vertically
  – Scale horizontally
    • More application servers
    • Balanced via DNS/hardware/software
    • Session management becomes harder
  – The database is usually the big problem