Content-distribution networks
Strategies

- Divide and conquer
  - Replicate
  - Partition
  - Distribute
  - Load balance
Outline

1. Server partitioning
2. DNS load balancing
3. Virtual servers
4. Case studies
1. Server partitioning (static)

- Run a new server per resource/service
  - e.g. www.blah.com, mail.blah.com, images.blah.com, shopping.blah.com

- Advantages
  - Disk utilization (no need to replicate all content)
  - Cache performance
  - Better suited for DevOps, CI/CD
    - Cookie policy, Content Security Policy, distributed independent development/deployment etc.

- Disadvantages
  - Without cloud provider support, you get...
    - Lower peak capacity if access to sites imbalanced
    - Coarse load balancing, not adaptive to spikes
  - Management costs of multiple sites
1. Server partitioning (dynamic)

- Active “forward deployment” of content to explicitly named servers near client
- Redirect requests from origin servers via dynamic URL rewriting of embedded content
- Application-level multicast based on geographic location of client
  - Example: Akamai
1. Server partitioning (dynamic)

Dynamically loaded content servers

pdx.edu

espn.go.com

Local, high-speed ISP

Requested page with links to embedded content rewritten OR an HTTP redirect
1. Server partitioning (dynamic)

- Advantages
  - Improved network utilization
  - Cost savings ($ network bandwidth >> $ storage)
  - Better load distribution if replicas based on popularity

- Disadvantages
  - Distributed management costs
  - Complexity in integration with CDN provider
2. DNS load balancing

- Popularized by NCSA circa 1993
- Fully replicated server farm
- IP address per node
- Adaptively resolve server name (round-robin, load-based, or geographic-based)
2. DNS load balancing

DNS cache
Host: www.ncsa.uiuc.edu ttl=15min
DNS ns0.ncsa.uiuc.edu ttl=3days

[a-m].root-servers.net
*.ncsa.uiuc.edu is served by
ns0.ncsa.uiuc.edu (141.142.2.2)
ns1.ncsa.uiuc.edu(141.142.230.14)
dns1.cso.uiuc.edu (128.174.5.103)
ns.indiana.edu (129.79.1.1)
2. DNS load balancing

- Advantages
  - Simple to implement
  - Uses existing DNS infrastructure

- Disadvantages
  - Coarse load balancing
  - DNS caching at local name servers affects performance
  - Requires full server replication
3. Virtual servers

- Large server farm appearing as a single virtual server
  - Single front-end for connection routing

- Algorithms
  - Load-based
    - Response times
    - Server # of connections
    - Server CPU utilization
    - Weighted round-robin
  - Layer 3 info (source IP address)
  - Layer 4 info (ports)
  - Layer 5-7 info
    - URLs
    - Cookies
    - SSL session IDs
    - User-Agent, client capabilities
    - Requires connection termination and reverse-proxying
Olympic web server (1996)

Internet

pdx.edu

SYN routing
ACK forwarding

Load info

Token Ring

4 x T3

4

IP=X

3

IP=X

2

IP=X

1

pdx.edu

Internet
Olympic web server (1996)

- Front-end node (reverse NAT)
  - TCP SYN
    - Route to particular server based on policy
    - Store decision (connID, realServer)
  - TCP ACK
    - Forward based on stored decision
  - TCP FIN or a pre-defined timeout
    - Remove entry
- Servers
  - IP address of outgoing interface = IP address of front-end’s incoming interface
  - Treats front-end, token-ring, and cluster as one virtual server
Olympic web server (1996)

- **Advantages**
  - Only ACK traffic is processed (i.e. headers rewritten)
  - More reactive to load than DNS

- **Disadvantages**
  - Potential non-stickiness between requests
    - SSL sessions for a single client
    - Cache performance
Virtual server variations (L2-L4)

- Evolved into hardware switch implementations
  - And now software-based ones (nginx)

- Load balancing algorithms
  - Anything contained within TCP/IP header
    - "5-tuple" <sourceIP, sourcePort, destIP, destPort, protocol>
    - hash(source, dest, protocol)
  - Server characteristics
    - Least number of connections
    - Fastest response time
    - Server idle time
  - Other
    - Weighted round-robin based on server capabilities
    - Random
Virtual servers with L5

- Can also load balance based on content (i.e. URL)
- Requires one to proxy server connection until URL sent, before routing to backend servers
  - Also known as a "reverse-proxy"
- Examples: nginx, Google's front-end (GFE), CloudFlare, many hardware switches
- Switch/proxy
  - Terminates TCP handshake
  - Rewrites sequence numbers going in both directions
L5 switches

- SYN SN=A
- SYN SN=B ACK=A
- ACK=B
- HTTP request
  - Rewrite Y to X
  - C to B
- ACK
  - Rewrite X to Y
  - B to C
- HTTP response
  - HTTP request
  - SYN SN=A
  - SYN SN=C ACK=A
  - ACK=C
  - HTTP request
  - SYN SN=A
- Reverse proxy
  - Route request
  - SYN SN=A
  - SYN SN=C ACK=A
  - ACK=C
  - HTTP request
  - L5 switch VirtualIP=X
  - Real server RealIP=Y
L5 switching

- **Advantages**
  - Increases effective cache/storage sizes
  - Allows for session persistence (SSL, cookies)
  - Support for user-level service differentiation
    - Service levels based on cookies, user profile, User-Agent, URL
    - DDoS prevention based on request/user

- **Disadvantages**
  - Hot-spots
  - Overhead (custom ASICs needed to process at line-speed)
[a-m].root-servers.net

*.yahoo.com is served by
ns1.yahoo.com (204.71.177.33)
n3.europe.yahoo.com (195.67.49.25)
n2.dca.yahoo.com (209.143.200.34)
n5.dcx.yahoo.com (216.32.74.10)

DNS cache
Host: www.yahoo.com
NameServers: yahoo.com

pdx.edu

akamaitech.net

us.yimg.com

ns1.yahoo.com

www.yahoo.com is served by
204.71.200.68
204.71.200.67
204.71.200.75
204.71.202.160
204.71.200.74

yahoo.com
Google Cloud Platform

- Supported via Cloud DNS, Load Balancer
  - Map DNS records to your GCP instances
  - Spread HTTP requests across multiple compute engine instances and regions via a single (anycasted) IP address or a small collection of IP addresses
    - 3 kinds (L4 connection load balancing, L7 content-based load balancing, network latency load balancing)
  - Forward deploy content via compute engine instances in load balancer to leverage edge caches in GCP
DDoS prevention via CDN
DDoS prevention

- Can use CDN for distributed denial-of-service mitigation
  - CDN manages your DNS to point to forward-deployed nodes
  - Performs a reverse proxy operation on nodes as previously
    - Terminates connections and examines request, before forwarding to content nodes
    - Drops sources of unwanted requests
      - Mirai traffic
    - Drop malicious requests after analysis by web-application firewall (WAF)
      - Common XSS payloads, known exploits
- Examples: CloudFlare, Akamai, Google, Microsoft
CloudFlare architecture

Without CloudFlare:
- Visitor
- Crawlers and bots
- Attackers
- Slow pipes
- Your naked website

With CloudFlare:
- Visitor
- Crawlers and bots
- Attackers
- Fast pipes
- CloudFlare's globally distributed network
- CloudFlare protected website
Azure DDoS Protection (4/18/18)

- Reverse-proxy at edge
- "L7" protection
  - WAF (SQLi, XSS filter)
  - Rate-limit per IP addr
  - Protocol attacks (floods)

Microsoft Azure

Azure DDoS Protection for virtual networks generally available

Posted on April 18, 2018

Anupam Vij, Senior Product Manager, Azure Networking

Issue

- Issue: HTTPS proxying
  - To proxy an https connection at edge, CDN must have
    - Certificate of site it's protecting
    - Private key associated with certificate to decrypt key from client
      - e.g. client encrypts random key with public key of site
      - Can only be decrypted by private key
  - But, not all sites want to give up private key to CloudFlare (or other CDNs)
Key server architecture

- Site co-locates key server that implements part of TLS requiring private key

CloudFlare Keyless SSL

Key negotiation

Visitor
- Client random
- Server random
- Public key certificate
- Premaster secret
- Session key

1. Visitor sends hello, client random, and protocols supported
2. Server sends server random and public key certificate
3. Visitor encrypts premaster secret with public key

CloudFlare
- Client random
- Server random
- Premaster secret
- Encrypted premaster secret
- Session key

Origin server
- Cached content
- Uncached content

Key server
- Private key

Both the visitor and CloudFlare derive identical session keys from the client random, server random, and premaster secret. The visitor can request content from CloudFlare, and the request will be encrypted.
Labs
Based on the location of the local name server performing a name resolution, most sites served by a CDN attempt to resolve their names to the IP address of the nearest server that hosts it.

Using `dig`, we will resolve `www.google.com` from different local name servers to see how resolution differs from different locations.

- Lookup geographic locations of the following DNS servers via [https://www.iplocation.net/](https://www.iplocation.net/)
  - 131.252.208.53 (PSU)
  - 198.82.247.66 (Virginia Tech)
- Then, using `dig`, resolve `www.google.com` from each of the DNS servers and record each result.
- Lookup up geographic locations for each that do not give Google's Mountain View headquarters as the result.
  - What is the geographic distance between the DNS server and web server?
- Perform a traceroute to each of the addresses.
  - Do the routes reveal any information on the accuracy of the geographic locations given? Answer might be no.
CDN Lab #2: HTTP load balancing

- Scaling and deploying a load balanced web site
- In under an hour!

via checkboxes!
CDN Lab #2: HTTP load balancing

- Scaling and deploying a load balanced web site
- Skip intro steps if not needed, use exact naming
- Launch Cloud Shell and copy files from lab's bucket on Google Cloud Storage (similar to AWS S3 buckets)

```
mkdir networking101
cd networking101
gsutil cp gs://networking101/* .
```

- Cloud Deployment Manager
  - Allows one to specify and deploy collections of VMs
  - For lab, configuration include in files copied via YAML and Jinja files (more on Jinja later)

```yaml
# networking-lab.yaml
resources:
- name: compute-engine-setup
  type: compute-engine-template.jinja

# compute-engine-template.jinja
- name: w2-vm
  type: vm-template.jinja
  properties:
    machineType: n1-standard-1
    zone: us-west1-b
    network: {{ NETWORK_NAME }}
    subnetnetwork: us-west1-s2
    ip: 10.11.0.100
```
CDN Lab #2: HTTP load balancing

- Launch the configuration to instantiate the network on GCP
  - Will need to enable the Deployment Manager API
  - Grab some coffee (takes a while to instantiate)

```
gcloud deployment-manager deployments create networking101 \
  --config networking-lab.yaml
```
CDN Lab #2: HTTP load balancing

- Show the machines running within Cloud Shell and Compute Engine UI
- Note: If ssh works on Step 3, you may skip Step 4 (read through it anyway just for fun) Launch an ssh session on each VM to perform Step 5 and 6
  - For Step 6, repeat the pair-wise ping table from the various VMs
  - Pinging external hosts step optional
- Step 8: Set up network to allow HTTP traffic to all instances (name the rule http-server)
  - Done either via `gcloud` command-line or via Console UI

```bash
gcloud compute firewall-rules create nw101-allow-http \
  --allow tcp:80 --network networking101 --source-ranges 0.0.0.0/0 \
  --target-tags http-server
```

- Note: "Networking" via the Menu is now labeled "VPC Network"
CDN Lab #2: HTTP load balancing

- **Step 9: Managed Instance Groups**
  - In Compute Engine, "Instance Templates" are recipes for creating identical VMs in a group (Instance Groups)
  - Group is managed by a policy
    - e.g. Maximum 5, Minimum 1, add a node when avg load > 0.8, subtract a node when avg load < 0.8
  - Creation via command line with a startup script taken from a different bucket
    - Example template creation command for machines residing in us-east1 (Sets subnet and region, associate http-server tag to allow HTTP traffic)
    - Run startup script at gs://networking101-lab/startup.sh

```bash
# Create template for machines residing in us-east1
# Set subnet and region, associate with http-server tag to allow HTTP
gcloud compute instance-templates create "us-east1-template" \
  --subnet "us-east1" \
  --metadata "startup-script-url=gs://networking101-lab/startup.sh" \
  --region "us-east1" \
  --tags "http-server"
```
CDN Lab #2: HTTP load balancing

- Startup script installs Apache2 and PHP, then updates index files to list region
  - Recall awk, $(), ``, from previous lab
  - sed: Unix stream editor for filtering and transforming text (via regexp)
    - substitutes the string "region-here" with environment variable

```
#! /bin/bash
apt-get update
apt-get install -y apache2 php
cd /var/www/html
rm index.html -f
rm index.php -f
wget https://storage.googleapis.com/networking101-lab/index.php
META_REGION_STRING=$(curl "http://metadata.google.internal/computeMetadata/v1/instance/zone" -H "Metadata-Flavor: Google")
REGION=`echo "$META_REGION_STRING" | awk -F/ '{print $4}'`
sed -i "s|region-here|$REGION|" index.php
```
CDN Lab #2: HTTP load balancing

- In Console UI, instantiate managed instance group from templates for both regions (note the configurations are different)
  - us-east-1 dynamic from 1 to 5 nodes
  - europe-west1 static 3 nodes
Create a new instance group

Use an instance group when configuring a load-balancing backend service or to group VM instances. Learn more

Name
us-east1-mig

Description (Optional)

Location
Multi-zone groups span multiple zones which assures higher availability. Learn more
- Single-zone
- Multi-zone

Region
us-east1

Specify port name mapping (Optional)

Instance definition
us-east1-template

Autoscaling
On

Autoscale based on
For best results read Configuring autoscaling instance groups
- HTTP load balancing usage

Target load balancing usage
Scaling dynamically creates or deletes VMs to meet the group target. Learn more
- 80%

Minimum number of instances
1

Maximum number of instances
5

Cool-down period
45 seconds

Autohealing
VMs in the group are recreated as needed. You can use a health check to recreate a VM if the health check finds the VM unresponsive. If you do not select a health check, VMs are recreated only when stopped. Learn more

Health check
- No health check

Initial delay
300 seconds

Create Cancel
CDN Lab #2: HTTP load balancing

- Hit the web page of one of the instance
  http://<Instance_IP>
  - If request fails, you may need to check your HTTP firewall rule and restart instances with the traffic enabled
- Step 10: Now at Network Services => Load balancing
  - Start configuration and specify backend configuration (e.g. a backend service implemented via the VMs in the instance groups)
• Go through *entire* configuration and then Create
Result
CDN Lab #2: HTTP load balancing

- Step 11
  - Show site via IP address assigned to load balancer (the Anycast IP address)
    - Which instance group did the request go to?

![Google Cloud Platform UI showing load balancer details and networking lab results](image-url)
CDN Lab #2: HTTP load balancing

- Place the site under siege, from w1-vm
  
  `singe -c 250 http://<http-loadbalancer-ip>`

- Show the autoscaling turning on for us-east1 instance group
CDN Lab #2: HTTP load balancing

- Perform optional steps 12-13, skip 14
  - Perform traceroutes from VMs to pdx.edu
  - Skip MTR section
- Make sure to cleanup!
  - Use tabs in Cloud Shell to delete in parallel
  - Note on firewall rule, it's named networking101-allow-http

- [https://codelabs.developers.google.com/codelabs/cloud-networking-101](https://codelabs.developers.google.com/codelabs/cloud-networking-101)
Extra
Proxinet Example

- CPU utilization and cache hit rates are biggest concerns
- Ignore network efficiency and other resources
- Support user and URL differentiation
- Straight up L5
  - killed on hot-spot URLs
- Straight up load-based algorithm
  - killed on low cache hit rates unless global cache used
Proxinet Example

- Solution: Use a hybrid like LARD
  - load balance with URL to a certain limit
  - load balance with least connections when load imbalanced
  - provision by URL based on Benjamins

http://www.cs.princeton.edu/~vivek/ASPLOS-98/
Transparent Web Caching

- Redirect web requests to cache transparently
- Eliminates client management costs
- How?
  - Put web cache/redirector in routing path
  - Redirector
    - Pick off cacheable web requests
    - rewrite destination address and forward to cache
    - rewrite source address and return to client

Transparent Web Caching

Pick off web requests
rewrite addresses
route to caches
Scalable Secure Servers

- SSL handshake
- Server intensive processing
  - 200 MHz PowerPC
  - Client ~12ms processing
  - Server ~50ms processing
- Session reuse avoids overhead of handshake
- Subsequent requests must return to initial server
Scalable Secure Servers

Client

1. Verify cert, extract server public key, encrypt master secret w/ key

2. Decrypt master secret with private key, generate keys and randoms

3. Generate keys from master secret + randoms

Server

Client Hello
Client random + GMT + sessionID=0
cipher suites + compression methods

Server Hello
Server random + certificate + sessionID
cipher suites + compression methods

Client Key Exchange
Master secret encrypted w/ server public key

Finished

Application data

Initial SSL Handshake
Scalable Secure Servers

1. Generate keys from cached master secret and current randoms

2. Generate keys from cached master secret and current randoms

Client

Client Hello
Client random + GMT + sessionID
cipher suites + compression methods

Server Hello + Finished
Server random + certificate + sessionID
cipher suites + compression methods

Finished

Application data

Server

SSL session reuse
Scalable Secure Servers

- Source IP switching solution
  - solves affinity problem
  - load balancing poor
- SSL session ID switching
  - solves affinity problem
  - load balancing on initial handshake