Review by example:
Building scalable web services
Building scalable web services

• A relatively easy problem…
  – Why?
    • HTTP: stateless, request-response protocol
    • decoupled, independent requests
  – How?
    • divide and conquer
    • replicate, partition, distribute, load balance
Outline

• Application layer tricks
  – explicit server partitioning
  – dynamic name resolution

• Transparent networking tricks
  – virtual servers

• Case studies
  – scalable content delivery (Yahoo!)
  – content transformation engines
  – transparent web caches
  – scalable secure servers
Explicit server partitioning (static)

- Run a new server per resource/service
- Example
  - www.blah.com
  - mail.blah.com
  - images.blah.com
  - shopping.blah.com
  - my.blah.com
  - etc. etc.
Explicit server partitioning (static)

• Advantages
  – better disk utilization
  – better cache performance

• Disadvantages
  – lower peak capacity
  – coarse load balancing across servers/services
  – management costs
Explicit server partitioning (dynamic)

• Basis for CDNs (Content Distribution Networks)
• Active “forward deployment” of content to explicitly named servers near client
• Redirect requests from origin servers by
  – HTTP redirects
  – dynamic URL rewriting of embedded content
• Application-level multicast based on geographic information
  – Akamai
Explicit server partitioning (dynamic)

Requested page with links to embedded content rewritten OR an HTTP redirect
Explicit server partitioning (dynamic)

• **Advantages**
  – better network utilization
  – better load distribution

• **Disadvantages**
  – distributed management costs
  – storage costs
    • currently OK as ($ network bw >> $ storage)
Outline

• Application layer tricks
  – explicit server partitioning
  – dynamic name resolution

• Transparent networking tricks
  – virtual servers

• Case studies
  – scalable content delivery (Yahoo!)
  – content transformation engines
  – transparent web caches
  – scalable secure servers
DNS load balancing

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

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

[pdx.edu]

[a-m].root-servers.net
*.ncsa.uiuc.edu is served by
ns0.ncsa.uiuc.edu (141.142.2.2)
nsl.ncsa.uiuc.edu (141.142.230.144)
dns1.cso.uiuc.edu (128.174.5.103)
ncs.indiana.edu (129.79.1.1)

[ncsa.uiuc.edu]
DNS load balancing

• Advantages
  – simple, easy to implement
  – uses existing infrastructure

• Disadvantages
  – coarse load balancing
  – local DNS caching affects performance
  – full server replication
DNS RFCs

• RFC 1794
  – “DNS Support for Load Balancing”

• RFCs 1034 and 1035 (1987)
  – Replace older DNS RFCs 882 and 883 (1983)
Outline

• Application layer tricks
  – explicit server partitioning
  – dynamic name resolution

• Transparent networking tricks
  – virtual servers

• Case studies
  – scalable content delivery (Yahoo!)
  – content transformation engines
  – transparent web caches
  – scalable secure servers
Virtual servers

• Large server farm => single virtual server
• Single front-end for connection routing
• Routing algorithms
  – by load (response times, least connections, server load, weighted round-robin)
  – by layer 3 info (IP addresses)
  – by layer 4 info (ports)
  – by layer 5-7 info (URLs, Cookies, SSL session IDs, User-Agent, client capabilities, etc. etc.)
Olympic web server (1996)

SYN routing
ACK forwarding

Pdx.edu

Internet

Token Ring

Load info

4 x T3

1

2

3

4

IP=X

IP=X

IP=X

IP=X

IP=X
Olympic web server (1996)

• Front-end node
  – 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
Olympic web server (1996)

- **Advantages**
  - only ACK traffic is processed
  - more reactive to load than DNS

- **Disadvantages**
  - non-stickiness between requests
    - SSL
    - cache performance
  - can’t support L5 switching to support stickiness
    - must proxy both ways of connection
    - need to rewrite ACKs going both ways
  - software solution (prone to DOS)
L2-L4 virtual servers

- Hardware switches: reverse NAT

![Diagram showing Internet, ISPs, Hybrid cloud, and Hosting provider with IP=X and Private IP addresses connected to virtual servers.]
L2-L4 virtual servers

- **Load balancing algorithms**
  - anything contained within TCP SYN packet
    - 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, random
L5 virtual servers

• Server-side NAT device as before
• Spoof server connection until URL sent
• Switch based on content in request
• Connections proxied through switch switch terminates TCP handshake
  – switch rewrites sequence numbers going in both directions
L5 switches

Client

SYN SN=A

SYN SN=B ACK=A

ACK=B

HTTP request

L5 switch
VirtualIP=X

Rewrite Y to X
C to B

Rewrite X to Y
B to C

ACK

Real server
RealIP=Y

HTTP request

SYN SN=A

SYN SN=C ACK=A

ACK=C

HTTP response
L5 switching

• Advantages
  – increases effective cache/storage sizes in a similar manner as explicit server partitioning
  – allows for session persistence (SSL, cookies)
  – support for user-level service differentiation
    • service levels based on cookies, user profile, User-Agent, URL

• Disadvantages
  – content hot-spots
  – overhead
Load balancing switches

- Cisco Local Director/Arrowpoint
- F5 BIG/ip
- Foundry ServerIron
- Nortel/Alteon ACEDirector
- Linux virtual server
Integrated DNS/virtual server approaches

- LB switches coordinate and respond to DNS requests
  - based on load
  - based on geographic location
- Vendors
  - Cisco Distributed Director
  - F5 Labs BIG/ip with 3DNS
  - Nortel/Alteon ACEDirector3
Integrated example

1. Request for www.blah.com
2. www.blah.com?
3. www.blah.com is C
4. Request to www.blah.com

Load C < Load B < Load A or proximity C > proximity B > proximity A

Internet

[a-m].root-servers.net
www.blah.com served by A, B, and C
Issues with load-balancing virtual servers

- Hot-spot URLs
  - L5, URL switching bad

- Proxied sources
  - (i.e. HTTP proxies (AOL), SOCKS, NAT devices etc.)
  - L3, source IP switching bad

- Stateful requests (SSL)
  - Load-based/RR bad

- Optimizing cache/disk
  - non-L5 solutions bad

- Optimizing network bandwidth
  - non-Akamai-like solutions bad

- IP fragmentation
  - Breaks all algorithms unless switch defrags
Outline

• Application layer tricks
  – explicit server partitioning
  – dynamic name resolution

• Transparent networking tricks
  – virtual servers

• Case studies
  – scalable content delivery (Yahoo!)
  – content transformation engines
  – transparent web caches
  – scalable secure servers
Designing a solution

• Examine primary design goals
  – load balancing performance
  – cache hit rates
  – CPU utilization
  – network resources

• Apply solutions which fit problem
Yahoo!

[a-m].root-servers.net
*.yahoo.com is served by
ns1.yahoo.com (204.71.177.33)
ns3.europe.yahoo.com (195.67.49.25)
ns2.dca.yahoo.com (209.143.200.34)
ns5.dcx.yahoo.com (216.32.74.10)
Proxinet Example

• Application
  – “Browser in the sky”
    • Download and rendering done on a server
    • Server does
      – HTTP protocol functions
      – HTML parsing, rendering, and layout
      – Caching
      – Transcoding of images
      – Packaging and compression
    • Client (Palm/PocketPC) does
      – Basically nothing

• Server architecture
  – CPU utilization and cache hit rates are biggest concerns
  – Network efficiency and other resources are non-issues
  – Want to support user and URL differentiation
    • L5 killed on hot-spot URLs
    • Load based algorithms killed on low cache hit rates (unless global cache is used)
Proxinet Example

• Solution: Use a hybrid like LARD
  http://www.cs.princeton.edu/~vivek/ASPLOS-98/
  – load balance with URL to a certain limit
  – load balance with least connections when load imbalanced
  – provision by URL based on $$ from web site?
Transparent Web Caching

- Redirect web requests to cache transparently
- Eliminates client management costs over explicit HTTP proxies
- 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

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

Client

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

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

Very slow

3. Generate keys from master secret + randoms
Scalable Secure Servers

Client

2. Generate keys from cached master secret and current randoms

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

1. Generate keys from cached master secret and current randoms

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