Outline

- Admin and recap
- Case studies: Content Distribution
  - Forward proxy (web cache)
  - Akamai
  - YouTube
- P2P networks
  - Overview
  - The scalability problem
Assignment three questions
Recap: Load Direction

server state

net path property between servers/clients

specific request of a client

server selection algorithm

server routing

notify client about selection (direction mech.)
Recap: Direction Mechanisms

DNS name1

IP1
Cluster1 in US East
Load balancer

IP2
Cluster2 in US West
Load balancer

Cluster2 in Europe
Load balancer

- Rewrite
- Direct reply
- Fault tolerance

DNS name2

IPn

servers

Load balancer
proxy

Load balancer
Scalability of Traditional C-S Web Servers

DNS

app. server

C₀

client 1

client 2

client 3

client n
Outline

- Admin and recap
- Case studies: Content Distribution
Content Distribution History...

“With 25 years of Internet experience, we’ve learned exactly one way to deal with the exponential growth: Caching”.

(1997, Van Jacobson)
Initial Approach: Forward Cache/Proxy

- Web caches/proxy placed at entrance of an ISP

- Client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client
Benefits of Forward Web Caching

Assume: cache is “close” to client (e.g., in same network)

- smaller response time: cache “closer” to client
- decrease traffic from distant servers
  - link at institutional/local ISP network often bottleneck
  - Cache Hit ratio increases logarithmically with number of users
- Web protocols evolved extensively to accommodate caching, e.g. HTTP 1.1
What Went Wrong with Forward Web Caches?

- However, client site (forward) Web caching was developed with a strong ISP perspective, leaving content providers out of the picture
  - It is the ISP who places a cache and controls it
  - ISP’s main interest to use Web caches is to reduce bandwidth

- In the USA: Bandwidth relative cheap

- In Europe, there were many more Web caches
  - However, ISPs can arbitrarily tune Web caches to deliver stale content
Content Provider Perspective

- Content providers care about
  - User experience latency
  - Content freshness
  - Accurate access statistics
  - Avoid flash crowds
  - Minimize bandwidth usage in their access link
Content Distribution Networks

- CDN design perspective: service to content publishers
  - performance scalability (high throughput, going beyond single server throughput)
  - geographic scalability (low propagation latency, going to close-by servers)
  - content publisher control/access
Akamai

- Akamai – original and largest commercial CDN operates around 137,000 servers in over 1,150 networks in 87 countries
- Akamai (AH kuh my) is Hawaiian for intelligent, clever and informally “cool”. Founded Apr 99, Boston MA by MIT students
- Akamai evolution:
  - Files/streaming (our focus at this moment)
  - Secure pages and whole pages
  - Dynamic page assembly at the edge (ESI)
  - Distributed applications
Akamai Scalability Bottleneck

See Akamai 2009 investor analysts meeting
Basic of Akamai Architecture

- **Content publisher** (e.g., CNN, NYTimes)
  - provides base HTML documents
  - runs *origin* server(s)

- **Akamai runs**
  - *edge* servers for hosting content
    - Deep deployment into 1150 networks
  - customized **DNS redirection servers** to select edge servers based on
    - closeness to client browser
    - server load
Linking to Akamai

- Originally, URL Akamaization of embedded content: e.g.,
  
  `<IMG SRC= http://www.provider.com/image.gif >`
  
  changed to
  
  `<IMG SRC = http://a661.g.akamai.net/hash/image.gif>`

  Note that this DNS redirection unit is per customer, not individual files.

- URL Akamaization is becoming obsolete and supported mostly for legacy reasons
  
  - Currently most content publishers prefer to use DNS CNAME to link to Akamai servers
Akamai Load Direction Flow

Client requests site

Client gets CNAME with domain name in Akamai

Multiple redirections to find nearby edge servers

Client is given 2 nearby web replica servers (fault tolerance)

Exercise: Zoo machine

- Check any web page of New York Times and find a page with an image
- Find the URL
- Use
  \%dig +trace +recurse
  to see Akamai load direction
If the directed edge server does not have requested content, it goes back to the original server (source).
Two-Level Direction

proximity: high-level DNS determines client location; directs to low-level DNS, who manages a close-by cluster
Local DNS Alg: Potential Input

- $p(m, e)$: path properties (from a client site $m$ to an edge sever $e$)
  - Akamai might use a one-hop detour routing (see akamai-detour.pdf)
- $a_k^m$: request load from client site $m$ to publisher $k$
- $x_e$: load on edge server $e$
- caching state of a server $e$
Local DNS Alg

- Details of Akamai algorithms are proprietary
- A Bin-Packing algorithm (column 12 of Akamai Patent) every $T$ second
  - Compute the load to each publisher $k$ (called serial number)
  - Sort the publishers from increasing load
  - For each publisher, associate a list of random servers generated by a hash function
  - Assign the publisher to the first server that does not overload
Hash Bin-Packing

LB: maps request to individual machines inside cluster
Experimental Study of Akamai Load Balancing

- **Methodology**
  - 2-months long measurement
  - 140 PlanetLab nodes (clients)
    - 50 US and Canada, 35 Europe, 18 Asia, 8 South America, the rest randomly scattered
  - Every 20 sec, each client queries an appropriate CNAME for Yahoo, CNN, Fox News, NY Times, etc.

See [http://www.aqualab.cs.northwestern.edu/publications/Ajsu06DBA.pdf](http://www.aqualab.cs.northwestern.edu/publications/Ajsu06DBA.pdf)
Server Pool: to Yahoo

Target: a943.x.a.yimg.com (Yahoo)

Client 1: Berkeley

Client 2: Purdue

day

night
Server Diversity for Yahoo

Majority of PL nodes see between 10 and 50 Akamai edge-servers.

Nodes far away from Akamai hot-spots.
Server Pool: Multiple Akamai Hosted Sites

The diagram shows the number of Akamai Web Replicas for various clients. The X-axis represents the clients, and the Y-axis represents the number of Akamai Web Replicas. The clients include:
- cs.purdue.edu
- comet.columbia.edu
- cs.uoregon.edu
- cs.umass.edu
- lbnl.nodes.planet-lab.org

The clients listed are associated with Yahoo!, Amazon, AOL, NYTimes, Apple, Monster, FOX News, MSN, PCWorld, and FEMA.
Load Balancing Dynamics

![Graph showing CDF (prob < X) vs. Inter-redirection time (s) for Berkeley to Yahoo, Korea to Yahoo, and Brazil to Yahoo. The graph includes points labeled Berkeley, Brazil, and Korea.](image_url)
Redirection Effectiveness: Measurement Methodology

Akamai Low-Level DNS Server

Planet Lab Node

9 Best Akamai Replica Servers
Do redirections reveal network conditions?

- **Rank** = \( r_1 + r_2 - 1 \)
  - 16 means perfect correlation

<table>
<thead>
<tr>
<th>Rank</th>
<th>csail.mit.edu</th>
<th>cs.vu.nl</th>
<th>pop-ce.rnp.br</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>95</td>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>85</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>7</td>
<td>65</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>9</td>
<td>55</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>11</td>
<td>45</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>12</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>13</td>
<td>35</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>14</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>15</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>18</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Brazil is poor
- MIT and Amsterdam are excellent

Percentage of time Akamai's selection is better or equal to rank
A content publisher (e.g., a radio or a TV station) encodes streams and transfer them to entry points.

Group a set of streams (e.g., some popular some not) into a bucket called a portset. A set of reflectors will distribute a given portset.

When a user watches a stream from an edge server, the server subscribes to a reflector.

Compare with Web architecture.
Testing Akamai Streaming Load Balancing

(a) Add 7 probing machines to the same edge server
(b) Observe slow down
(c) Notice that Akamai removed the edge server from DNS; probing machines stop

Figure 7: Slow load balancing experiment
You Tube

- 02/2005: Founded by Chad Hurley, Steve Chen and Jawed Karim, who were all early employees of PayPal.
- 10/2005: First round of funding ($11.5 M)
- 03/2006: 30 M video views/day
- 07/2006: 100 M video views/day
- 11/2006: acquired by Google
- 10/2009: Chad Hurley announced in a blog that YouTube serving well over 1 B video views/day (avg = 11,574 video views/sec)
Pre-Google Team Size

- 2 Sysadmins
- 2 Scalability software architects
- 2 feature developers
- 2 network engineers
- 1 DBA
- 0 chefs
while (true)
{
    identify_and_fix_bottlenecks();
    drink();
    sleep();
    notice_new_bottleneck();
}
YouTube Major Components

- Web servers
- Video servers
- Thumbnail servers
- Database servers
### YouTube: Web Servers

**Components**
- Netscaler load balancer; Apache; Python App Servers; Databases

**Python**
- Web code (CPU) is not bottleneck
  - JIT to C to speedup
  - C extensions
  - Pre-generate HTML responses
- Development speed more important
YouTube: Video Popularity

YouTube: Video Popularity

How to design a system to handle highly skewed distribution?

Fig. 8. Recently added YouTube videos rank by popularity

YouTube: Video Server Architecture

- Tiered architecture
  - CDN servers (for popular videos)
    - Low delay; mostly in-memory operation
  - YouTube servers (not popular 1-20 per day)
YouTube Redirection Architecture

YouTube servers

CDN server located in YouTube or Limelight network

(3) HTTP Get MSG

(4) Flash video stream

(1) HTTP Get MSG

(2) HTTP Redirect MSG

Client

[Example of (1)]
Get /get_video?video_id=G_Y3y8escmA
HTTP/1.1

[Example of (2)]
HTTP/1.1 303 See other
Location: http://sjc-v110.sjc.youtube.com
/get_video?video_id=G_Y3y8escmA
YouTube Video Servers

- Each video hosted by a mini-cluster consisting of multiple machines
- Video servers use the lighttpd web server for video transmission:
  - Apache had too much overhead (used in the first few months and then dropped)
  - Async io: uses epoll to wait on multiple fds
  - Switched from single process to multiple process configuration to handle more connections
Thumbnail Servers

- Thumbnails are served by a few machines
- Problems running thumbnail servers
  - A high number of requests/sec as web pages can display 60 thumbnails on page
  - Serving a lot of small objects implies
    - lots of disk seeks and problems with file systems, inode and page caches
    - may ran into per directory file limit
  - Solution: storage switched to Google BigTable
Design 1: Squid in front of Apache

- Problems
  - Squid worked for a while, but as load increased performance eventually decreased: Went from 300 requests/second to 20
  - under high loads Apache performed badly, changed to lighttpd

Design 2: lighttpd default: By default lighttpd uses a single thread

- Problem: often stalled due to I/O

Design 3: switched to multiple processes contending on shared accept

- Problems: high contention overhead/individual caches
Thumbnails Server: lighttpd/aio
Scalability of Content Distribution

DNS

origin

depth servers

client 1

client 2

client 3

client n
Objectives of P2P

- **The scalability problem**
  - Share the resources (storage and bandwidth) of individual clients to improve scalability/robustness

- **The Lookup problem**
  - More generally, moving from a host-centric Internet to a “data-centric” Internet supporting data persistency, availability, and authenticity
But P2P is not new

Original Internet was a p2p system:
- The original ARPANET connected UCLA, Stanford Research Institute, UCSB, and Univ. of Utah
- No DNS or routing infrastructure, just connected by phone lines
- Computers also served as routers

P2P is simply an iteration of scalable distributed systems
P2P Systems

- File Sharing: BitTorrent
- Streaming: Octoshape, Adobe 10.1 later
- PPLive...
- Games: Xbox ...
Outline

- Admin and recap
  - Case studies: Content Distribution
    - Forward proxy (web cache)
    - Akamai
    - YouTube
  - P2P networks
    - Overview
    - The scalability problem
An Upper Bound on Scalability

- **Assume**
  - need to achieve same rate to all clients
  - only uplinks can be bottlenecks
- What is an upper bound on scalability?
The Scalability Problem

- Maximum throughput
  
  \[ R = \min\{C_0, \frac{(C_0 + \Sigma C_i)}{n}\} \]

- The bound is theoretically approachable
Theoretical Capacity: upload is bottleneck

- Assume $c_0 > (C_0 + \sum C_i)/n$
- Tree i:
  - server $\rightarrow$ client i: $c_i/(n-1)$
  - client i $\rightarrow$ other n-1 clients
- Tree 0:
  - server has remaining
  - $c_m = c_0 - (c_1 + c_2 + ... c_n)/(n-1)$
  - send to client i: $c_m/n$

\[ R = \min\{C_0, (C_0 + \sum C_i)/n\} \]
Why not Building the Trees?

- Clients come and go (churns): maintaining the trees is too expensive
- Each client needs $N$ connections
Key Design Issues

- Robustness
  - Resistant to churns and failures

- Efficiency
  - A client has content that others need; otherwise, its upload capacity may not be utilized

- Incentive: clients are willing to upload
  - 70% of Gnutella users share no files
  - nearly 50% of all responses are returned by the top 1% of sharing hosts

- Lookup problem
Discussion: How to handle the issues?

- Robustness
- Efficiency
- Incentive
Outline

- Admin and recap
  - Case studies: Content Distribution
    - Forward proxy (web cache)
    - Akamai
    - YouTube
  - P2P networks
    - Overview
    - The scalability problem
      - BitTorrent
BitTorrent

- A P2P file sharing protocol
- Created by Bram Cohen in 2004
  - Spec at bep_0003: http://www.bittorrent.org/beps/bep_0003.html
Outline

- Admin and recap
  - Case studies: Content Distribution
  - P2P networks
    - Overview
    - The scalability problem
      - BitTorrent
        - Lookup
BitTorrent

- Mostly tracker based
- Tracker-less mode; based on the Kademlia DHT
BitTorrent: Lookup

HTTP GET MYFILE.torrent

MYFILE.torrent

http://mytracker.com:6969/
S3F5YHG6FEB
FG5467HGF367
F456JI9N5FF4E
...

user
Metadata (.torrent) File Structure

- Meta info contains information necessary to contact the tracker and describes the files in the torrent
  - URL of tracker
  - file name
  - file length
  - piece length (typically 256KB)
  - SHA-1 hashes of pieces for verification
  - also creation date, comment, creator, ...
Tracker Protocol

- Communicates with clients via HTTP/HTTPS

- **Client GET request**
  - `info_hash`: uniquely identifies the file
  - `peer_id`: chosen by and uniquely identifies the client
  - `client IP and port`
  - `numwant`: how many peers to return (defaults to 50)
  - `stats`: e.g., bytes uploaded, downloaded

- **Tracker GET response**
  - `interval`: how often to contact the tracker
  - `list of peers, containing peer id, IP and port`
  - `stats`
Tracker Protocol

User sends a "register" to tracker

Tracker responds with a list of peers:

<table>
<thead>
<tr>
<th>ID</th>
<th>Peer Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID1</td>
<td>169.237.234.1:6881</td>
</tr>
<tr>
<td>ID2</td>
<td>190.50.34.6:5692</td>
</tr>
<tr>
<td>ID3</td>
<td>34.275.23.143:4545</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>ID50</td>
<td>23:168</td>
</tr>
</tbody>
</table>
Outline

- Admin and recap
  - Case studies: Content Distribution
  - P2P networks
    - Overview
    - The scalability problem
      - BitTorrent
        - Lookup
          - Robustness and efficiency
Piece-based Swarming

- Divide a large file into small blocks and request block-size content from different peers

  Block: unit of download

- If do not finish downloading a block from one peer within timeout (say due to churns), switch to requesting the block from another peer
Detail: Peer Protocol

(Over TCP)

Peers exchange bitmap representing content availability
- `bitfield` msg during initial connection
- `have` msg to notify updates to bitmap
- to reduce bitmap size, aggregate multiple blocks as a piece
Peer Request

- If peer A has a piece that peer B needs, peer B sends interested to A

- unchoke: indicate that A allows B to request

- request: B requests a specific block from A

- piece: specific data

http://www.bittorrent.org/beps/bep_0003.html
Key Design Points

- request:
  - which data blocks to request?

- unchoke:
  - which peers to serve?
Request: Block Availability

- Request (local) **rarest first**
  - achieves the fastest replication of rare pieces
  - obtain something of value
Block Availability: Revisions

- When downloading starts (first 4 pieces): choose at random and request them from the peers
  - get pieces as quickly as possible
  - obtain something to offer to others

- Endgame mode
  - defense against the “last-block problem”: cannot finish because missing a few last pieces
  - send requests for missing pieces to all peers in our peer list
  - send cancel messages upon receipt of a piece
BitTorrent: Unchoke

- Periodically (typically every 10 seconds) calculate data-receiving rates from all peers

- Upload to (unchoke) the fastest
  - constant number (4) of unchoking slots
  - partition upload bw equally among unchoked

commonly referred to as “tit-for-tat” strategy
Optimistic Unchoking

- Periodically select a peer at random and upload to it
  - typically every 3 unchoking rounds (30 seconds)

- Multi-purpose mechanism
  - allow bootstrapping of new clients
  - continuously look for the fastest peers
    (exploitation vs exploration)