CS433/533
Computer Networks
Lecture 13
CDN & P2P for Scalability
10/10/2012

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

Admin
- Assignment three questions

Recap: Load Direction

Recap: Direction Mechanisms

Scalability of Traditional C-S Web Servers
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
  - ISPs 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

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

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

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

Akamai Load Direction

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 server \( e \))
  - Akamai might use a one-hop detour routing (see akamai-detour.pdf)
- \( a^e_m \): 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 U.S and Canada, 35 Europe, 38 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.

Akamai Low-Level DNS Server

See http://www.aqualab.cs.northwestern.edu/publications/Ajsu06DBA.pdf

Server Pool: to Yahoo

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

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

Number of Akamai Web Replicas

Load Balancing Dynamics

$CDF (p折磨 < X)$

Inter-redirection time (s)

Redirection Effectiveness: Measurement Methodology

9 Best Akamai Replica Servers
Do redirections reveal network conditions?

- Rank = r1 + r2 - 1
  - 16 means perfect correlation
  - 1 means perfect anti-correlation

Brazil is poor

- CSAIL: MIT are excellent
- csail.mit.edu
- csail.rice.edu
- pop-ce.mp.com

Percentage of time Akamai’s selection is better or equal to rank

Akamai Streaming Architecture

- A content publisher (e.g., a radio or a TV station) encodes streams and transfers 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.

Testing Akamai Streaming Load Balancing

- Add 7 probing machines to the same edge server
- Observe slowdown
- Notice that Akamai removed the edge server from DNS; probing machines stop

YouTube

- 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)

YouTube Design Alg.

```java
while (true) {
    identify_and_fix_bottlenecks();
    drink();
    sleep();
    notice_new_bottleneck();
}
```

Pre-Google Team Size

- 2 Sysadmins
- 2 Scalability software architects
- 2 feature developers
- 2 network engineers
- 1 DBA
- 0 chefs

http://video.google.com/videoplay?docid=-6304664351441328559#
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


How to design a system to handle highly skewed distribution?

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 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 run into per directory file limit
  - Solution: storage switched to Google BigTable

Thumbnail Server Software Architecture

- 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

Scalability of Content Distribution

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
P2P

- 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

The Scalability Problem

- Maximum throughput
  \[ R = \min\{c_0, (c_0 + \Sigma c_i)/n\} \]
  - The bound is theoretically approachable

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?

Theoretical Capacity: upload is bottleneck

- Assume \( c_0 > (c_0 + \Sigma c_i)/n \)
- Tree \( i \):
  - server \( \rightarrow \) client \( i \) : \( c_i/(n-1) \)
  - client \( i \) \( \rightarrow \) other \( n-1 \) clients
- Tree \( 0 \):
  - server has remaining \( c_m = c_0 - (c_1 + c_2 + \ldots c_n)/(n-1) \)
  - send to client \( i \) : \( c_m/n \)
Why not Building the Trees?

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

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

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 90% of all responses are returned by the top 1% of sharing hosts.
- Lookup problem.
### BitTorrent

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

### 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

### 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
  - Block: 16KB

- If do not finish downloading a block from one peer within timeout (say due to churn), 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

---

**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)