CSE 123b
Communications Software

Spring 2002

Lecture 13: Content Distribution Networks
(plus some other applications)

Stefan Savage

Some slides courtesy Srini Seshan

Today's class

- Quick examples of other application protocols
  - Mail, telnet, NFS
- Content Distribution Networks (CDN)

Quick descriptions of some other sample applications

- Sending E-mail
  - SMTP
- Remote terminal
  - Telnet, SSH
- Distributed File Systems
  - NFS

Simple Message Transfer Protocol (SMTP)

Like HTTP: TCP connection (port 25), ASCII string commands
Sample session:

HELO cs.ucsd.edu
Hello cs.ucsd.edu [132.239.4.64]
MAIL FROM: savage@cs.ucsd.edu
250 OK
RCPT TO: joe@cs.berkeley.edu
250 OK
DATA
354 Startup mail input; end with <CRLF>.<CRLF>
Hi Joe… how're you doing?
<CRLF><CRLF>
250 OK
QUIT
221 Closing connection

Telnet

- TCP-based protocol (port 23)
  - Telnet client and telnet server
- First negotiate capabilities (e.g. terminal size, speed, line and a time vs character at a time, etc.)
- Then simply send keystrokes from client to server and send data strings from server to client
  - Characters transmitted as 7 bits (8^{th} bit 0)
  - In-band signalling
    - » Byte 0xff means "interpret as command"
    - » What if you need to send the symbol 0xff? Send it twice.

Network File System (NFS)

- UDP-based protocol
- Remote Procedure Call (RPC) design
  - READ, WRITE, LOOKUP, REMOVE, RENAME, MKDIR, etc…
  - Header describes method and data types, followed by data
  - All requests fit in a single UDP datagram (up to 8k in v2, 64k in V3); fragmentation
  - Errors in data stream?
  - Security?

Content Distribution Networks

- Goal: Improve performance/scalability for downloading content (i.e. Web pages)
- Approach: Replicate content (particularly Web content) on many servers

- Challenges
  - How to replicate content
  - Where to replicate content
  - How to find replicated content
  - How to choose among known replicas
  - How to direct clients towards replica
    - » DNS, HTTP 304 response, anycast, etc.
- Akamai

How to replicate content

- Push model
  - Proactively copy content to specific replicas
  - How to choose these?

- Pull model
  - Reactively replicate content to nodes that request it
  - Content is replicated to places where it is popular

Server Selection

- How do direct clients to a particular server?
  - As part of routing → anycast, cluster load balancing
  - As part of application → HTTP redirect
  - As part of naming → DNS

- **Which** server?
  - Lowest load → to balance load on servers
  - Best performance → to improve client performance
    - » Based on Geography? RTT? Throughput? Load?
  - Any alive node → to provide fault tolerance

Routing Based

- Anycast
  - Give service a single IP address
  - Each node implementing service advertises route to address
  - Packets get routed routed from client to “closest” service node
    - » *Closest* is defined by routing metrics
    - » May not mirror performance/application needs

  - This is done today (sometimes by accident)

Routing Based

- Cluster load balancing
  - Router in front of cluster of nodes directs packets to server
  - Must be done on connection by connection basis – why?
    - » Forces router to keep per connection state
  - How to choose server
    - » Easiest to decide based on arrival of first packet in exchange
    - » Primarily based on local load
    - » Can be based on later packets (e.g. HTTP Get request) but makes system more complex

Application Based

- HTTP support simple way to indicate that Web page has moved
- Server receivess GET request from client
  - Decides which server is best suited for particular client and object
  - Returns HTTP redirect to that server
- Can make informed application specific decision
- May introduce additional overhead → multiple connection setup, name lookups, etc.

Naming Based

- Client does name lookup for service
- Name server chooses appropriate server address
- What information can it base decision on?
  - Server load/location → must be collected
  - Source address in DNS request
  - Round-robin
    - Randomly choose replica
    - Avoid hot-spots
  - [Semi-]static metrics
    - Geography
    - Route metrics

Naming Based

- Predicted application performance
  - How to predict?
  - Only have limited info at name resolution
- Multiple techniques
  - Static metrics to get coarse grain answer
  - Current performance among smaller group
- How does this affect caching?
  - Typically want low TTL to adapt to load changes
  - What do the first and subsequent lookups do?

How Akamai Works

- Content is prepared by rewriting URLs for replicated content
  - <img src="http://cnn.com/af/x.gif"> replaced with <img src="http://a73.g.akamaitech.net/7/23/cnn.com/af/x.gif">
- Clients fetch html document from server
  - E.g. fetch index.html from cnn.com
- Client is forced to resolve aXYZ.g.akamaitech.net hostname for replicated content

How Akamai Works

- gTLD/root server gives NS record for akamai.net
- Akamai.net name server returns NS record for g.akamaitech.net
  - Name server chosen to be in region of client's name server (based on IP address of request)
  - TTL is large
- G.akamaitech.net name server choses a content server in region and returns it to client
  - Uses aXYZ name & hash function over request to pick
  - TTL is small

Akamai Content Servers

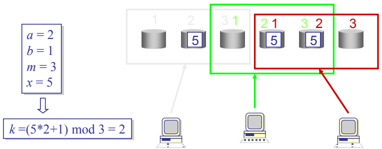
- Are really caches
  - Modified name contains file name
  - If content server doesn't have that object then it is requested from primary server and cached
- Tricky issue is selecting **which** local content server to use for a particular request
  - Want to spread load evenly
  - But want minimal impact if server is added or removed

Naïve approach: Content hashing

- Basic idea: hash pages according to their associated keys
- Straightforward solution
  - Assume m caches (servers), $1, 2, \ldots, m$
  - Store page with key x on cache $(ax + b) \bmod m$
- Advantages:
  - Load balancing: each cache stores roughly the same number of pages
  - Page location: a client can easily locate the cache storing a particular page

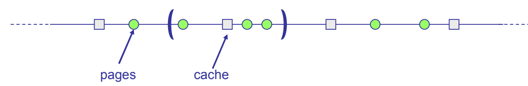
But…

- What happens when the number of caches change?
  - Virtually every page will change its location!
    - » $(ax + b) \bmod m \rightarrow (ax + b) \bmod (m+1)$
- What happens when a user know only a subset of caches (i.e., users have different views) ?
  - Each user will look on a different cache for the same page



Solution: Consistent Hashing

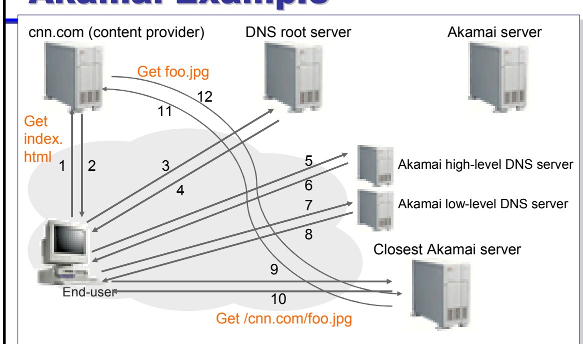
- Assume
  - Each cache (server) is identified by an *id* uniformly distributed in range [0, 1]
  - The key of each page is uniformly distributed within the same range [0, 1]
- A page is stored to the cache (server) which is the **closest** in the identifier space



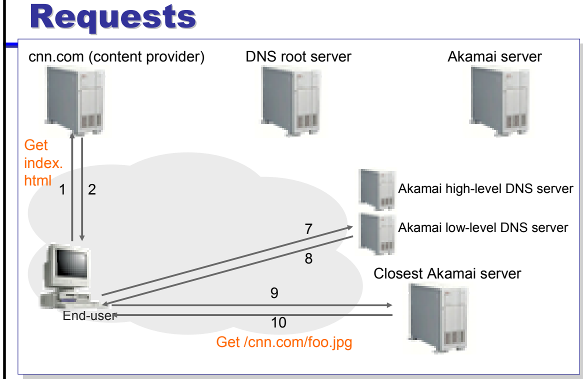
Consistent Hash – Advantages

- Monotone → addition of bucket does not cause movement between existing buckets
- Spread & Load → small set of buckets that lie near object
- Balance → no bucket is responsible for large portion of unit interval

Akamai Example



Akamai – Subsequent Requests



Caveats

- Approach only applies to static objects
  - Amazon Web page is different for everyone
- Assumes IP address of DNS request is correct
- Need good metric to capture "closeness" in network to get best performance
- Based on "pull"-model… what about suddenly popular content?
- However, in practice, is very effective

Summary

- Content distribution
  - Replicate content to improve response time/overhead
- Issues
  - How to replicate content
  - How to select best replica
  - How to direct client to replica

Next time...

- Peer-to-peer networks
  - Napster, Gnutella, KaZaA, Chord/CFS, etc.