Caching & CDN's

- Assigned reading
  - [K+99] Web Caching with consistent hashing

Background

- HTTP: L7, simple protocol, works over TCP
  - Stateless, request/response protocol
  - About 80% of Internet traffic
  - Flavors: parallel, persistent HTTP
  - Methods: GET most common
  - Workload: popularity of objects show zipf-distribution
HTTP support for caching

- Conditional requests (IMS)
- Servers can set expires and max-age
- Request indirection: application level routing
- Range requests, entity tag
- Cache-control header
  - Requests: min-fresh, max-stale, no-transform
  - Responses: must-revalidate, public, private, no-cache

Overview

- Web caches
  - Aspects
  - Cache hierarchies – location of content
  - problems
- Content distribution networks
- New directions

Aspects

- Why web caching?
- Cache consistency
- Source of cache misses
- Caching: where in the network?
- Cache placement/replacement

Why web caching?

- Client-server architecture is inherently unscalable
  - Proxies: a level of indirection
- Reduce client response time
  - Direct and indirect effect
  - Less load on the server:
    - Server does not have to over-provision for slashdot effect
- Reduce network bandwidth usage
  - Wide area vs. local area use
  - These two objectives are often in conflict
    - May do exhaustive local search to avoid using wide area bandwidth
    - Prefetching uses extra bandwidth to reduce client response time
Web Caching - advantages
- Also used for security
  - Proxy is only host that can access Internet
  - Administrators make sure that it is secure
- Performance
  - How many clients can a single proxy handle?
- Caching
  - Provides a centralized coordination point to share information across clients
- How to index
  - Early caches used file system to find file
  - Metadata now kept in memory on most caches

Obscure advantages
- Connection caching [Feldmann 1999]
  - HTTP: small objects, overhead in setting up connection
  - Multiplex multiple requests over single persistent HTTP connection
  - Proxy maintains persistent HTTP connections to clients and servers
- Split TCP connection
  - TCP throughput increases as RTT decreases

Cache consistency - leases
- Only consistency mechanism in HTTP is for clients to poll server for updates
- Should HTTP also support invalidations?
  - Problem: server would have to keep track of many, many clients who may have document
  - Possible solution: leases
  - Leases – server promises to provide invalidates for a particular lease duration
  - Server can adapt time/duration of lease as needed
    - To number of clients, frequency of page change, etc
  - Proxies make leases scalable

Proxies – cache misses
- Capacity
  - How large a cache is necessary or equivalent to infinite
  - On disk vs. in memory → typically on disk
- Compulsory
  - First time access to document (large caches)
  - Non-cacheable documents
    - CGI-scripts
    - Personalized documents (cookies, etc)
    - Encrypted data (SSL)
- Consistency
  - Document has been updated/expired before reuse
  - Conflict → no such issue
Cache Hierarchies

- Use hierarchy to scale a proxy
  - Why?
    - Larger population = higher hit rate (less compulsory misses)
    - Larger effective cache size
  - Why is population for single proxy limited?
    - Performance, administration, policy, etc.
- NLANR cache hierarchy
  - Most popular
  - 9 top level caches
  - Internet Cache Protocol based (ICP)
  - Squid/Harvest proxy
  - How to locate content?

ICP (Internet cache protocol)

- Simple protocol to query another cache for content
- Uses UDP – why?
- ICP message contents
  - Type – query, hit, hit_obj, miss
  - Other – identifier, URL, version, sender address
  - Special message types used with UDP echo port
    - Used to probe server or “dumb cache”
  - Query and then wait till time-out (2 sec)
  - Transfers between caches still done using HTTP
ICP vs HTTP
• Why not just use HTTP to query other caches?
• ICP is lightweight – positive and negative
  • Makes it easy to process quickly
  • HTTP has many functions that are not supported by ICP
  • Extra RTT (2 sec) for any proxy-proxy transfer
  • Does not scale to large number of peers

Optimal Cache Mesh Behavior
• Ideally, want the cache mesh to behave as a single cache with equivalent capacity and processing capability
• ICP: many copies of popular objects created – capacity wasted
• More than one hop needed for searching object
• Locate content – how?

Hinting
• Have proxies store content as well as metadata about contents of other proxies (hints)
  • Minimizes number of hops through mesh
  • Size of hint cache is a concern – size of key vs. size of document
• Having hints can help consistency
  • Makes it possible to push updated documents or invalidations to other caches
• How to keep hints up-to-date?
  • Not critical – incorrect hint results in extra lookups, not incorrect behavior
  • Can batch updates to peers

Summary Cache
• Primary innovation – use of compact representation of cache contents
  • Typical cache has 80 GB of space and 8KB objects → 10 M objects
  • Using 16byte MD5 → 160 MB per peer
  • Solution: Bloom filters
• Delayed propagation of hints
  • Waits until threshold %age of cached documents are not in summary
  • Perhaps should have looked at %age of false hits?
Errors tolerated

- Suppose A and B share caches, A has a request for URL r that misses in A,
  - false misses: r is cached at B, but A didn’t know
    Effect: lower total cache hit ratio
  - false hits (false +ves): r is not cached at B, but A thought it is
    Effect: wasted query messages
  - stale hits: r is cached at B, but B’s copy is stale
    Effect: wasted query messages

Bloom Filters

- Proxy contents summarize as a M bit value
- Each page stored contributes k hash values in range [1..M]
  - Bits corresponding to the k hashes set in summary
- Check for page = if all k hash bits corresponding to a page are set in summary, it is likely that proxy has summary
- Tradeoff \( \rightarrow \) false positives
  - Larger M reduces false positives
  - What should M be? 8-16 * number of pages seems to work well
  - What about k? Is related to (M/number of pages) \( \rightarrow \) 4 works for above M

Hierarchy Problems – Population Size

- How does population size affect hit rate?
- Critical to understand usefulness of hierarchy or placement of caches
- Issues: frequency of access vs. frequency of change (ignore working set size \( \rightarrow \) infinite cache)
- UW/Msoft measurement \( \rightarrow \) hit rate rises quickly to about 5000 people and very slowly beyond that
- Proxies/Hierarchies don’t make much sense for populations > 5000
  - Single proxies can easily handle such populations
  - Hierarchies only make sense for policy/administrative reasons

Problems – Common Interests

- Do different communities have different interests?
  - I.e. do CS and English majors access same pages? IBM and Pepsi workers?
- Has some impact \( \rightarrow \) UW departments have about 5% higher hit rate than randomly chosen UW groups
  - Many common interests remain
- Is this true in general? UW students have more in common than IBM & Pepsi workers
- Some related observations
  - Geographic caching – server traces have shown that there is geographic locality to interest
  - UW & MS hierarchy performance is bad – could be due to size or interests?
Problems with caching we just saw

- Over 50% of all HTTP objects are uncacheable.
- Sources:
  - Dynamic data → stock prices, frequently updated content
  - CGI scripts → results based on passed parameters
  - SSL → encrypted data is not cacheable
    - Most web clients don’t handle mixed pages well → many generic objects transferred with SSL
  - Cookies → results may be based on passed data
  - Hit metering → owner wants to measure # of hits for revenue, etc, so, cache busting

Problems

- Aborted transfers
  - Many proxies transfer entire document even though client has stopped → eliminates saving of bandwidth
- Client misconfiguration
  - Many clients have either absurdly small caches or no cache
- Session –
  - HTTP: stateless
  - Not much interesting things can be done
  - Sessions needed for e-commerce

Resurrection of caching

- Caching is good.
- Economic motive for both the client and the server.
- Just described – client caching – problems:
  - Content providers do not have enough control.
  - Dynamic content, personalization is on the increase → static caching no longer suffices.
- Emergence of server farms, caches at various stages, and content delivery networks.

Overview

- Web caches
- Content distribution networks
- New Directions
CDN

- Replicate 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

Server Selection

- Service is replicated in many places in network
- How to 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 from client to “closest” service node
    - Closest is defined by routing metrics
    - May not mirror performance/application needs
  - What about the stability of routes?

Routing Based

- Cluster load balancing
  - Router in front of cluster of nodes directs packets to server
  - Can only look at global address (L3 switching)
  - Often want to do this on a connection by connection basis – why?
    - Forces router to keep per connection state
    - L4 switching – transport headers, port numbers
  - 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
L-7 switching

- Interpret requests, content-aware switches
- Have to do the initial hand-shake
- Different proxies for different content-types
- Load balancing vs locality
- Locality means all requests (even to a popular object) serviced by a single proxy
- Caching alleviates the above problem. Why?

Application Based

- HTTP supports simple way to indicate that Web page has moved
- Server gets 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.
- While good solution in general HTTP Redirect has some design flaws – especially with current browsers?

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
  - Name service client
    - Typically the local name server for client
- Round-robin
  - Randomly choose replica
  - Avoid hot-spots
- (Semi-)static metrics
  - Geography
  - Route metrics
  - How well would these work?

How Akamai Works

- Clients fetch html document from primary server
  - E.g. fetch index.html from cnn.com
- URLs for replicated content are replaced in html
  - E.g. `<img src="http://cnn.com/af/x.gif">` replaced with `img src="http://a73.g.akamaitech.net/7/23/cnn.com/af/x.gif"`>
- Client is forced to resolve aXYZ.g.akamaitech.net hostname
How Akamai Works

• How is content replicated?
• Akamai only replicates static content
  • Serves about 7% of the Internet traffic!
• Modified name contains original file
• Akamai server is asked for content
  • First checks local cache
  • If not in cache, requests file from primary server and caches file

How Akamai Works

• 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
  • TTL is large
• G.akamaitech.net nameserver choses server in region
  • Should try to chose server that has file in cache - How to choose?
  • Uses aXYZ name and consistent hash
  • TTL is small

Hashing

• Advantages
  • Let the CDN nodes are numbered 1..m
  • Client uses a good hash function to map a URL to 1..m
  • Say hash (url) = x, so, client fetches content from node x
  • No duplication – not being fault tolerant.
  • One hop access
  • Any problems?

Hashing

• Advantages
  • Let the CDN nodes are numbered 1..m
  • Client uses a good hash function to map a URL to 1..m
  • Say hash (url) = x, so, client fetches content from node x
  • No duplication – not being fault tolerant.
  • One hop access
  • Any problems?
  • What happens if a node goes down?
  • What happens if a node comes back up?
  • What if different nodes have different views?
Robust hashing

- Let 90 documents, node 1..9, node 10 which was dead is alive again
- % of documents in the wrong node?

Robust hashing

- Let 90 documents, node 1..9, node 10 which was dead is alive again
- % of documents in the wrong node?
- 10, 19-20, 28-30, 37-40, 46-50, 55-60, 64-70, 73-80, 82-90
- Disruption coefficient = ½
- Unacceptable, use consistent hashing – idea behind Akamai!

Consistent Hash

- “view” = subset of all hash buckets that are visible
- Desired features
  - Balanced – in any one view, load is equal across buckets
  - Smoothness – little impact on hash bucket contents when buckets are added/removed
  - Spread – small set of hash buckets that may hold an object regardless of views
  - Load – across all views # of objects assigned to hash bucket is small

Consistent Hash – Example

- Construction
  - Assign each of C hash buckets to random points on mod 2^n circle, where, hash key size = n.
  - Map object to random position on circle
  - Hash of object = closest clockwise bucket
- Smoothness → addition of bucket does not cause much movement between existing buckets
- Spread & Load → small set of buckets that lie near object
- Balance → no bucket is responsible for large number of objects
How Akamai Works

End-user → cnn.com (content provider) → DNS root server → Akamai server → Akamai high-level DNS server → Akamai low-level DNS server → Closest Akamai server → End user

Get index.html

Get foo.jpg

Get /cnn.com/foo.jpg

Overview

• Web caches
• Content distribution networks
• New Directions

Future directions

• Key to scaling: move interactions as close to the user as possible.
• Economic motive: cache at edge servers (pay as much as you use + improved performance).
• Have content pushed to the client via a content delivery network.
• Content provider need not worry about the distributed nature of content delivery.
Simple idea

Execute code fragments

- Execute code fragments at the edge server
- Code fragments – higher reuse
  - Depending on the input, produce the output.
- E-commerce applications require a session abstraction
  - Have support for session tracking.
- All benefits of caching!
- Problem?

Database caching

- Applications need to access the data.
- Multiple cache misses – multiple RTT latencies to execute code.
- Solution: Cache and prefetch data.
- Use program analysis to figure out what data is required, get it ahead of time, hide latency.

Next Lecture: P2P

- Peer-to-peer networks
- Assigned reading
  - [Cla00] Freenet: A Distributed Anonymous Information Storage and Retrieval System
  - [S+01] Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications
Have a good break!