

# Building HTTP Caches with Webrick

Doug Beaver - Ruby in the Rainy City - 12/10/2004

# HTTP Caches

- Caches (or web accelerators) are used all the time on the web, you just don't know it.
- HTTP lends itself to easy caching, since all requests have “keys” built into them, and all responses from remote servers include a built-in TTL for the returned data.
- Required technologies are a HTTP toolkit, a date parsing library, and a MD5 digest library.

# Proxy Caches

- A proxy server that transparently caches requested objects in memory or on disk. This is what most people mean when they say “web accelerator”. Easy to use, you just change your browser’s settings to point at the proxy, and you automatically start surfing the web more quickly. It caches the request results for outbound HTTP traffic.

# Reverse Proxy Caches

- A cache that sits between border servers and backend web servers, proxying inbound requests to the backend servers and sending their results back to the external requestor.
- A reverse proxy could answer for foo.com and forward requests to backend web servers (www1, www2, www3, etc). External users only see foo.com, they have no idea what the internal network topology looks like.

# Reverse Proxies II

- Not all reverse proxies are caches. Non-caching reverse proxies usually fill the role of load balancer or SSL accelerator.
- SSL acceleration is an interesting use of reverse proxies; custom hardware is used to support high numbers of concurrent SSL connections, and the SSL accelerator “wraps” the backend non-SSL web servers with SSL. This allows a more homogenous online fleet, and easier partitioning of SSL traffic.

# What Not to Cache

- POST requests - Typically used in form posts, which should be kept confidential. Additionally, POSTs can be quite large depending on the usage (file upload, etc).
- HEAD requests - By nature, a HEAD request is meant to be a quick realtime check of the headers for a given url.
- SSL traffic (inbound or outbound) - You never cache secure or authenticated traffic. Ever.

# What requests are left?

- GET requests!
- GETs are used for grabbing html and media content (images, embedded video, etc). The vast majority of internet HTTP traffic is in the form of GET requests.
- The rule of not caching POSTs and HEADs can be bent if you have control over the inbound requests. The SSL rule should never be broken.

# Let's do it!

- A caching proxy is simple to design. You need:
- HTTP proxy engine (input/output sockets, library for creating properly formatted HTTP requests / responses)
- Object storage system for cached data
- Object expiration system
- Cache configuration mechanism

# HTTP Engine

- We're going to use Webrick, which already includes HTTP proxy classes and an easy way to write servlets and mount them on given url prefixes.
- Webrick is powerful enough that we can implement reverse proxies as well (although that is an exercise left for the reader or a future presentation).

# Object Storage

- Reliable storage of cached objects requires:
- Balanced filesystem tree for storing objects.
- Quick addressing and loading of cached objects.
- Metadata system to lookup critical information about cached objects (age, size, HTTP status code).

# Object Storage II

- MD5 gives very quick cache checks.
- Take the MD5 hex digest of the GET string, and use the last N chars to create a directory bucket structure:
- `$checksum = getMD5(getString)`
- last 4 of \$checksum could be `C/9/A/I`
- store object at `cache/C/9/A/I/$checksum.data`
- Max of 16 dirs per bucket level; 65,536 total.

# Object Storage III

- The checksum.data file contains the results for the request, including the HTTP response headers.
- This allows quick lookup to see if there are cached results for a given request.
- Metadata for age and size come from file stat information on checksum.data.

# Object Expiration

- Out-of-band expiration is difficult when external object metadata is not maintained.
- When we get a cache hit for a given URL, we'll parse the HTTP headers from the checksum.data file and make sure the expiration date has not been hit yet.
- If object has expired, a request is made to the remote server and that new results data is placed in checksum.data, and then returned to the requestor.

# Cache Configuration

- Ports and hostnames
- Cache all hosts or just some?
- Virtual URLs?

# Cache Config II

- YAML is an easy way to setup cache config and keep things editable.

```
port: 5000
host: cacher.mydomain.com
maps:
  /y/*: www.yahoo.com
  /a/*: www.amazon.com
ignore:
  - ebay.com
  - metafilter.com
```

# Enhancements

- Simple implementation has no cap on disk usage. Metadata should be maintained on object size and creation date, which lets us use a LRU (least recently used) algorithm to cap disk usage at X MB. A Berkeley db would be a good way to store this, with the key being the epoch time of creation, padded with a ASCII control char, and then the size of the file in bytes. The value would be the cache id for the object.
- `1102747169\00120453 => $id` gives us a file created today with a size of 20453 bytes.

# Inventive Cache Usage

- Use REST web services for your website and use cheap reverse proxy caches to store results for some or all service calls.
- Use reverse proxy to balance load across servers in multiple geographic areas to guard against local interruptions of service.
- Remember - Don't be afraid to use caches on your internal or external network!