Chapter 1: HTTP Performance

It's a non functional requirement
They can be measured in terms of

- requests
- volume
- latency
- bandwidth
- utilization

Before you Optimize

1. Measure Something
2. Record the performance metric
3. Record and track the original settings for that metric
4. Run the tests multiple times to get a sample. Avoid luck and outliers

Because we are using statistics to quantity and compare performance, we use statistical analysis to quantify and determine if there is a quantifiable difference. To determine if there is an actual difference then make a t-test comparing the two performances and see if the difference is significant.

Web Performance Best Practices

Google says:

- Optimizing caching - keeping your application's data and logic off the network altogether
- Minimizing round-trip times - reducing the number of serial request response cycles
- Minimizing request overhead - reducing upload size
- Minimizing payload size - reducing the size of responses, downloads, and cached pages
- Optimizing browser rendering - improving the browser's layout of a page
- Optimizing for mobileNew! - tuning a site for the characteristics of mobile networks and mobile devices

Caching

- Caching increase locality
- Locality increases bandwidth
- Locality decreases latency
- Levels of cache:
  - CPU
  - Memory
  - Disk
  - Network

Use `ETags` and `Last-Modified` headers.

User Agent Cache

> Also, if the response does have a Last-Modified time, the heuristic expiration value SHOULD be no more than some fraction of the interval since that time. A typical setting of this fraction might be 10%. – RFC2616 section 13

This means if it was modified 10 minutes ago, you should probably hit it up again in a minute.

Thus it is up to the browser to emit a request. They do so upon expiry or last modified time heuristic. Or the user forces a refresh like CTRL-SHIFT-R or ctrl-shit click on the refresh button. In browser cache is the most local and high performance cache!

Cache-Control

Generally sent by User Agent. Indicates how they want to handle this request. It signals proxies and caches how to handle the request.

no-cache

You must revalidate - We didn't give it a time

- A 304 response is fine
- Forces a request out to the server
- max-age=0 means the same thing

no-store

Don't store anything

- Suggests that the results are not-cacheable and emphemeral.
- Will not act as DRM

max-age=

Cache-Control: max-age=seconds in a HTTP response tells the Use-Agent the maximum age they should let this resource last

- Easy to deploy
- Cache-Control: max-age=259200 = 3 Days
- Benefit: no date math for you!
- Benefit: No date formatting!
- Disadvantage: Have to predict max-age!

Header specific caching

These are the header specific for caching

Expires

Expires tells the Use-Agent after which date they should ask for a new instance of the resource.

- Easiest to deploy
- Very simple
- Causes lots of problems if set wrong!

If-Modified-Since

- Conditional HTTP Request
- Return a 304 if not modified since
- If-Modified-Since: Mon, 07 Apr 2014 03:00:05 GMT. Don't send me anything new unless the resource has been modified after that time.
- If the response is anything but a 200 OK, return a normal response instead of the 304

Last-Modified

> The Last-Modified entity-header field value is often used as a cache validator. In simple terms, a cache entry is considered to be valid if the entity has not been modified since the LastModified value. - HTTP RFC 2616 Fielding et al.

Last-Modified is a date that the resource was last modified

- Used for simple caching
- Requires the HTTP server to respond

ETag

What if you cannot guess or estimate the time that content will be safe? What if content updates all the time, unpredictably? What if content updates but all changes aren't that important:

- E.g. your age does increase every second but maybe it isn't important to caching to have your age updated per each second?

How do you make it?

- If strong (exact content) just use a hash like SHA1
- If weak then hash some content you think is relevant and prefix with `W/"etagvalue"` to indicate it is a weak hash
- If hashing is pointless make an etag of actual values in plaintext
- Keep it short

HTTP Response Header

- Contains a name or tag indicating the content or revision of a resource.
  - Is not date related
  - Tends to be content related
  - Can be any value
  - Can use any hash

Dangers of ETags

Cookies part II

- Etags allow for vendors (advertisers) to finger print your client because your client will send the etags back.
- If you deny cookies, you tend to send etags

Too much information

- Some Etags contain irrelevant information!
- What if the browser reboots and the Etags are lost?

Cross Cutting Concern

Performance is a cross cutting concern because it interacts with other functionality:

- Security

- Lack of encryption means global proxying
- Authentication can limit caching
- Authentication can imply state

- State

- State can limit caching
- State can limit layering

Round Trips

DNS Lookups, Connections, HTTP transactions

- Async is fast: Just send it! Who cares when it arrives
- Rountrips are synchronous and slow: We must wait for a response!
- Avoid HTTP Redirects that aren't cacheable
- Rewrite Server Side
- Avoid too many HTTP hosts
- Can you piggyback?
- CSS Sprites are often recommended to reduce number of image requests
- Avoid CSS imports

Round Trip Tricks

Use multiple static content hostnames:

- Take a hit in DNS lookup
- But improve parallel download performance
- Static hosts should not be dynamic and have stable IPs

Reduce Request Size

Giant Cookies - NOOOOO

- Giant URIs – NOOOO
- Too many headers? NOOOO
- Remember all that networking we went over?

Try to fit within the MTU!

Avoid Dynamism and Cookies for Static Content

For static content, do GETs to get it

- For static content avoid dynamism and cookies
- Cookies imply state and can mess up caching
- Use separate domains for static content to
- avoid statefulness

Minimize Resource Size

- Images - too big
- Javascript - minify (I dislike this one)
- GZIP Encoding!
- Sound - too big
- Video - too big
- You can fake Sound and Video in JS!

Minimize Number of Resources

- 1 or 0 CSS Files
- 1 or 0 Javascript Files
- 1 or 0 Images (CSS Sprites)
- 1 or 0 HTML Files

You could generate a page in JS and take no hit. 1 giant page has the problem if it is dynamic, but if it is 1 giant page that does dynamic things you can cache that page and never have to get HTML/JS/CSS again.

Optimize Rendering

Recommendations:

- CSS at the top
- Javascript at the bottom
- Content in the middle
- Give appropriate sizes and hints
- The layouter is quite expensive, give some hints and it will go to town.

Defer JavaScript

Nasty trick:

- Encode javascript as a string and eval it later when you need it.
- Avoids heavy page loading and JS parsing with the browser is working hard.
- CPU driven

Content Delivery Network

- These are global networks of content providers that can mirror your content.
- Excellent latency and locality
- Great for static stuff
- $$$

GZIP things

- Turn on GZIP encoding to make things smaller
- Watch out! This affects latency
- Improves bandwidth
- Balancing act
- Generally recommended

Fix DNS

- Use A and AAAA records instead of CNAMEs
- Have your authoritative name server give the results
- Allow for caching of DNS requests to avoid excessive lookups
- Provide many A records in 1 response
- Reduce number of hosts on 1 page
- Some recommend using CNAME to allow multiple connections (so it depends!)

Avoid Indirection

- Redirections
- Imports
- Dynamic choosing of content
- Javascript downloading of content

Avoid POST

POST is not idempotent

- POST is not cacheable
- POST is dynamic
- POST smashes all the performance infrastructure in a fine powder and blows it into your face.

Check for errors

403s, 404s, 410s, etc. Are all slow

- Often the server works harder to serve an error than it does to serve real content.
- Errors cause exceptions, exceptions cause latency and pain and reporting.
- Errors cause logging - more IO
- Errors are worthless requests hogging up resources

Encoding

Sending a JSON encoding is not always appropriate in terms of size.

- Sending an audio stream as ASCII text? Bad idea. Use an appropriate format.
- Do you need lossy or losseless?
- Do you need XML? JSON? CSV? Binary?

Repeat yourself or don't repeat yourself

Cache matters if the cache can avoid you repeating yourself then go for it. But sometimes denormalizing data and providing duplicate information avoids more requests.

Async over Sync

Asynchronous means that you can do other things while something is occurring.

- Synchronous is blocking which implies latency.
- Synchronous means round trips
- Async means parallelizable
- Synchronous means serialized

Javascript Includes

Some people like to include common libraries from the library homepage.

- Benefit: Someone else has done this so the user has cached it.
- Disadvantage: What if they get hacked
- Disadvantage: What if they are slower than you?
- Disadvantage: What if you lose locality?