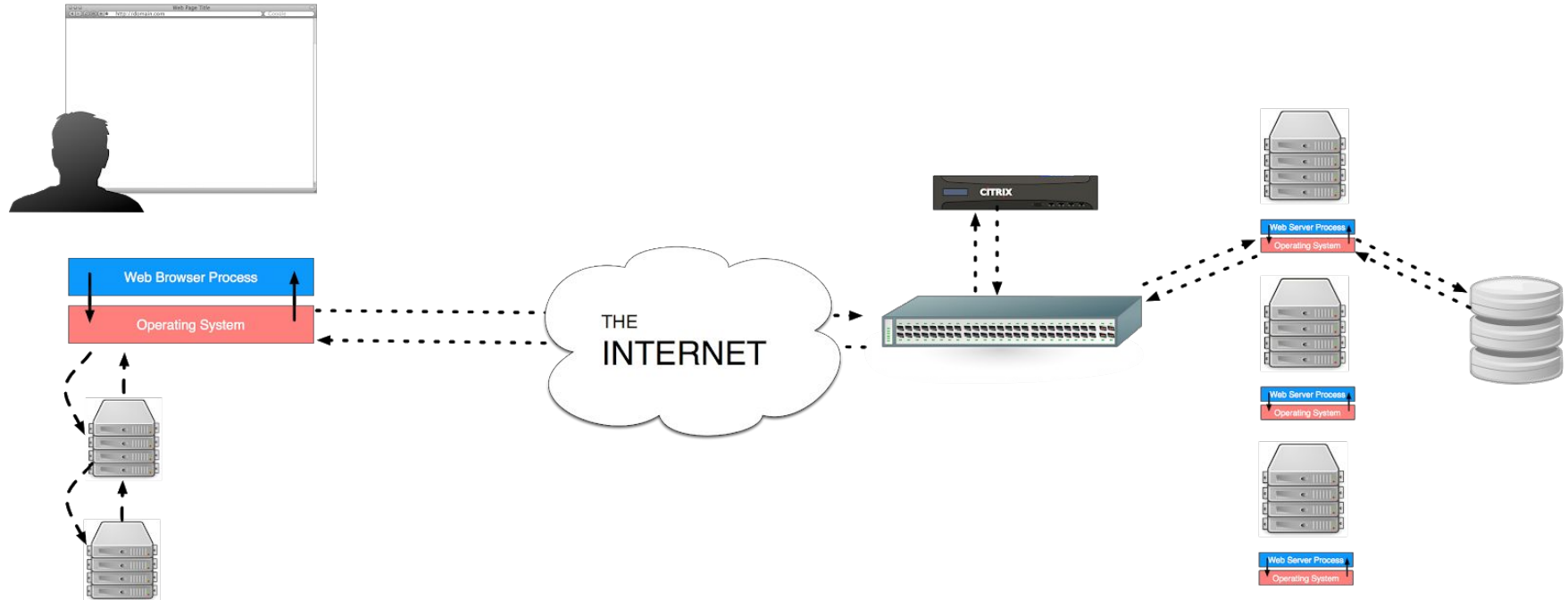


CS188

Scalable Internet Services

John Rothfels, 10/22/20

Motivation



Motivation

We want our important application data persisted safely in our data center.

And it needs to be regularly read and updated by geographically distributed clients.

And it needs to be fast. 

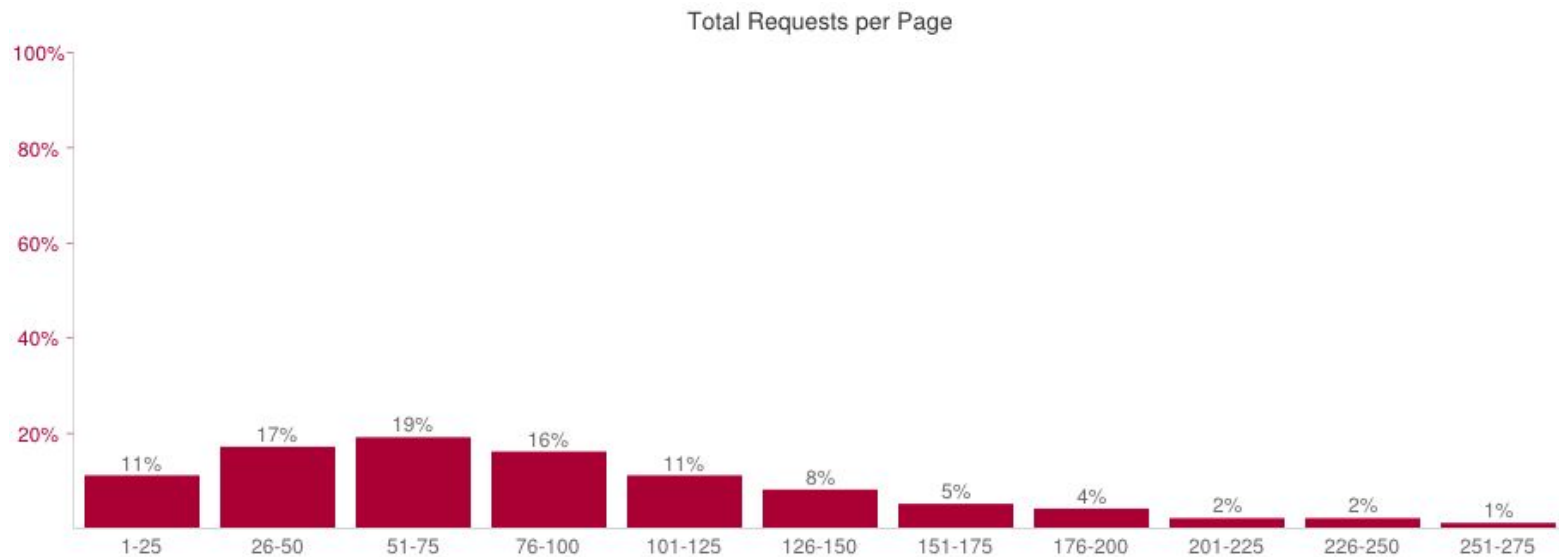
Motivation

Performance matters!

Delay	User Reaction
0 - 100ms	Instant 😍
100 - 300ms	Slight perceptible delay 😊
300 - 1000ms	Task focus, perceptible delay 😐
1 second +	Mental context switch 😬
10 seconds +	I'll come back later ☐

Motivation

A page is more than a single request...



[source](#)

Motivation

The fastest request is one that never happens!

A **cache** can store data so that future requests for that data can be served faster.

! ? Where can we put caching?

Motivation

The fastest request is one that never happens!

A **cache** can store data so that future requests for that data can be served faster.

! ? Where can we put caching?

- Inside the browser (local storage)
- In front of the server (CDNs, etc.)
- Inside the application server (RAM)
- On a remote cache server (Redis, Memcached)
- Inside the database (query cache)

Client-side caching

! ? How does the browser cache data? How does it know when it can present previously seen data as current?

Client-side caching

!? How does the browser cache data? How does it know when it can present previously seen data as current?

The building blocks are HTTP headers!

- etag
- cache-control
 - max-age | no-cache | no-store | public | private
- if-modified-since
- if-none-match

Client-side caching

cache-control: no-store


When accompanying a response, the browser (or intermediate proxy) is instructed to not reuse this data under any circumstances.

This can also used for sensitive information.

Client-side caching

cache-control: no-cache

When accompanying a response, the browser (or intermediate proxy) is instructed to revalidate before reusing it.

 It doesn't mean “don't use the cache” -- it means get the value from the server before using it (but you could save it in the cache).

Client-side caching

cache-control: private

When accompanying a response, the browser (or intermediate proxy) is instructed that the data is specific to the requesting user.

Intermediate proxies should discard it, but a single-user browser can reuse it.

The opposite of this is **cache-control: public**

Client-side caching

`cache-control: max-age=120`

When accompanying a response, the browser (or intermediate proxy) should consider this copy stale if the specified number of seconds has passed.



This is the more modern version of the `expires` and `date` headers.

Client-side caching

etag: 5bf444d26f9f1c74

When accompanying a response, the browser will keep this “**entity tag**” along with saved copies of the resource.

When requesting the same resource in the future, this tag can be presented to indicate the version it had previously seen.

This isn't necessarily a digest of the resource that was served up, but can be thought of as such.

Client-side caching

`if-modified-since: Sun, 19 Oct 2014 19:43:31`

When accompanying a request, this indicates that the client already has a copy that was fresh as of the specified date.

If the server's copy is newer than the specified date, it will be served to the client.

If the server's copy hasn't changed since the specified date, the server will return 304 (not modified).

Client-side caching

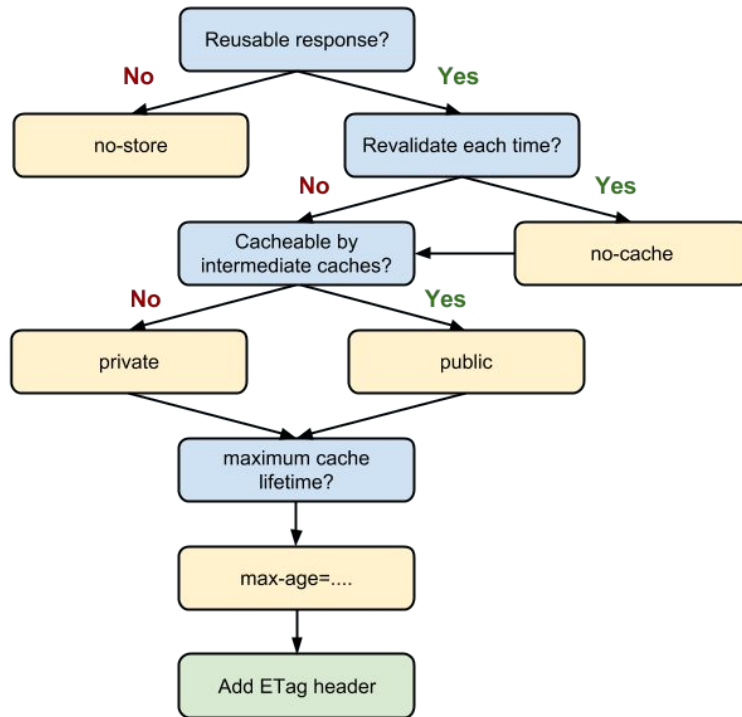
if-none-match: 5bf444d26f9f1c74

When accompanying a request, this indicates that the client has a cached copy with the associated tag. Multiple etags can be provided.

If the server's current version has one of the etags listed, the server will return 304 (not modified) with the etag of the current resource included.

If the server's version has a non-matching etag, then the result will be returned as normal.

Client-side caching



Client-side caching

Let's say we are serving up some JavaScript that won't change over the next day, but does have some user-specific code in it.

! ? What headers should the response include?

Client-side caching

Let's say we are serving up some JavaScript that won't change over the next day, but does have some user-specific code in it.

!? What headers should the response include?

We want it reusable, but private.

```
cache-control: private, max-age=86400
```

Client-side caching

Let's say we are serving up an image that may be changing in the future, and we never want a stale version shown. The image is not specific to the requestor.

! ? What headers should the response include?

Client-side caching

Let's say we are serving up an image that may be changing in the future, and we never want a stale version shown. The image is not specific to the requestor.

!? What headers should the response include?

We want it reusable with revalidation and public.

Cache-control: public, no-cache

ETag: 4d7a6ca05b5df656

Clients will request the resource with:

if-none-match: 4d7a6ca05b5df656

Client-side caching

Let's say we are serving up an image with the user's social security and credit card numbers.

!? What headers should the response include?

Client-side caching

Let's say we are serving up an image with the user's social security and credit card numbers.

!? What headers should the response include?

`Cache-control: no-store`

Client-side caching

! ? How does client-side caching work with GraphQL?

Client-side caching

! ? How does client-side caching work with GraphQL?

It's not what you'd expect...

- Requests (including fetches) are made using **POST**, which shouldn't do any caching
- Fetching data fills a piece of the data graph (in the browser / JavaScript)
- Requesting a piece of the data graph that has already been fetched can be satisfied from the result of the previous fetch...if we cache it!

Client-side caching

When making a query with Apollo, you specify a **fetch policy** to determine how to use the cache.

The default policy will use cached data if available.

And avoid network requests if possible! 

NAME	DESCRIPTION
<code>cache-first</code>	<p>Apollo Client first executes the query against the cache. If <i>all</i> requested data is present in the cache, that data is returned. Otherwise, Apollo Client executes the query against your GraphQL server and returns that data after caching it.</p> <p>Prioritizes minimizing the number of network requests sent by your application.</p> <p>This is the default fetch policy.</p>
<code>cache-only</code>	<p>Apollo Client executes the query <i>only</i> against the cache. It never queries your server in this case.</p> <p>A <code>cache-only</code> query throws an error if the cache does not contain data for all requested fields.</p>
<code>cache-and-network</code>	<p>Apollo Client executes the full query against both the cache <i>and</i> your GraphQL server. The query automatically updates if the result of the server-side query modifies cached fields.</p> <p>Provides a fast response while also helping to keep cached data consistent with server data.</p>
<code>network-only</code>	<p>Apollo Client executes the full query against your GraphQL server, <i>without</i> first checking the cache. The query's result <i>is</i> stored in the cache.</p> <p>Prioritizes consistency with server data, but can't provide a near-instantaneous response when cached data is available.</p>
<code>no-cache</code>	<p>Similar to <code>network-only</code>, except the query's result <i>is not</i> stored in the cache.</p>
<code>standby</code>	<p>Uses the same logic as <code>cache-first</code>, except this query does <i>not</i> automatically update when underlying field values change. You can still <i>manually</i> update this query with <code>refetch</code> and <code>updateQueries</code>.</p>

Client-side caching

If rendering your React components on the server, requests for GraphQL data can be satisfied by Apollo client on the server before your HTML is sent to the browser.

! ? How do we avoid re-requesting the data from the browser?

Client-side caching

If rendering your React components on the server, requests for GraphQL data can be satisfied by Apollo client on the server before your HTML is sent to the browser.

!? How do we avoid re-requesting the data from the browser?

Send the Apollo cache data from the server to the browser.

Initialize the browser Apollo client with the cached data from the server.

Client-side caching

Using Apollo, we get several features on top of caching:

- Ability to read/write directly to/from the cache (avoiding the network)
- Ability to “subscribe” React components to a data query
 - Any component with `useQuery` specifies a piece of the data graph. If that piece of the graph is updated in the cache, Apollo can trigger a re-render of the component without making another request to the network
- Ability to manage local state alongside remotely fetched state
 - Interact with *all* of your application’s state via GraphQL, even if it isn’t fetched from a server. You can save data in the browser and query it via GraphQL w/ Apollo.

Client-side caching

! How do we invalidate data in the Apollo cache?

Client-side caching

! ? How do we invalidate data in the Apollo cache?

- re-fetch it from the server
 - On demand?
 - Polling?
- write new data to the cache manually
 - ! ? When would we ever want to do this?
- evict it manually

With great power comes great responsibility. And greater bugs. 