




Caching is Hard


Matt Robenolt
@mattrobenolt

So, what is Disqus?

So, what is Disqus?

3 comments

 78




Leave a message...


Post as Stranger


Newest ▾

Community

My Disqus

Share 






Winston Churchill · a minute ago

It looks like we have a new visitor amongst us. Hey stranger, what's your name and what are you doing here?


10 ^ | ▾ Reply Share ›



Marilyn Monroe · a minute ago

What just happened? Where are we anyway?


7 ^ | ▾ Reply Share ›




Elvis Presley → Marilyn Monroe · a minute ago

I think you mean... when are we?

10 ^ | ▾ Reply Share ›

 Comment feed

 Subscribe via email

DISQUS

 Comment feed

 Subscribe via email

DISQUS

So, what is Disqus?

ALSO ON THE STARS OF THE WORLD BLOG

Tyson Crazy? Or Is He Just Winking?



Mike Tyson — I'm not crazy! I'm the former heavyweight champion of the world! I had to have some brains to get this far.

The Real Marilyn, In Her Own Words



Marilyn Monroe — Fame will go by and, so long, I've had you, fame. If it goes by, I've always known it was fickle. So at least it's something I experience, but that's not where I live.

Brando: The Lion of the Screen



Marlon Brando — Nobody tells me what to do. You keep needlin' me, if I want to, I'm gonna take this joint apart.

Is Tupac Coming Back?



Tupac — I don't have no fear of death. My only fear is coming back reincarnated. I don't see myself being special; I just see myself having more responsibilities than the next man.

Powered by **DISQUS**

Powered by **DISQUS**

live.
something I experience, but that's not where I
I've always known it was fickle. So at least it's
live. So long, I've had you, fame. If it goes by,

next man.
myself having more responsibilities than the
I don't see myself being special; I just see
myself. My only fear is coming back reincarnated.

So, what is Disqus?

```
dsq.src = '//' + Disqus_shortcode + '.disqus.com/embed.js';
```

Our Stack

Well, some of it.

Our Stack



First, some numbers.

First, some numbers.

- ~1 billion unique visitors per month
- ~5MM new threads per day
- Total requests
 - ~35,000/s
- Varnish
 - ~25,000/s
- Django backends
 - ~12,000/s.
- ~66% cache hit ratio
 - Could be **much much** better

Let's talk about HTTP clients.

Clients do terrible things.

Clients do terrible things.

- ◉ Cookies
- ◉ Cache busting tokens
 - ◉ `/embed.js?_1234567`
- ◉ Querystrings
- ◉ People **will** use your shit in ways that you never planned.

Let's talk about applications.

Applications do terrible things.

Applications do terrible things.

- CSRF tokens
- Set-Cookie
- Cache-Control: no-cache
- Vary: Cookie

Everything is terrible.

How does the embed load?

How does the embed load?

- JavaScript bootloader
- Load `<iframe>` as anonymous user
- 3 minimum *critical* HTTP requests
- 1 optional API request to fetch user specific data and layer it on top

Request #1

*.disqus.com/embed.js

*.disqus.com/embed.js

```
sub vcl_recv {
    if (req.url ~ "^/embed\.js") {
        error 750;
    }
}
sub vcl_error {
    if (obj.status == 750) {
        set obj.http.Location = "http://go.disqus.com/
embed.js";
        set obj.response = "Found";
        set obj.status = 302;
        return(deliver);
    }
}
```

Ugly... but it works!

*.disqus.com/embed.js

- This request alone is ~10,000/s on average
- Previously hit our slow backends
- Maintain ability to toggle behavior with DNS

Request #2

`go.disqus.com/embed.js`

go.disqus.com/embed.js

```
sub vcl_recv {
    if (req.http.Cookie) {
        set req.http.X-Order =
regsub(req.http.Cookie, "^.*?disqus\.order=([^\;]
+).*?$", "\1");

        if (req.http.X-Order == req.http.Cookie) {
            set req.http.X-Order = "default";
        }
    } else {
        set req.http.X-Order = "default";
    }

    set req.url = "/current/build/next/embed."
req.http.X-Order ".js";

    unset req.http.Cookie;
}
```


go.disqus.com/embed.js

```
sub vcl_fetch {  
    if (req.url ~ "/embed\.\w+\.js$") {  
        set beresp.http.Vary = "Accept-Encoding, X-  
Order";  
        set beresp.http.Cache-Control = "public,  
max-age=10";  
        set beresp.ttl = 10s;  
        set beresp.grace = 24h;  
    }  
}
```

Not bad.

go.disqus.com/embed.js

- Origin fetches from our static media server once every 10 seconds to refresh
- This logic used to be handled by our app on the first request
- Avoids varying by Cookie at the cache level
 - Still vary on Cookie at the client, but meh
- Will serve stale for 24h if we fuck up

Request #3

`disqus.com/embed/comments/`

disqus.com/embed/comments/

```
sub vcl_recv {  
    if (req.url ~ "^/embed/comments/\?") {  
        unset req.http.Cookie;  
    }  
}
```

disqus.com/embed/comments/

```
sub vcl_fetch {
    set beresp.grace = 4h;

    set beresp.ttl =
std.duration(regsub(beresp.http.Surrogate-Control,
"max-age=(\d+)", "\1s"), 60s);
    unset beresp.http.Surrogate-Control;

    unset beresp.http.Vary;
    set beresp.http.Vary = "Accept-Encoding";
    set beresp.http.Cache-Control = "no-cache, public,
must-revalidate";
    unset beresp.http.Set-Cookie;
}
```

Disqus is loaded!

disqus.com/embed/comments/

- Control the cache duration from the app with Surrogate-Control
- We don't want to cache at the client, but we do at the edge
- Explicitly coerced a request to anonymous
- Prevented our app from sending back something stupid
- Works in the event of app failure for up to 4h

disqus.com/embed/comments/

- Can more reliably load the embed and at least show an error message
- If it's a hot thread, it's very likely that it is cached
- Can cache threads longer in the event of high loads

Request #4

`disqus.com/.../threadDetails`

...last one.

disqus.com/.../threadDetails

```
sub vcl_recv {  
    // Remove cache busting token  
    set req.url = regsuball(req.url, "([\?|&])_\d+=1",  
"\1");  
}
```

disqus.com/.../threadDetails

```
sub vcl_recv {
    set req.http.Extension = regsub(req.url, "^.*?
threadDetails(\.[^?]+)?.*?$", "\1");

    // Extract the `thread`
    if (req.url ~ "thread=") {
        set req.http.Thread-Id = regsub(req.url, "^.*?
thread=(\d+)?.*?$", "\1");
    } else {
        set req.http.Thread-Id = "";
    }
}
```

disqus.com/.../threadDetails

```
sub vcl_recv {  
    // Reconstruct a uniform URL  
    set req.url = "/api/3.0/embed/threadDetails" +  
req.http.Extension + "?thread=" + req.http.Thread-Id +  
"&api_key=" + req.http.API-Key;  
  
    // Clean up these "Headers"  
unset req.http.Extension;  
unset req.http.Thread-Id;  
unset req.http.API-Key;  
  
    // Remove trailing &'s and ?'s  
set req.url = regsuball(req.url, "[\?|&]+$", "");  
}
```

disqus.com/.../threadDetails

```
sub vcl_fetch {  
    set beresp.http.Vary = "Accept-Encoding";  
    set beresp.ttl = 5m;  
    set beresp.grace = 15m;  
    unset beresp.http.Set-Cookie;  
}
```

3000 req/s saved.

3000 req/s saved.

- Stopped busting our own cache with a cache busting token
- Large majority of traffic is from anon
- Normalized all anon traffic into a common cache key
- Profit

High-er Availability

High-er Availability

- We pushed our first 3 HTTP requests out of our data center and into Fastly
- This reduces our latency by a ridiculous amount
- Our network is nowhere near as reliable or consistent
- Working towards 5 9s

What did all of this accomplish?

What did all of this accomplish?

- ◉ After a thread has been cached once, our embed can be loaded entirely to working state without hitting our app servers
- ◉ On an uncached thread, we can reliably show an error message instead of nothing at all
- ◉ Our backends process overall, ~6000 req/s less

My take aways.

My take aways.

- ◉ Varnish is pretty rad
- ◉ It's a lot of work to effectively use Varnish
- ◉ Optimize for anon, then layer on user information
- ◉ Understand both sides of Varnish
- ◉ Don't cache too much
- ◉ Be really really careful with user-specific caching!

WTB If-Modified-Since!

WTB If-Modified-Since!

- ◉ experimental-ims branch
- ◉ “200 Ok Not Modified”
- ◉ A ton of long tail data
- ◉ Very simple and efficient to serve a 304 Not Modified
- ◉ <https://www.varnish-cache.org/trac/wiki/BackendConditionalRequests>

Bad times were had.

Bad times were had.

- ◉ We tested the experimental-ims branch in production
- ◉ Very very short TTLs, with a really long keep
- ◉ Paired up with SSD file storage
- ◉ Varnish kept OOM'ing and crashing
- ◉ Had to keep restarting Varnish every 4-6 hours
- ◉ Really looking forward to trying this in Varnish 4.0

How we DDoS'd Fastly.

How we DDoS'd Fastly.

- ◉ *.disqus.com/count.js loader script
- ◉ *.disqus.com/count.js?**q=1**&... actually loads the payload
- ◉ Really bad idea. Really old legacy.
- ◉ Tried to optimize hit ratio, and ignored all querystrings
- ◉ Infinite redirect loop, in all browsers

How we DDoS'd Fastly.

```
diff --git a/shortname.disqus.com.vcl b/shortname.disqus.com.vcl
index 74b59e8..9a17bad 100644
--- a/shortname.disqus.com.vcl
+++ b/shortname.disqus.com.vcl
@@ -9,7 +9,10 @@ sub vcl_recv {
    }
}

- if (req.url ~ "^/count\.js") {
+ // This absolutely *has* to be an exact match
+ // Anything else will cause really really bad things
to happen
+ // like an infinite loop bringing down all the things
+ if (req.url == "/count.js") {
    if (req.http.Fastly-SSL) {
        error 751 "https";
    } else {
```

Thanks.

@mattrobenolt
github.com/mattrobenolt