Felix Gessert

Building a Shop with Sub-Second Page Loads
Lessons Learned

Hosting & Scalability
Presentation is loading
Why performance matters

- **100 ms**
- **400 ms**
- **500 ms**

Loading...

Average: 9.3s

- **0% Traffic**
- **0% Visitors**
- **0% Revenue**

Why performance matters: improving load times can significantly increase traffic, visitors, and revenue.
## What is the goal?

### Speed Perception

<table>
<thead>
<tr>
<th>Delay</th>
<th>Perception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 100 ms</td>
<td>Instant</td>
</tr>
<tr>
<td>100 – 300 ms</td>
<td>Small perceptible delays</td>
</tr>
<tr>
<td>300 – 1000 ms</td>
<td>Machine is working</td>
</tr>
<tr>
<td>1+ s</td>
<td>Mental context switch</td>
</tr>
<tr>
<td>10+ s</td>
<td>Task abandoned</td>
</tr>
</tbody>
</table>

Loading time < 1s

---

Concrete Example
A Scalable Webshop

Expectations:
• 2.7 Million viewers
• 300,000 Visitors in 30 minutes
• 20,000 Requests per second
• 4 weeks development and testing
Concrete Example
A Scalable Webshop

Goal:
Load time < 1s & 100% Availability

Expectations:
• 2.7 Million Viewers
• 300,000 Visitors in 30 minutes
• 20,000 Requests per second
• 4 weeks development and testing
If performance is so important for Business Success... what causes slow page load times?
State of the Art
The Three Bottlenecks

Network

Frontend

Backend
BOTTLENECK I
Optimizing Frontend Performance
Frontend Performance
The critical rendering path

```html
<!doctype html>
<title>Code Talks</title>
<link href=all.css rel=stylesheet />
<script src=app.css ></script>
<div>
  <h1>Web Performance</h1>
</div>

<script>
  elem.style.width = "50px";
  document.write("JS is awesome!");
</script>
```

div { color: green; }
h1 { padding: 10px; }
Inlining critical CSS and JS “above the fold”

Load CSS first, JS last

Load non-critical CSS and JS asynchronously

Compress images

Progressive rendering

Single-page application

Minify and concatenate CSS and JS

PageSpeed Insights

Test your performance

GTmetrix

Load non-critical CSS and JS asynchronously
Frontend Performance
Applied in the example

Load time: 767 ms
Size: 565.9 KB
Requests: 24

THINKS Tower+ Sporthandtuch - TV: DHDL | STRYVE

GET www.thinks.com 200 OK thinks.com 7.1 KB 26ms
GET 01.png 200 OK thinks.com 86.4 KB
GET founders.png 200 OK thinks.com 93.3 KB
GET jquery.min.js 200 OK ajax.googleapis.com 28.6 KB
GET app.js 200 OK thinks.com 86.6 KB
GET app.css 200 OK thinks.com 25.9 KB
GET sprite.png 200 OK thinks.com 34.3 KB
GET bg.png 200 OK thinks.com 103.6 KB
GET bg1.png 200 OK thinks.com 31.3 KB
GET zHzcR_WhtjSQC0oH996 200 OK fonts.gstatic.com 13.2 KB
GET IQHow_FEYlIDC4Gzy_n 200 OK fonts.gstatic.com 13.2 KB
GET connect 200 OK thinks.com 3.6 KB
GET analytics.js 200 OK google-analytics.com 11.3 KB
GET logo-stryve-inv.svg 200 OK thinks.com 939 B
GET logo-thinks-inv.svg 200 OK thinks.com 1.2 KB
GET 10.png 200 OK thinks.com 21.4 KB
GET alert_message 200 OK thinks.com 281 B
GET grau_anthrazit 200 OK thinks.com 511 B
GET merino_taupe 200 OK thinks.com 514 B
GET graphite_anthrazit 200 OK thinks.com 534 B
GET platinum_black 200 OK thinks.com 572 B
GET neonblue_black 200 OK thinks.com 504 B
GET neonred_black 200 OK thinks.com 563 B
GET neongreen_black 200 OK thinks.com 630 B

24 Requests 565.9 KB 693ms (onload: 767ms)
BOTTLENECK II

Tuning Network Performance
DNS Lookup
- Every domain has its own DNS lookup

Initial connection
- TCP makes a three-way handshake $\rightarrow$ 2 roundtrips (1 with TCP Fast Open)
- SSL connections have a more complex handshake $\rightarrow$ +2 roundtrips (only 1 with TLS False Start or Session Resumption)

Time to First Byte
- Depends heavily on the distance between client and the backend
- Includes the time the backend needs to render
  - Session lookups, Database Queries, ...

Content Download
- New connections are slow (initial congestion window is small) $\rightarrow$ many roundtrips

Network Performance
Break down of a single resource load

Maximum 6 parallel connections
Network

Bandwidth vs. Latency

Page Load Time as bandwidth increases

Page Load Time as latency decreases

Netzwerk
Bandbreite vs. Latenz

2× Bandwidth = Same Load Time

½ Latency ≈ ½ Load Time
Network Performance
Common Tuning Knobs

• **Persistent** connections
• Avoid **redirects**
• Set **caching headers** (no heuristic caching)
• **Content Delivery Networks**
  • Lower Latency
  • Cache images, CSS, JS
  • Terminate SSL early and optimized

• **Single Page Apps:**
  • Small initial page with JS
  • Loads data lazily

HTTP/2:
• **Multiplexing** over 1 TCP connection (no head-of-line blocking)
• Request **Pipelining**
• Server **Push**
• **Header Compression**
Why HTTP/2 Matters

Typical Improvement

HTTP

3.22s

HTTPS

4.03s

HTTP with CDN

0.44s

HTTPS with CDN and h2

0.35s
HTTP/1.1 vs. HTTP/2

524 ms vs. 268 ms
Why CDNs Matter
Why CDNs Matter

- Lower **latency** to client
- **Caching** on the edge
- DDOS protection

- **Failover** & Stale-on-error
- Scalable
- Warm **backend** connections
Network

Thinks

1. Avoid **redirects**, when necessary serve from CDN.
2. Heavy **browser and CDN caching**.
3. **Persistent** backend connections and **IP anycasting**.
4. **HTTP/2** with optimized **SSL and TCP**.
5. **Gzip** compression for text-based files.
6. **Minimize DNS lookups**.
Network Thinks

- Avoid redirects, when necessary serve from CDN
- Heavy browser and CDN caching
- Persistent backend connections and IP anycasting
- HTTP/2 with optimized SSL and TCP
- Gzip compression for text-based files
- Minimize DNS lookups

Minus 1 point:
Google Analytics TTL too short
Making the backend fast and scalable
Backend Performance
Overview

- Load Balancing
- Auto-scaling
- Failover

- Stateless Sessions
- Minimize shared state
- Efficient Code & IO

- Horizontally scalable databases (e.g. “NoSQL”)
  - Replication
  - Sharding
  - Failover
Shop Backend
On Baqend Cloud

Backend-as-a-Service API:
Data, Queries, User Login, etc.

Scalable Databases

Inclusion of all Web Caches
Access through HTTP

Shop Backend On Baqend Cloud

Desktop
Mobile
Tablet

Internet

Content-Delivery-Network

Caches

Baqend

redis
mongoDB
elasticsearch.
Shop Backend
On Baqend Cloud

CDN on fastly
Baqend Cloud on Amazon Web Services

On Baqend Cloud
Load Testing

6.8 M Requests
10K/s
Latency 5ms
Performance: State of the Art
Summarized

Frontend

Doable with **best practices** and support through **build tools**

Network

Caching and CDNs help, but only for **static content**

Backend

Many frameworks and platforms, but horizontal **scalability** is very difficult
How AMP works:

• Stripped down HTML + AMP tags (e.g. img) → rendered asynchronously by AMP runtime
• CSS must be inlined + <50 KB +
• No custom JS (except in iframes)
• Only static sizes → no repaints
• Cached in Google CDN, as long as it is crawled the next time → only suited for static media, e.g. news

How to apply these techniques for any website?

https://www.ampproject.org/docs/reference/spec.html
Static Content
- Images
- CSS, JS, fonts
→ easy to cache

Dynamic Data
- Content (e.g. product)
- JSON (e.g. stock counter)
- APIs & Queries
→ uncachable
Goal: Low-Latency for Dynamic Content
By Serving Data from Ubiquitous Web Caches
How does this work?
Dynamic Caching in Detail

Has **Time-to-Live** (expiration)

False-Positive Rate: \( f \approx \left(1 - \frac{\ln n}{m}\right)^k \)
Hash-Functions: \( k = \left\lfloor \ln(2) \cdot \left(\frac{n}{m}\right) \right\rfloor \)

With 20,000 entries and a 5% false positive rate: **11 Kbyte**

**Consistency:** Δ-Atomicity, Read-Your-Writes, Monotonic Reads, Monotonic Writes, Causal Consistency

\[ f \approx 1 - e^{-\frac{\ln n}{m}} \]

\[ k = \left\lfloor \ln(2) \cdot \left(\frac{n}{m}\right) \right\rfloor \]
How does this work?
Dynamic Caching in Detail

7 Years
Research & Development

Universität Hamburg


F. Gessert, W. Wingerath, S. Friedrich, and N. Ritter, „Scalable Data Management: NoSQL Data Stores in Research and Practice“, 32nd IEEE International Conference on Data Engineering, ICDE, 2016

From Objects to Queries

Query Caching

Create
Update
Delete

Server

Pub-Sub

Real-Time Queries (Websockets)

How to cache queries:
„Give me the most popular products that are in stock.“

Fresh Bloom filter

InvaliDB

Query Caching

How to cache queries:
„Give me the most popular products that are in stock.“

Real-Time Queries (Websockets)

InvaliDB

Fresh Bloom filter
Developing a Shop on Baqend

**Dashboard**
- Schema (e.g. products)
- Data (e.g. orders)
- Users

**CLI**
- Deploy shop frontend and backend code
- Run tests & staging

**REST & SDK**
- Load products
- Call checkout
E-COMMERCE

How to make legacy shops fast?
New Standard: Service Workers

What they do:

- **Proxy** any HTTP request
- **Offline Cache** for "progressive web apps"
Service Workers + Baqend Caching

Baqend Worker:

- **Redirect** requests to Baqend for faster delivery by including a **snippet**
- **Update** of cached data: refresh of stored data against origin

Public Beta in 3 weeks
commerce.codetalks.de
(Rails)

Try it: codetalks.baqend.com
(with offline mode)
In action: 
makefast.baqend.com
Typical Speedup: 15x

Impact of Global Caching
THINKS USE CASE

How do these techniques work for a high-traffic shop?
< 1 second Page Load Time

Simultaneous Users

7.8% Conversion Rate

3% Server Usage

Shops in "Die Höhle der Löwen"
The Google Page Speed Scores for Season 3, 09/06/2016
High Cache Hit Rate: 99%

Peak: >20,000 Requests per Second

Logo in TV-Screen: sudden spike

Microcaching extremely effective under high load

>3.2 Gigabit/s
Concurrent Users

91% Mobile Traffic

At the peak GA slowed down and started downsampling
Lessons Learned

**Frontend**
- **Simple** frontend means good performance
- **Single-page application**
- **Tooling** for optimizations (Inlining, Above-the-fold, Minification, etc.)

**Network**
- **Caching** in the CDN and browser
- **Dynamic data** should also be cached
- **Minimize latency**, SSL and HTTP tuning

**Backend**
- Horizontal **scaling** with stateless web servers
- **NoSQL** databases
- **Cloud**-hosted
- **Load-Tests** important
- **Failover** and **autoscaling**

fg@baqend.com  @baqendcom
Building a Shop with Sub-Second Page Loads: Lessons Learned

Here is the story of how we leveraged research on web-caching and NoSQL systems to prepare a webshop for hundreds of thousands of visitors due to TV Show publicity and everything that we learned along the way.
Recommendations
Literature and Tools

Good Ressources:

- https://developers.google.com/web/fundamentals/performance/?hl=en
- https://hpbn.co/

Good Tools:

- https://developers.google.com/speed/pagespeed/
- https://gtmetrix.com
- https://www.baqend.com/
- http://www.webpagetest.org/