UX/Frontend

Web Performance
Die effektivsten Techniken aus der Praxis

Erik Witt
Presentation is loading
Why performance matters

Average: 9.3s

Loading...

- 1%
- 100 ms
- 9%
- 500 ms
- 500 ms
- 20%
- 1s
- 7%
- 1s

Conversions
Traffic
Visitors
Revenue

Aberdeen Group
Google
Yahoo!
Amazon.com
What should be the goal?

<table>
<thead>
<tr>
<th>Delay</th>
<th>User Perception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 100 ms</td>
<td>Instant</td>
</tr>
<tr>
<td>100 – 300 ms</td>
<td>Small perceptible delay</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>Tasks is abandoned</td>
</tr>
</tbody>
</table>

Stay under 1000 ms to keep users attention

Concrete Example
A scalable webshop

Expectations:
• 2.7 Million Viewers
• 300,000 Visitors in 30 minutes
• 20,000 Requests per second
If performance is so important for Business Success...
...what causes slow page load times?
State of the Art
The tree bottlenecks
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>
```

Network ➔ HTML ➔ DOM ➔ JavaScript ➔ Execution ➔ CSS ➔ CSSOM ➔ DOM ➔ Render Tree ➔ Layout ➔ Paint
Frontend Performance
Render and parser blocking

What to do:
• **Inline** critical CSS and JS **above the fold**
• CSS at the **top**, JS at the **bottom**
• Load non-critical CSS and JS **asynchronously**
• Rendering the page **progressively**
• **Minify, concatenate, compress** and **cache** CSS, JS and images

Google Developers, Web Fundamentals
https://developers.google.com/web/fundamentals/performance/critical-rendering-path/analyzing-crp
Frontend Performance
Tools to improve your page load

Profiling

Minification & Compression

GTmetrix  WEBPAGETEST  PageSpeed Insights

Google Closure  tiny.png

Inlining & Optimization

Critical  PostCSS  processhtml

UglifyJs & cssmin
# Frontend Performance

**Applied in the example**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Size</th>
<th>Load time</th>
<th>Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.thinks.com">www.thinks.com</a></td>
<td>7.1 KB</td>
<td>26 ms</td>
<td></td>
</tr>
<tr>
<td>01.png</td>
<td>86.4 KB</td>
<td>20 ms</td>
<td></td>
</tr>
<tr>
<td>founders.png</td>
<td>93.3 KB</td>
<td>31 ms</td>
<td></td>
</tr>
<tr>
<td>jquery.min.js</td>
<td>8.6 KB</td>
<td>145 ms</td>
<td></td>
</tr>
<tr>
<td>app.js</td>
<td>86.6 KB</td>
<td>20 ms</td>
<td></td>
</tr>
<tr>
<td>app.css</td>
<td>25.9 KB</td>
<td>13 ms</td>
<td></td>
</tr>
<tr>
<td>sprite.png</td>
<td>34.3 KB</td>
<td>24 ms</td>
<td></td>
</tr>
<tr>
<td>bg.jpg</td>
<td>103.6 KB</td>
<td>68 ms</td>
<td></td>
</tr>
<tr>
<td>bg.jpg</td>
<td>31.3 KB</td>
<td>29 ms</td>
<td></td>
</tr>
<tr>
<td>zhxrr_WihjSSOQ0hJ9T.css</td>
<td>13.2 KB</td>
<td>276 ms</td>
<td></td>
</tr>
<tr>
<td>How_FYIDC4Gay_m.css</td>
<td>13.2 KB</td>
<td>281 ms</td>
<td></td>
</tr>
<tr>
<td>connect</td>
<td>3.6 KB</td>
<td>281 ms</td>
<td></td>
</tr>
<tr>
<td>analytics.js</td>
<td>11.3 KB</td>
<td>198 ms</td>
<td></td>
</tr>
<tr>
<td>logo-stryve-inv.svg</td>
<td>939 B</td>
<td>7 ms</td>
<td></td>
</tr>
<tr>
<td>logo-thinks-inv.svg</td>
<td>1.2 KB</td>
<td>7 ms</td>
<td></td>
</tr>
<tr>
<td>10.png</td>
<td>21.4 KB</td>
<td>7 ms</td>
<td></td>
</tr>
<tr>
<td>alert_message</td>
<td>281 B</td>
<td>7 ms</td>
<td></td>
</tr>
<tr>
<td>grau_anthracit</td>
<td>511 B</td>
<td>7 ms</td>
<td></td>
</tr>
<tr>
<td>marine_taupe</td>
<td>534 B</td>
<td>6 ms</td>
<td></td>
</tr>
<tr>
<td>graphite_anthracit</td>
<td>534 B</td>
<td>6 ms</td>
<td></td>
</tr>
<tr>
<td>platinum_black</td>
<td>572 B</td>
<td>6 ms</td>
<td></td>
</tr>
<tr>
<td>neonblue_black</td>
<td>504 B</td>
<td>6 ms</td>
<td></td>
</tr>
<tr>
<td>neonred_black</td>
<td>563 B</td>
<td>6 ms</td>
<td></td>
</tr>
<tr>
<td>neonred_black</td>
<td>563 B</td>
<td>6 ms</td>
<td></td>
</tr>
<tr>
<td>24 Requests</td>
<td>565.9 KB</td>
<td>693 ms (onload: 767 ms)</td>
<td>24</td>
</tr>
</tbody>
</table>
Network Performance

Break down of a single resource load

**DNS Lookup**
- Every domain has its own DNS lookup

**Initial connection**
- TCP makes a three way handshake → 2 roundtrips (1 with the new TCP Fast Open)
- SSL connections have a more complex handshake → +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 the page
  - Session lookups, Database Queries, Template rendering ...

**Content Download**
- Files have a high transfer time on new connections, since the initial congestion window is small → many roundtrips

Maximum 6 parallel connections
Network Performance
The average website

Total Transfer Size & Total Requests

Transfer Size: 2480 KB
Total Requests: 104
Network Performance
Network latency impact

Network Performance
Network latency impact

2× Bandwidth = Same Load Time

½ Latency ≈ ½ Load Time

---
Network Performance
Common Tuning Knobs

• **Persistent** connections, if possible **HTTP/2**
• Avoid **redirects**
• Explicit **caching headers** (no heuristic caching)
• **Content Delivery Networks**
  • To reduce the distance between client and server
  • To cache images, CSS, JS
  • To terminate SSL early and optimized

• **Single Page Apps:**
  • Small initial page that loads additional parts asynchronously
  • Cacheable HTML templates + load dynamic data
  • Only update sections of the page during navigation

HTTP/2 Performance
Advantages:
• Server **Push**
• **Header Compression**
• Request **Pipelining**
• **Multiplexing** over 1 TCP connection (no head-of-line blocking)
Network Performance
Applied in the example

What we do:
- Heavy browser and CDN Caching
- Avoid Redirects (+ serve from CDN)
- Single Page App functionality
- Persistent Backend Connections
- Gzip Compression
- IP Anycasting to nearest POP

![Google PageSpeed Insights](https://www.thinks.com/)

100 / 100 User Experience 99 / 100 99 / 100
Backend Performance

Overview

Load Balancer
- Load Balancing
- Auto-scaling
- Failover

Application Server
- Stateless session handling
- Minimize shared state
- Efficient Code & IO

Database
- Horizontally scalable databases (e.g. “NoSQL”)
  - Replication
  - Sharding
  - Failover
Backend Performance
Applied in the example

Backend-as-a-Service Middleware:
Caching, Transactions, Schemas, Invalidation Detection, …

Unified REST API
Standard HTTP Caching

Desktop
Mobile
Tablet

Content-Delivery-Network

BaQend

polyglot
Storage

Backend Performance
Applied in the example

Backend-as-a-Service Middleware:
Caching, Transactions, Schemas, Invalidation Detection, …

Unified REST API
Standard HTTP Caching

Desktop
Mobile
Tablet

Content-Delivery-Network

BaQend

polyglot
Storage

Standard HTTP Caching

Unified REST API

Backend Performance
Applied in the example

Backend-as-a-Service Middleware:
Caching, Transactions, Schemas, Invalidation Detection, …

Unified REST API
Standard HTTP Caching

Desktop
Mobile
Tablet

Content-Delivery-Network

BaQend

polyglot
Storage

Standard HTTP Caching

Unified REST API

Backend Performance
Applied in the example

Backend-as-a-Service Middleware:
Caching, Transactions, Schemas, Invalidation Detection, …

Unified REST API
Standard HTTP Caching

Desktop
Mobile
Tablet

Content-Delivery-Network

BaQend

polyglot
Storage

Standard HTTP Caching

Unified REST API

Backend Performance
Applied in the example

Backend-as-a-Service Middleware:
Caching, Transactions, Schemas, Invalidation Detection, …

Unified REST API
Standard HTTP Caching

Desktop
Mobile
Tablet

Content-Delivery-Network

BaQend

polyglot
Storage

Standard HTTP Caching

Unified REST API

Backend Performance
Applied in the example

Backend-as-a-Service Middleware:
Caching, Transactions, Schemas, Invalidation Detection, …

Unified REST API
Standard HTTP Caching

Desktop
Mobile
Tablet

Content-Delivery-Network

BaQend

polyglot
Storage

Standard HTTP Caching

Unified REST API

Backend Performance
Applied in the example

Backend-as-a-Service Middleware:
Caching, Transactions, Schemas, Invalidation Detection, …

Unified REST API
Standard HTTP Caching

Desktop
Mobile
Tablet

Content-Delivery-Network

BaQend

polyglot
Storage

Standard HTTP Caching

Unified REST API
Performance: State of the Art
Summarized

**Frontend**
- Doable with the right set of best practices
- Good support through build tools

**Latency**
- Caching and CDNs help, but a considerable effort and only for static content

**Backend**
- Many frameworks and platforms
- Horizontal scalability is very difficult
Performance: State of the Art
Summarized

Good Resources:

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

Good Tools:

https://developers.google.com/speed/pagespeed/

https://gtmetrix.com

http://www.webpagetest.org/
Accelerated Mobile Pages
Google’s Approach for a Faster Web

How AMP works:
• Stripped down HTML + AMP tags → rendered asynchronously by AMP runtime
• All CSS must be inlined
• All JS must be async
• Static sizes (e.g. iframes) → no repaints
• Cached in Google CDN, as long as it is crawled the next time (no invalidation) → only suited for static media, e.g. news

How to apply the same techniques for any website?

https://www.ampproject.org/docs/reference/spec.html
Goal: Low-Latency for Dynamic Content

By Serving Data from Ubiquitous Web Caches
Innovation

Problem:

6 Years Research & Development
New Algorithms Solve Consistency Problem

Stale Data

Innovation Problem: changes cause stale data
Innovation
Solution: Baqend proactively revalidates data

6 Years
Research & Development

α
New Algorithms
Solve Consistency Problem

Bloom filter
Is still fresh?
0 1 1 1 1 1 0 1
update

Years Research & Development
New Algorithms Solve Consistency Problem
Innovation
Solution: Baqend proactively revalidates data

6 Years


W. Wingerath, F. Gessert, S. Friedrich, „Real-time stream processing for Big Data“, Big Data Analytics it - Information Technology, 2016


New Algorithms


F. Gessert, „Skalierbare NoSQL- und Cloud-Datenbanken in Forschung und Praxis“, BTW 2015

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

Faster Page Loads

- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency
Continuous Query Matching
Generalizing the Cache Sketch to query results

Main challenge: when to invalidate?
- **Objects**: for every update and delete
- **Queries**: as soon as the query result changes

How to detect query **result** changes in real-time?
Add, Change, Remove all entail an invalidation and addition to the cache sketch.
InvaliDB

Architecture

Create
Update
Delete

BAQEND

Pub-Sub

Continuous Queries (Websockets)

Polyglot Views

Fresh Caches

Fresh Cache Sketch

Scalable ACID Transactions

• Solution: Conflict-Avoidant Optimistic Transactions
  • Cache Sketch fetched with transaction begin
  • Cached reads → Shorter transaction duration → less aborts
**Scalable ACID Transcations**

- **Novelty:** ACID transactions on sharded DBs like MongoDB

- **Current Work:** DESY and dCache building a scalable namespace for their file system on this

![Graph showing performance comparison with and without caching](graph.png)
The World's Fastest Backend

Build websites and apps that load instantly.

Sky-rocket your Development

Start building now. Baqend Cloud is free and easy to get started with.
Parse: 2.6 s
Baqend: 0.3 s
Kinvey: 3.9 s
What impact does Baqend have in practice?
Summary

HTTP Caching + Cache Sketch + Invali-dations + TTL Estimation + RT Query Matching

Frontend
< 1 second Page Load Time

7.8% Conversion Rate

Simultaneous Users

3% Server Usage

Shops in "Die Höhle der Löwen"

The Google Page Speed Scores for Season 3, 09/06/2016
Our Vision for Baqend:
„A web without load times“

www.baqend.com
@Baqendcom