Optimierung im Web

Connor Gäde

10.03.2017

Zusammenfassung

Das Internet nimmt einen immer größeren Teil in unserer Gesellschaft ein. Immer mehr Dienste werden auf das Web ausgelagert, immer mehr Technologien internetfähig. Umso wichtiger ist es, die Reaktionszeiten von Webapplikationen zu optimieren. Hierfür müssen die benötigten Ressourcen richtig an das Web angepasst werden, damit auch Nutzer mit niedriger Internetleistung diese optimal Nutzen können. Zusätzlich bilden Echtzeitapplikationen eine weitere Herausforderung, da das zugrundeliegende HTTP Protokoll ursprünglich nicht für diese Aufgabe konzipiert wurde. Bei großen Webseiten besteht außerdem das Problem, die Serverlast optimal auf die einzelnen Server zu verteilen, um Ausfälle zu verhindern. Hierfür werden Load Balancer verwendet. Diese Arbeit bietet einen Überblick über die verschiedenen Verfahren, um diese Aufgaben zu bewältigen.

Inhaltsverzeichnis

1 Einführung

Das Internet ist eine der wohl größten Errungenschaften der modernen Zivilisation. Viele Millionen Menschen sind heute miteinander vernetzt und immer mehr Tätigkeiten lassen sich über das Internet verrichten. Ein Großteil aller Unternehmen ist dort vertreten. Insbesondere durch die Verbreitung von Smartphones ist der Zugang quasi überall und jederzeit möglich.

Diese Verbreitung hat allerdings zur Folge, dass es für eine Webseite wichtig ist, aus der Masse an Konkurrenten hervorzustechen. Neben dem Aussehen ist dabei insbesondere die Reaktionszeit der Seite ein wichtiges Kriterium. Gerade das mobile Internet ist in vielen Gebieten noch sehr langsam, was das Optimieren der Zugriffs- und Reaktionszeit umso wichtiger macht.

1.1 Aufbau einer Webseite

Eine Webseite wird durch die Hypertext Markup Language (HTML) beschrieben. Dabei wird die Seite in einen Baum aus den einzelnen Elementen der Seite strukturiert. Ein Element wird dabei durch ein Starttag $< a >$ zusätzlichen und ein Endtag $< \backslash a >$ beschrieben. Zudem kann das Starttag zusätzliche Attribute enthalten, das Endtag besteht teilweise nur implizit. Das HTML Dokument hat einen Kopfteil (head) und einen Rumpf (body). Der Kopf enthält Meta-Informationen zur Seite, der Rumpf den Inhalt, der tatsächlich dargestellt wird. (Siehe Listing 1) [9]

Listing 1: Eine simple HTML Datei

```
<!DOCTYPE html>
<html>
  <head>
    <title>Beispiel</title>
    <link rel="stylesheet" href="beispiel.css"
     type="text/css" />
  </head>
  <body>
    <p>Beispieltext</p>
    <img src="beispiel.jpg" alt="Beispielbild">
    <script type="text/javascript"
     src="beispiel.js"></script>
  </body>
</html>
```

Dieses Dokument wird vom Browser in eine Baumstruktur, den sogenannten Document Object Model (DOM) tree, übersetzt. Dieser kann daraufhin von

Skript Dateien (normalerweise Javascript) manipuliert werden, um die Seite dynamisch zu verändern. Die Elemente des Baumes werden daraufhin vom Browser dargestellt. Dieser Darstellungsprozess kann durch Stylingsprachen wie CSS beeinflusst werden. [9]

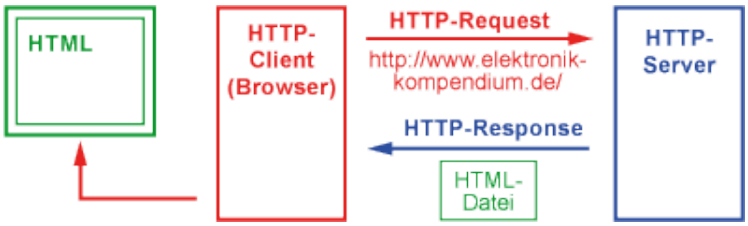

Abbildung 1: Client-Server Kommunikation bei Aufruf einer Webseite[16]

Ruft nun ein Nutzer eine Internetadresse auf, wird mithilfe des Hypertext Transfer Protokoll (HTTP) eine Anfrage an den Server gestellt. Dieser sendet daraufhin als Antwort die dazugehörige HTML Datei zurück, welche der Browser danach wie zuvor beschrieben darstellt.[16] Zudem stellt er weitere Anfragen für die restlichen Elemente, welche zur Darstellung benötigt werden, wie Bilder, Stylesheets und Skripte. [4, 16] Seit HTTP/2 können diese zusätzlichen Elemente auch vor einer weiteren Anfrage durch den Browser eigenständig vom Server gesendet werden.[7]

2 Optimierung von Ressourcen

Um die Performance einer Webseite zu maximieren, müssen die einzelnen Ressourcen richtig für das Web angepasst werden. Dies umfasst die richtige Anordnung von Komponenten, das Minimieren von Dateigrößen und die Reduzierung von deren Anzahl. Im Folgenden wird auf jede dieser Maßnahmen näher eingegangen.

Das richtige Anordnen von Ressourcen im HTML Dokument kann bereits die gefühlte Performance der Webseite erhöhen. Stylesheets sollten im Dokumentenkopf platziert werden, da der Browser so während des Interpretierens des Dokumentes die Webseite mit allen bereits vorhandenen Elementen rendert. Befinden sich Style-Anweisungen im Dokumentenrumpf, so muss der Browser die gesamte Seite neu rendern, sobald er diese interpretiert. Einige Browser blockieren sogar den Renderprozess, bis alle Style-Anweisungen interpretiert worden sind.[4, 13]

Skript-Dateien sollten dagegen möglichst weit am Ende des HTML Dokumentes angeordnet werden. Wird ein Skript heruntergeladen, so wird der

Renderprozess blockiert, bis das Skript ausgeführt wurde, da es den Inhalt der Webseite verändern könnte. Zudem sollten sowohl Skript, als auch Style Anweisungen in eigene Dateien ausgelagert werden, damit sie im Cashe gespeichert werden können und so bei zukünftigen Aufrufen nicht neu geladen werden müssen. [4, 13]

In HTTP/1.1 kann eine tatsächliche Performance Steigerung durch das verteilen von Ressourcen auf mehrere Domains erreicht werden. Dies ermöglicht es dem Browser mehr Ressourcen gleichzeitig herunterzuladen. Die zusätzlichen DNS Lookups erzeugen jedoch ebenfalls Verzögerungen, daher sollten nicht zu viele verschiedene Domains verwendet werden. In HTTP/2 sollte diese Praxis vermieden werden, da durch Multiplexing ohnehin mehrere Ressourcen parallel über eine Verbindung heruntergeladen werden können und so nur der Nachteil durch die DNS lookups besteht.[4, 12]

2.1 Reduzierung von Dateigrößen

Das Reduzieren von Dateigrößen kann die initiale Ladezeit einer Webseite signifikant beeinflussen. Bei erstmaligem Betreten einer Seite muss jede Datei heruntergeladen werden bevor sie für zukünftige Besuche im Cache gespeichert wird. Gerade für Nutzer mit niedriger Internetgeschwindigkeit, insbesondere von Mobilgeräten, ist es daher wichtig, diesen Prozess zu minimieren.

Der einfachste Schritt ist es, Dateien zu komprimieren. Hierfür muss lediglich die Komprimierung mit Gzip oder Deflate in den Servereinstellungen aktiviert werden. Des weiteren können in Style- und Skript-Dateien sämtliche Kommentare und Leerstellen entfernt werden, sowie möglichst kurze Variablennamen benutzt werden, um die Dateigröße zu minimieren. Hierfür gibt es diverse Online-Tools. [4, 12]

Bilder lassen sich auf multiple Weise minimieren. Sie sollten nicht in einer größeren Auflösung als benötigt auf dem Server hinterlegt werden. Das Runterskalieren mithilfe von HTML Anweisungen ändert nicht die ursprüngliche Dateigröße. Zudem sollten nur die Formate gif, png und jpeg genutzt werden, wobei png meistens gif vorzuziehen ist. Von jpeg lassen sich die Metadaten entfernen, um diese weiter zu minimieren. Bei gif Dateien kann zudem die Anzahl der Farbkanäle angepasst werden und sollte möglichst minimiert werden. Des weiteren gibt es diverse Optimierungstools, um Bilddateien zu optimieren.[4, 5]

2.2 Reduzierung von HTTP Anfragen

Die Reduzierung von HTTP Anfragen ist ein großer Bestandteil der Optimierung von HTTP/1.1 Webseiten. Dadurch, dass nur einzelne Dateien über eine limitierte Anzahl Verbindungen heruntergeladen werden kann, wird durch das zusammenfassen von Dateien die Ladezeit optimiert. CSS und Skriptdateien können für alle Seiten zusammengefasst werden. Ebenso können Bilder entweder zu Imagemaps zusammengefasst werden, in welchen durch das deklarieren von Bereichen verschiedene Verlinkungen eingebunden werden können, oder zu Spritesheets aus welchen mit Hilfe von CSS die einzelnen Teilbilder ausgewählt und einzeln dargestellt werden können.[4, 12]

Ein weiterer Weg, Dateien Zusammenzufassen ist das Inlining. Hierfür können Dateien in das 'data' url Schema konvertiert werden, welches eine Textrepräsentation der Datei darstellt und somit in die HTML oder CSS Datei integriert werden kann. (Siehe Abbildung 2) Der resultierende Text ist jedoch größer als die eigentliche Datei, weshalb abgewägt werden muss, ob der das Einsparen der HTTP Anfrage das Anwachsen der Dateigröße ausgleicht. [4, 12, 15]

```
<img width="18" height="32" src="data:image/gif;
base64,R0lGODlhEgAgAMQAAOYpMaQYIMUgKQAgeyBz7kqs/
4vV///NMf+0GOasava9e//VpNViEP////b29lJSUkFBQSkpK
SAgIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAACwAAAAAEgAgAAAF7WAjjkNpjig5GCxbDCnZukUNp
0Ot7wVxkz1CQiKpEY4/0eAoFDiOEiTuuFA4E4RFNKkkKL7fh
ELC7RLDWF9MKUm43RBIZM0mJhBxeezxOEAOEgh8cQwpfAd9A
Ih8CAgSKAsAfweUlY4PEI8NVQkBfxIGiw8Sg49hnoCKfHykc
Y9uIoB9q6OlKbKifAAAAQEofhJ/tBBVCgm/mXGLccXHsXJEE
n55EMbODQcRwcHTeRERhdjf23F4cd/gIgjj2nLr6Okj6xLa4
+8M4Sj40UQQCPlOJOwaRQdFNFcFRxyEQDChiGAIHUKMmJCfJ
hQhAAA7" />
```

Abbildung 2: Beispiel eines 'data' url Schema mit dazugehöriger Grafik (Konvertiert mit [10])

In HTTP/2 sollten diese Zusammenfassungen allerdings vermieden werden, da nun parallel mehrere Dateien heruntergeladen werden können und der Server auch ohne vorherige Anfrage bereits Ressourcen an den Nutzer übertragen werden können. Somit hat das Zusammenfassen hauptsächlich den negativen Effekt, dass die wenigen großen Dateien im Cache schon

durch kleinere Änderungen ungültig werden und neu heruntergeladen werden müssen. Daher ist es für die Performanz einer HTTP/2 Webseite eher von Vorteil, Dateien mit häufigen Änderungen, von welchen mit wenigen zu trennen. Dateien die durch Inlining eingefügt wurden, können zudem nicht von der Priorisierung des Browsers beachtet werden, welche es dem Browser erlaubt Dateitypen in gewünschter Reihenfolge herunterzuladen und ebenfalls durch HTTP/2 eingeführt wurde. [12] Dabei sollte allerdings bedacht werden, dass Stand 10.03.2017 ca. 20% der globalen Internetnutzer kein HTTP/2 unterstützen. [1]

3 Bidirektionale Kommunikation

Das HTTP ist ein request/response Protokoll, das heißt der Client sendet Anfragen auf welche der Server antwortet. Ein HTTP/1.1 Server kann weder selbstständig eine Verbindung aufbauen, noch ohne vorhergehende Anfrage eine Antwort senden. [14] HTTP/2 erlaubt es dem Server, auch Antworten ohne dazugehörige Anfrage zu senden, sofern zuvor bereits eine andere Anfrage des Clients eingetroffen ist.[7] Allerdings befinden sich Bibliotheken, die diese Technologie nutzen, noch in der Entwicklung.[6] Asynchrone Ereignisse können somit nicht durch den Server mitgeteilt werden, was die Realisierung von Echtzeitanwendungen erschwert.

3.1 Polling

Das Polling ist die einfachste Methode, um asynchrone Ereignisse zu erhalten. Dabei wird lediglich in regelmäßigen Abständen eine Anfrage an den Server gesendet. Dies ist allerdings für Echtzeitanwendungen ungeeignet, da neue Änderungen durch das Aktualisierungsintervall nur verzögert sichtbar sind. Zudem werden überflüssige Anfragen gestellt, wenn gar keine Änderungen vorhanden sind. Des weiteren wird die Anzahl der HTTP Anfragen an den Server vervielfacht. Dafür ist dieses Verfahren HTTP/2 kompatibel und kann auch bei Serverpools mit Lastverteilern oder Proxies problemlos genutzt werden, da keine langlebige Verbindung besteht. [6, 14]

3.2 Long Polling

Das Long Polling minimiert die Zeit, bis eine neue Änderung beim Client eintrifft, indem eine Anfrage solange offen gehalten wird, bis ein neues Ereignis eintrifft. Dieses wird daraufhin als Antwort an den Client gegeben, woraufhin eine neue Anfrage gestellt wird. Dieses Verfahren stellt im Vergleich

zum Polling keine überflüssigen Anfragen und die Ereignisse sind Zeitnah sichtbar. Zudem bestehen die selben Kompatibilitätsvorteile das Pollings. Allerdings werden noch immer bei jedem Ereignis eine neue HTTP Anfrage gestellt wird. Je nach Frequenz der Ereignisse kann dies den Server stärker belasten, als das Polling. [6, 14]

3.3 WebSocket Protokoll

Das WebSocket Protokoll wurde entwickelt, um Browser-Applikationen einen Mechanismus bereitzustellen, ohne das wiederholte Erstellen von HTTP Anfragen beidseitige Kommunikation aufbauen zu können. Hierfür sendet der Client eine HTTP Upgrade Anfrage an den Server, um eine feste, bidirektionale TCP Verbindung zwischen beiden Parteien aufzubauen. Über diese kann der Server nun selbstständig Änderungen mitteilen ohne neue Verbindungen aufbauen zu müssen. [6, 14] Inzwischen werden WebSockets von über 90% der Nutzer global unterstützt. [2] Einige ältere Browser sind jedoch nicht Kompatibel, daher müsste für vollständige Erreichbarkeit eine zweite Methode als Ersatz genutzt werden. Des weiteren ist das Protokoll nicht vollständig mit HTTP/2 kompatibel. Im Gegensatz zu den vorher erwähnten Methoden werden bei mehreren geöffneten Tabs auch mehrere Verbindungen erzeugt. Zudem ergeben sich durch die permanente Verbindung Probleme mit Proxis und beim Load Balancing. [6, 11, 14]

4 Lastverteilung

Die in den vorherigen Abschnitten behandelten Optimierungsmaßnahmen beschränkten sich bisher auf das Minimieren der Serverlast durch einzelne Nutzer. Bei Webseiten mit hohen Zugriffszahlen sorgt allerdings bereits die große Anzahl an Nutzern für eine hohe Serverlast. Um bei Ausfall eines einzelnen Servers einen Totalausfall der Webseite zu vermeiden, bietet es sich an, die Serverlast auf mehrere Server zu verteilen. Dies hat den weiteren Vorteil, dass zur Erweiterung der Rechenkapazität neue Server dazugeschaltet werden können, ohne den alten ersetzen zu müssen.[3, 17]

Da allerdings eine Domain immer nur einem physikalischen Server zugewiesen werden kann, braucht man hierfür einen sogenannten Load Balancer, welcher die Anfragen auf die einzelnen Server verteilt.[17] Dabei sollte gewährleistet werden können, dass ein Anwender auch immer mit dem selben Server verbunden bleibt. Wenn Informationen im Browser oder im Servercache zwischengespeichert werden, kann es sonst zu Performance-Verlusten

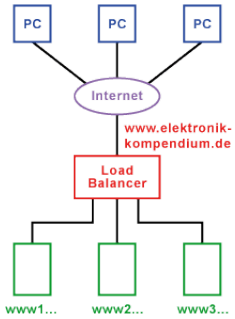

Abbildung 3: Anbindung mehrerer Server mit Loadbalancer[17]

oder Transaktionsfehlern kommen. [3, 18] Des weiteren sollte der Load Balancer möglichst effizient die Anfragen verteilen und sicherstellen, dass nur Server angewählt werden, welche online sind.[3]

Es gibt Hardware und Software basierte Load Balancer. Welche Variante besser geeignet ist, ist abhängig von der Art der Anwendung und dem Unternehmen.[8, 18] Hardware Load Balancer benutzen meist spezialisierte Prozessoren, bei größerer Last muss das Gerät gegen ein teureres ausgetauscht werden.[3, 8] Software Load Balancer sind dagegen meist flexibler und können auf beliebiger Hardware installiert werden.[3] Sie besitzen allerdings nicht alle Funktionen eines Hardware Load Balancers. Zudem können Software basierte Load Balancer sensitiv gegenüber dem Betriebssystem oder der Virtuellen Umgebung sein.[8]

4.1 DNS-Verfahren

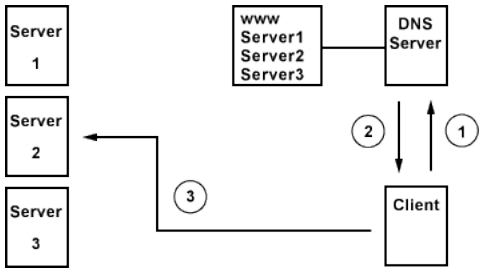

Abbildung 4: DNS Server verteilt abwechselnd IP Adressen [18]

Eine einfache Form der Lastverteilung ist das Eintragen mehrerer eigenständiger Server auf dem zur eigenen Domain gehörenden DNS-Server.

Es werden nun abwechselnd Anfragen auf einen der Server weitergeleitet. Sobald sich ein Nutzer einmal mit einem der Server verbunden hat, wird er diesen auch in Zukunft für die Verbindung mit der Domain nutzen, da die zugehörige IP-Adresse des Servers vom Gerät des Nutzers lokal gespeichert wird. Dadurch wird auch die Persistenz der Session gewährleistet. Allerdings wird die individuelle Serverlast nicht belastet, sodass im ungünstigsten Fall alle Nutzer auf den gleichen Server zugreifen. Zudem benötigt jeder Server eine eigene, im Internet erreichbare IP.[18]

4.2 Hard- und Software basierte Verfahren

Nutzt man einen tatsächlichen Load Balancer, wird dessen IP-Adresse, wie bereits zuvor erwähnt, für die Domain der Webseite genutzt.[17] Die Server selbst müssen daher zum Internet keine eigene IP-Adresse besitzen. Um sicherzustellen, dass ein Nutzer immer mit dem selben Server kommuniziert, speichert hier der Load Balancer die IP des jeweiligen Nutzers und leitet diesen entsprechend weiter. [3, 18]

4.3 Round-Robin Verfahren

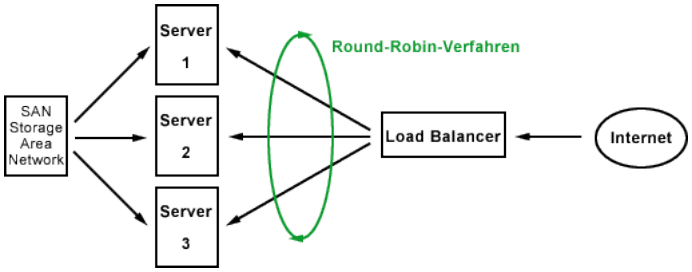

Abbildung 5: Abwechselnde Zuweisung durch Round-Robin-Verfahren[18]

Beim Round-Robin Verfahren verteilt der Load Balancer Anfragen nacheinander auf die einzelnen Server.[3, 18] Dies entspricht im Prinzip dem DNS-Verfahren, nur das hier der Load Balancer die Persistenz der Session übernimmt. Die tatsächliche Serverlast wird jedoch weiterhin nicht beachtet.[18]

4.4 Feedback-basierte Verfahren

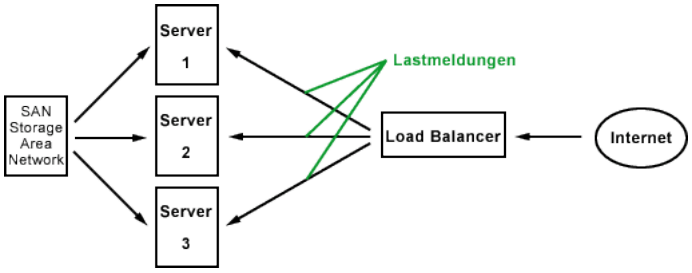

Abbildung 6: Lastverteilung durch Serverfeedback [18]

Feedback-basierte Verfahren verteilen die Anfragen, wie der Name schon sagt, anhand von Feedback durch die einzelnen Server. Durch die Informationen zur tatsächlichen Serverauslastung kann der Load Balancer die Nutzer anhand von einer internen Rangliste an den am wenigsten belasteten Server weiterleiten.[3, 18] Hierfür müssen allerdings neben der Anzahl der aktiven Verbindungen auch die jeweiligen Rechenkapazitäten der Server beachtet werden.[3] Da der Serverstatus dem Load Balancer bekannt ist, kann so auch verhindert werden, dass Anfragen an Server weitergeleitet werden, welche offline sind. Für das Feedback muss zuvor eine Kommunikation zwischen Server und Load Balancer realisiert werden. Dies erhöht den Konfigurationsaufwand.[3]

4.5 URL-basiertes Verfahren

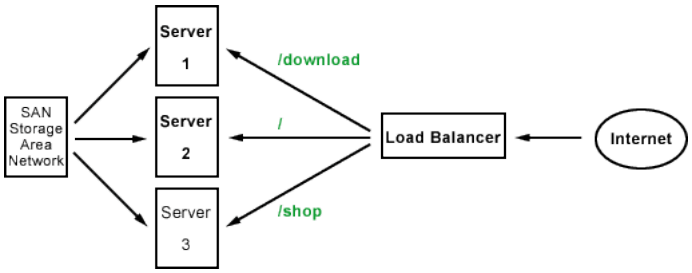

Abbildung 7: Verteilung spezifischer URL Anfragen auf verschiedene Server [18]

Ein weiteres Verfahren, um Serverlast zu verteilen, ist es, die Anfragen je nach URL an bestimmte Server weiterzuleiten. Die Verzeichnisse werden

auf verschiedene Rechner verteilt. Hierfür muss allerdings eine vorhergehende Analyse der Zugriffszahlen erfolgen, um zu ermitteln, welche Seiten wie oft aufgerufen wird. Diese Analyse sollte regelmäßig wiederholt werden. Es ist zudem spezielle oder besonders schnelle Hardware notwendig, um den Datenstrom sämtlicher Anfragen nach Verzeichnissen zu filtern.[18]

4.6 Dienst-basiertes Verfahren

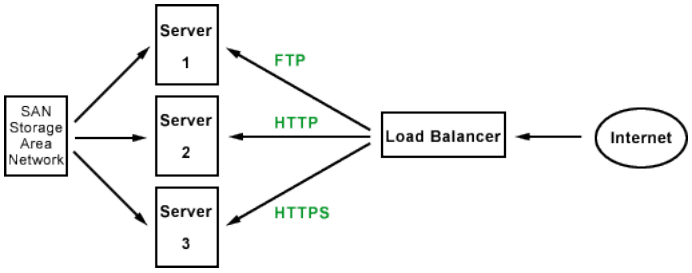

Abbildung 8: Verteilung verschiedener Dienste auf verschiedene Server [18]

Bietet man verschiedene Dienste an, so kann man die Serveranfragen auch anhand der gewünschten Dienste auf dafür dedizierte Server verteilen lassen. Die Identifikation des gewünschten Dienstes ist dabei einfacher als die der URL, da jeder Dienst auf einen spezifischen Port zugreift. Auch hier muss der Datenverkehr analysiert werden, um die Rechenleistung effektiv aufzuteilen.[18]

5 Fazit

Zusammengefasst sollten für das Optimieren einer Webseite die Dateien möglichst stark komprimiert werden, um Download-Zeiten zu verringern. Des weiteren sollte auf die korrekte Anordnung der Dateien geachtet werden, um auch die gefühlte Ladezeit zu beschleunigen. Stylesheets sollten an den Dokumentenkopf gesetzt werden, Skriptdateien möglichst weit an das Dokumentenende. Das Zusammenfassen von Dateien sollte nur für Kompatibilität mit älteren Browsern praktiziert werden, ebenso wie das Verteilen der Ressourcen auf verschiedene Hostnamen. Im Hinblick auf HTTP/2 sollten Dateien eher anhand ihrer Langlebigkeit getrennt werden, um den Cache optimal zu nutzen. Asynchrone Ereignisse lassen sich mit verschiedenen Methoden. Wenn keine große Genauigkeit gewünscht ist, kann regelmäßiges

Polling die beste Lösung sein. Ist Echtzeit relevant, sollte im Hinblick auf Effizienz das WebSocket Protokoll verwendet werden, allerdings ist Long Polling kompatibler. Bei großer Serverlast kann ein Load Balancer die Anfragen auf mehrere Server aufteilen. Auch hier gibt es verschiedene Methoden, welche meist kombiniert werden. Im Hinblick auf die Zukunft lässt sich sagen, dass die Optimierung im Web stark im Wandel ist. HTTP/2 ist noch relativ jung und hat die Weboptimierung bereits stark verändert. Insbesondere die eingebauten Server Push Mechanismen könnten die Echtzeitapplikationen in Zukunft maßgeblich beeinflussen.

Literatur

[1] Can i use http/2? `http://caniuse.com/#search=http%2F2` (letzter Zugriff 10.03.2017).

[2] Can i use websockets? `http://caniuse.com/#search=websockets` (letzter Zugriff 10.03.2017).

[3] What is load balancing? `https://www.nginx.com/resources/glossary/load-balancing/` (letzter Zugriff 10.03.2017).

[4] Best practices for speeding up your web site, 2006. `https://developer.yahoo.com/performance/rules.html` (letzter Zugriff 10.03.2017).

[5] Hovhannes Avoyan. 30 tips to optimize html/css/images for smooth web experience, 2011. `http://www.monitis.com/blog/30-tips-to-optimize-htmlcssimages-for-smooth-web-experience/` (letzter Zugriff 10.03.2017).

[6] Paul Banks. The state of real-time web in 2016, 2016. `https://banksco.de/p/state-of-realtime-web-2016.html` (letzter Zugriff 10.03.2017).

[7] M. Belshe, R. Peon, and M. Thomson. Hypertext transfer protocol version 2 (http/2). RFC 7540, RFC Editor, May 2015. `http://www.rfc-editor.org/rfc/rfc7540.txt`.

[8] Stephen J. Bigelow. Hardware vs. software load balancer: Which is better for an enterprise?, 2016. `http://searchitoperations.techtarget.com/answer/`

Hardware-vs-software-load-balancer-Which-is-better-for-an-enterprise (letzter Zugriff 10.03.2017).

[9] Alex Danilo, Arron Eicholz, Steve Faulkner, and Travis Leithead. HTML 5.1. W3C recommendation, W3C, November 2016. https://www.w3.org/TR/2016/REC-html51-20161101/.

[10] Mike Fosket. Image to data uri converter, 2016. https://websemantics.uk/tools/image-to-data-uri-converter/ (letzter Zugriff 10.03.2017).

[11] Wolfram Hempel. Load balancing websocket connections, 2016. https://deepstream.io/blog/load-balancing-websocket-connections/ (letzter Zugriff 10.03.2017).

[12] Ryan Hodson. Http/2 for web developers, 2015. https://blog.cloudflare.com/http-2-for-web-developers/ (letzter Zugriff 10.03.2017).

[13] Brian Jackson. Perceived web performance – what is blocking the dom?, 2017. https://www.keycdn.com/blog/blocking-the-dom/ (letzter Zugriff 10.03.2017).

[14] S. Loreto, P. Saint-Andre, S. Salsano, and G. Wilkins. Known issues and best practices for the use of long polling and streaming in bidirectional http. RFC 6202, RFC Editor, April 2011. http://www.rfc-editor.org/rfc/rfc6202.txt.

[15] Larry Masinter. The 'data' url scheme. RFC 2397, RFC Editor, August 1998. http://www.rfc-editor.org/rfc/rfc2397.txt.

[16] Patrick Schnabel. Http - hypertext transfer protocol. http://www.elektronik-kompendium.de/sites/net/0902231.htm (letzter Zugriff 10.03.2017).

[17] Patrick Schnabel. Load balancer (lastverteiler). http://www.elektronik-kompendium.de/sites/net/0904131.htm (letzter Zugriff 10.03.2017).

[18] Patrick Schnabel. Load balancing, 2006. http://www.elektronik-kompendium.de/sites/net/0906201.htm (letzter Zugriff 10.03.2017).