1 Einführung in die Performance-Optimierung

Mit der fortschreitenden Entwicklung hin zu immer schnelleren Internetanschlüssen und leistungsfähigeren Endgeräten steigt die Erwartungshaltung der Nutzer an Anwendungen und Internetpräsenzen. Sie fordern kurze Ladezeiten, flüssige Bedienung und ständige Verfügbarkeit. Gerade in Zeiten von Twitter, Facebook, Google+, YouTube und ähnlichen Diensten steigt der Bedarf an der schnellen Verbreitung von Inhalten. Neben dem Angebot und der Qualität der Dienste spielt zunehmend die Lade- und Darstellungszeit und damit die Performance einer Webseite eine wichtige Rolle – nicht nur um die Erwartungen der Endnutzer zu erfüllen, sondern auch um das Angebot unter hohen Lastbedingungen in seiner Qualität zu wahren. Damit ist die Performance einer Webseite heutzutage das ausschlaggebendste Kriterium und wirkt sich dadurch maßgeblich auf den Umsatz eines Angebots aus.

In den vergangenen Jahren hat sich jedoch die durchschnittliche Ladezeit einer Webseite trotz schnellerer Internetanschlüsse kaum verbessert. Gründe hierfür liegen unter anderem an der nahezu linear zur Verbindungsgeschwindigkeit gestiegene Datenmenge, die für die immer reichhaltigeren Webseiten übertragen werden muss. Dies stellt vor allem für die stetig wachsende Anzahl an mobilen Internetzugängen und Telekommunikationsprovider ein Problem dar.

1.1 Definition

Der Begriff »Optimierung« bedeutet die Verbesserung eines Merkmals, eines Materials oder eines Produktes. Die Optimierung von Performance im Kontext dieses Buches zielt auf die Verbesserung der Leistungsfähigkeit einer Webanwendung im Hinblick auf die Nutzung der Systemressourcen und der Geschwindigkeit sowie der »User Experience«[1] ab. Eine Optimierung kann allerdings erst dann begonnen werden, wenn operationalisier-

[1] Die *User Experience* oder auch das *Nutzererlebnis* beschreibt hier die Erfahrungen mit einem Softwareprodukt oder einer Webanwendung.

bare (messbare) Kriterien für die Software existieren. Es muss definiert werden, auf welchen Teil der Anwendung sich die Optimierung konzentriert und wie die Performance im Kontext der Anwendung definiert ist. Das heißt, es muss definiert werden, was schnell und langsam bedeutet und wie »schnell« und »langsam« gemessen werden können. Das Bestimmen, das Messen und die Auswertung der Kriterien erfolgt in Performance-Tests, die iterativ durchgeführt werden. Jeder Performance-Test basiert somit auf den Erkenntnissen des vorangegangenen Tests und untersucht entweder den gleichen, optimierten Bereich oder konzentriert sich auf einen komplett neuen Bereich der Software.

1.2 Vielseitigkeit in Client-Server-Umgebungen

Endanwendern steht in der heutigen Zeit eine Vielzahl an Geräten für die Nutzung von Internetdiensten zur Verfügung. Sind die meisten Webseitenbenutzer vor ein paar Jahren hauptsächlich noch über Desktop-PCs oder Laptops von zu Hause aus ins Internet gegangen, hat sich der Trend dank neuer Geräte in Richtung der mobilen Nutzung durch Smartphones und Tablets gewendet. Durch den verstärkten Ausbau der mobilen Netze ist das Surfen über Smartphones und Tablets inzwischen schneller und angenehmer geworden. Die Verbindungsqualität ist in der Regel jedoch trotzdem durch die Mobilfunknetze und deren Auslastung beschränkt. Zudem sind die Datentarife der Endanwender und deren Verbindungsgeschwindigkeiten ein weiterer Geschwindigkeitsfaktor. Deshalb bieten Webseitenanbieter immer häufiger optimierte Inhalte für mobile Geräte an, um die Benutzer nicht durch große und langsame Webseiten zu verärgern und zu verlieren. Gerade für Webshops spielt dieser Faktor eine nicht zu unterschätzende Rolle.

Festzustellen ist allerdings auch, dass Webanwendungen trotz der steigenden Leistung der stationären und auch mobilen Internetanschlüsse im Schnitt nicht schneller laden: Webangebote werden durch hochauflösende Grafiken, Multimediaelemente, ausgeprägte Animationen und Interaktionen immer größer in ihrer Datenmenge und benötigen mehr Rechenleistung bei ihrer Darstellung. Die Ladezeit einer Webseite verringert sich in der Regel nicht proportional mit der steigenden Verbindungsgeschwindigkeit der Internetanschlüsse.

Für Dienste zwischen Servern (z.B. Datenaustausch via APIs) verhält sich dieses Problem ähnlich: Die Ausführungszeit der Anwendung auf den Server sollte möglichst gering sein, die Bandbreite muss eine Vielzahl von Anfragen verarbeiten können und die Latenz zum Angebot sollte gering sein, um einen hohen Durchsatz erreichen zu können.

1.3 Mythen

Im Bereich von Webanwendungen sind über die Jahre diverse Mythen und Trugschlüsse entstanden. Einige werden von Entwickler zu Entwickler weitergegeben oder stehen als vermeindliche Empfehlungen in Blogs oder Foren, ohne vom Autor valide überprüft worden zu sein. Einige dieser Mythen werden an dieser Stelle vorgestellt.

»Cachen bis der Arzt kommt«

Ist eine Anwendung langsam oder kommt sie bei Lastspitzen schnell an ihre Grenzen, kann Caching Abhilfe schaffen: Mehrfach aufgerufene und verwendete Inhalte werden hierbei in einem schnellen Speichermedium auf Abruf gehalten und müssen somit nicht jedes Mal neu generiert werden.

Für einige Entwickler ist deshalb Caching das »Mittel« gegen jegliche Arten von Performance-Problemen, wodurch die Aussage »Cachen bis der Arzt kommt« in diesem Fall zurecht entstanden ist. Caching ist in der Tat ein eher einfaches Verfahren, um die Leistung eines Systems zu erhöhen und Zugriffe auf »teure« Ressourcen zu reduzieren. Jedoch sollten zuerst die Anwendung selbst und die verwendeten Algorithmen auf ihre Performance hin untersucht und optimiert werden, bevor Caching aktiviert wird. Die Notwendigkeit eines Cache zeigt nur, dass ein Bereich der Anwendung inperformant ist, wie z.B. Festplattenzugriffe. Grundlegendes Fazit ist, dass Caching nicht überall sinnvoll ist und es daher nur bedacht eingesetzt werden sollte. Statt blind auf Caching als Lösung aller Performance-Probleme zu vertrauen, sollte Caching wohlüberlegt durch die Erstellung eines Caching-Konzeptes eingesetzt werden. Weitere Denkanstöße zu diesen Ansätzen werden in Kapitel 5 vorgestellt.

»Synchrone Verarbeitung ist langsam«

Gerade durch die seit mehreren Jahren in Webseiten verbreitete asynchronen Aufrufe durch Ajax-Anfragen hat die synchrone Verarbeitung einen immer schlechteren Ruf bekommen. Asynchrone Verarbeitung bietet im Vergleich zur synchronen Verarbeitung große Geschwindigkeitsvorteile, z.B. können dadurch Teilbereiche einer Webseite dann nachgeladen werden, wenn sie benötigt werden. Dadurch kann Bandbreite gespart und die Ladezeit deutlich verbessert werden. Clientseitig bietet die asynchrone Verarbeitung u.a. den Nachteil, dass vermehrt HTTP-Anfragen auftreten können. Weitere Nachteile können je nach Anwendungsfall in der Barrierefreiheit oder bei der Suchmaschinenoptimierung entstehen.

Serverseitig kann die Realisierung von asynchroner Verarbeitung einen großen Performance-Gewinn zeigen. Die Konzeption und die Implementierung ist allerdings aufwendiger als die clientseitige Realisierung, da

meist zusätzliche Hardware und geeignete Programmiersprachen verwendet werden müssen. Weiterhin ist serverseitiges Queuing nicht bei allen Anwendungsszenarien verwendbar. Gerade wenn Abhängigkeiten zwischen Berechnungsergebnissen bestehen, muss asynchrone Verarbeitung nicht schneller sein als die synchrone Verarbeitung.

»Lasttests brauchen doch nur große Anwendungen«

Oft entsteht der Trugschluss, dass Performance- und Lasttests nur für wirklich große Webanwendungen nötig sind, die viele Tausend Besucher bewältigen müssen. Als große Beispiele fallen meist die Namen der »Success Stories« wie »Facebook«, »Twitter« oder »StudiVZ«. Allerdings haben auch diese großen Unternehmen mit anfangs kleinen Webanwendungen begonnen und zum Zeitpunkt des Start-Ups keine oder wenige Tests durchgeführt. Mit zunehmender Bekanntheit und steigender Last auf den Systemen haben diese Tests aufgrund von vorausgegangenen Ausfällen immer mehr an Bedeutung gewonnen. Dies zeigt sich auch an dem Beitrag von Tools, wie beispielsweise dem im August 2012 veröffentlichten Lasttest Framework »Iago« von Twitter und der Weiterentwicklung von Technologien wie »Memcached« oder dem PHP-zu-C-Compiler »HipHop for PHP» durch Facebook. Aus der Historie dieser Unternehmen kann entnommen werden, dass frühzeitiges Testen sinnvoll ist, da es dank bereits vorhandener Tools nur einen geringen Mehraufwand bedeutet und die daraus gelernten Konzepte zur guten Qualität der Anwendung beitragen.

»JavaScript ist langsam«

JavaScript ist unter vielen Entwicklern als unsaubere, unübersichtliche und langsame Programmiersprache verrufen, da Programmcode bei jedem Aufruf erneut interpretiert und verarbeitet werden muss. In den ersten Generationen der Webbrowser diente JavaScript hauptsächlich nur für die Validierung von Formularen und für die Verarbeitung und Manipulation von einfachen Inhalten. Komplexe Operationen haben diese Browser durchaus an ihre Leistungsgrenzen gebracht, woraus der bleibende Eindruck der langsamen Verarbeitung und der Ausdruck »JavaScript ist langsam« entstanden sind.

Aktuelle Webbrowser verfügen hingegen über hoch optimierte JavaScript-Engines, die teilweise mehrfach ausgeführte Codefragmente direkt in Maschinencode übersetzen, um die Ausführungszeit zu verkürzen. Dadurch sind in modernen Browsern durchaus komplexe und umfangreiche Anwendungen wie Spiele, Textverarbeitung etc. mithilfe von JavaScript möglich. Die Aussage »JavaScript ist langsam« trifft daher heutzutage nicht mehr zu. Weitere Details zur Optimierung von JavaScript finden sich in Abschnitt 8.4. Die damit verbundenen allgemeinen Optimierungsmöglich-

keiten bei der Darstellung auf dem Client sind ebenfalls in Kapitel 8 dargestellt.

»Ist das Internet heutzutage nicht schon schnell genug?«

Obwohl die Internetverbindungen in den letzten Jahren um ein Vielfaches schneller geworden sind, hat sich die Ladezeit einer Webseite im Durchschnitt kaum verbessert, da sich fast linear zu der verfügbaren Bandbreite auch die Datenmenge der Webseiten vergrößert hat. Während in den ersten Jahren des Internets durch Modems mit geringen Bandbreiten zwischen 9,6 bis 56 kBit/s noch sehr auf die Reduzierung der Datenmenge geachtet wurde, haben viele Entwickler dies aufgrund der zur Verfügung stehenden Ressourcen aus den Augen verloren (siehe Abbildung 1.1). Zudem sorgt der Konkurrenzdruck zwischen Unternehmen ebenfalls dafür, dass Webseiten immer ausgefallener und komplexer werden, was ein größeres Dateivolumen zur Folge hat.

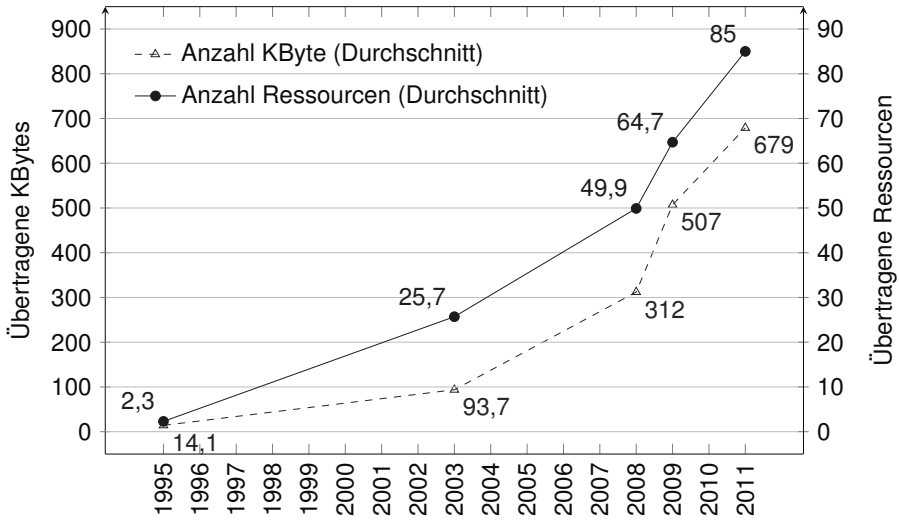

Abbildung 1.1: Entwicklung der Gesamtdatengröße und Anzahl der extern eingebundenen Ressourcen der Alexa-Top-1000-Webseiten von 1995 bis 2011 [Cha10, DPSG07, FB08, Sou11]

1.4 Gründe für die Performance-Optimierung

1.4.1 Verfügbarkeit des Webangebots

In der Vergangenheit hatten große Plattformen wie z.B. Twitter, Facebook oder MySpace immer wieder mit Performance-Problemen zu kämpfen, da diese zu Beginn ihres Markteintritts nicht mit einem so rasanten Wachstum gerechnet hatten. Auf der Plattform Twitter traten beispielsweise in der Phase steigender Popularität in den Jahren 2008 und 2009 Ausfälle bei Lastspitzen auf. Spätere Nachbesserungen verursachten hohe Aufwände und zogen eine lange Umbauzeit nach sich.

Dies ist nicht nur bei den großen Erfolgsgeschichten zu beobachten, sondern zieht sich vielmehr durch fast alle Webseiten und Angebote im Internet. Webseiten können online sowie offline durch eine Erwähnung in einem Artikel, einem Gewinnspiel oder durch ein Angebot, das auf einer der großen Plattformen empfohlen wird, für einen kurzen Moment große Aufmerksamkeit auf sich ziehen. Greifen daraufhin viele Besucher auf die Webseite zu, kann beobachtet werden, dass die Webseite meist nur noch sehr langsam reagiert oder gar nicht mehr erreichbar ist.

Beispielsweise verlinkte ein Musiker mit rund 130.000 Fans auf seiner Facebook-Fanseite ein Gewinnspiel eines Musik-Magazins, das eine Reise zu einem Auftritt des Künstlers in Las Vegas verloste. Dies führte aufgrund der Facebook-Kommentare innerhalb von 7 Minuten zur völligen Überlastung des gesamten Onlineangebotes des Magazins und zog einen mehrstündigen Ausfall mit der Fehlermeldung »Cannot establish Database Connection« nach sich. Da der Webseitenbetreiber offensichtlich nicht auf diesen Ansturm interessierter Facebook-Nutzer eingestellt war, blieb die Webseite lange Zeit nicht erreichbar. Ob der Betreiber den Ausfall erst Stunden später bemerkte oder ob dieser nicht in der Lage war, eine schnelle Optimierung durchzuführen, blieb dabei jedoch unklar. Das Beispiel zeigt, dass heute jeder ohne umfangreiche Computerkenntnisse in der Lage ist, mithilfe von Facebook, Twitter oder ähnlichen Diensten einen sogenannten »Distributed Denial of Service«-Angriff – kurz *DDOS*[2] – durchzuführen. Dies kann wie im Beispiel auch unwissend und ohne böswillige Absichten passieren.

Neben Onlineangeboten zum Lasttesten von Webseiten gibt es auch kostenfreie Software wie Apache JMeter, mit der schon mit geringen Kenntnissen viele Anfragen (engl. *Requests*) pro Sekunde auf ein Webangebot ausgeführt werden können (siehe Abschnitt 3.4.4). Die Verbindung von diesen Tools mit schnell und günstig verfügbaren Ressourcen durch Cloud-

[2] »Distributed Denial of Service«-Angriff meint den Versuch, eine Ressource gezielt durch viele Anfragen lahmzulegen. Dabei werden die Angriffe von vielen verschiedenen Computern gleichzeitig ausgeführt.

Services, wie z.B. Amazon EC2, bringt weitere Möglichkeiten, um Lasttests auf ein Angebot durchzuführen, kann aber gleichermaßen für böswillige Angriffe verwendet werden.

Die Notwendigkeit einer hohen Verfügbarkeit eines Webangebots ist stark vom angebotenen Inhalt abhängig. Eine private Webseite oder ein Blog haben meistens nicht den Anspruch einer 99,9-prozentigen Verfügbarkeit. Ein Webshop, eine Unternehmensseite oder eine (Web-)Anwendung können jedoch, je nach Angebot, eine höhere Verfügbarkeit voraussetzen.

Mit entsprechender Kenntnis in Kernbereichen der Software kann eine Webanwendung gegen größere Ausfälle sowohl im privaten Bereich als auch in Unternehmen vorbereitet und weitestgehend abgesichert werden. Allerdings ist neben dem Ansporn der Entwickler der wirtschaftliche Faktor ausschlaggebend für die Verfügbarkeit des Angebots. Je größer der wirtschaftliche Faktor einer Anwendung ist, umso wichtiger ist eine hohe Verfügbarkeit dieser Anwendung. Es ist zu beachten, dass es bei einem Ausfall durch zu hohe Last oft schwierig ist, ohne entsprechendes Vorwissen oder Vorarbeit das Angebot wieder in kürzester Zeit verfügbar zu machen. Im schlimmsten Fall müssen Teile der Software gänzlich überarbeitet werden, was viel Zeit beanspruchen kann.

1.4.2 Wirtschaftlicher Faktor

Um sich gegen einen großen Ansturm von Nutzern abzusichern, sind prinzipiell Kosten und Nutzen abzuwägen. Eine Anwendung oder ein Service, der nur ein- oder zweimal pro Jahr eine Lastspitze erfährt, muss nicht zwingend auch diese Lastspitzen abdecken können. Falls jedoch ein größerer Anteil des Jahresumsatzes während dieser Spitzen generiert wird (z.B. während der Weihnachtszeit), ist es essenziell notwendig, sich auf diese Spitzen vorzubereiten. Unterschieden werden muss jedoch zwischen völlig zufälligen und vorhersagbaren Lastspitzen, die z.B. durch Feiertage oder ähnliche Ereignisse hervorgerufen werden. So kann die im vorigen Beispiel genannte Gewinnspielaktion des Künstlers zu unerwarteten Lastspitzen für ein Musikmagazin führen, wohingegen z.B. ein Fotobuchhersteller genau vorhersagen kann, dass vor besonderen Feiertagen wie Muttertag, Vatertag, Ostern oder Weihnachten der Großteil der Jahresbestellungen innerhalb kürzester Zeit eingeht. Auf vorhersagbare Lastspitzen kann sich ein Anbieter einfacher wappnen als gegenüber nicht vorhersagbaren Ereignissen.

Neben der Vorhersagbarkeit muss ferner zwischen gewinnbringenden und nicht gewinnbringenden Anwendungen unterschieden werden. Eine Anwendung oder Webseite, die keine wirtschaftlichen Ziele verfolgt, wird auch weniger Mittel und weniger Interesse an der systematischen Behandlung für Lastspitzen haben. Hingegen hat der Fotobuchhersteller aus dem

obigen Beispiel ein sehr hohes wirtschaftliches Interesse daran, dass die Server während des Zeitraums, in dem der Großteil der jährlichen Bestellungen eingeht, erreichbar sind und stabil laufen.

1.4.3 Kostenersparnis

Ein weiterer wirtschaftlicher Faktor, der oft nicht in der Kosten- und Nutzenrechnung beachtet wird, sind die Einsparungen von Betriebskosten. Neben den direkten Kosten für die Anschaffung von Hardware müssen ferner die Stromkosten sowohl die Kosten für den Unterhalt der Hardware und deren Wartung als auch für den Datentransfer mit einbezogen werden. Eine hohe Verfügbarkeit einer Anwendung ist aus Sicht der Software gegeben, wenn diese pro Anfrage wenig Hardwareressourcen verbraucht. Eine auf Performance optimierte Anwendung verbraucht weniger Ressourcen, wodurch mehr Anfragen bei gleicher Hardware verarbeitet werden können und dadurch weniger Hardware bereitgestellt werden muss. Durch den gesunkenen Bedarf an Hardware sinkt der Wartungsaufwand und dadurch auch die Fehleranfälligkeit.

Die folgenden Beispiele zeigen deutlich, dass Kosten bereits durch den Einsatz verbesserter Softwarekomponenten oder optimierter Einstellungen reduziert werden können:

- Der Internet-Streaming-Anbieter *Netflix* hat 2008 durch die Aktivierung der Datenkompression auf den Servern bis zu 50 % des Datentransfervolumens zum Client einsparen können [Sco08]. Dadurch hat sich der Datenverkehr des gesamten Angebots um die Hälfte reduziert. Zusätzlich wurde dadurch als Nebeneffekt die User Experience um 13 % – 25 % verbessert. Ein Großteil der Einsparungen und Verbesserungen wurde durch die Veränderungen von Konfigurationseinstellungen und den Einsatz einer einfachen Architektur erreicht.
- Die Suchergebnisse der Suchmaschine *Bing* wurden testweise künstlich um eine unterschiedliche Sekundenzahl verzögert. Daraufhin sank bei einer Verzögerung von 2 Sekunden der durchschnittliche Erlös pro Benutzer um 4,3 % und die Zufriedenheit der Benutzer um 3,8 % [SB09].

Zu erwähnen ist ebenso die Verbesserung bei der Betrachtung unter ökologischen Gesichtspunkten: Eine optimierte Anwendung bedeutet häufig weniger Hardware, weniger Datentransfer und auch weniger Ressourcenbedarf auf dem Client, was sich mit insgesamt weniger Energiebedarf bemerkbar macht. Allerdings können zusätzliche Mechanismen, die zur Performance-Optimierung beitragen, auch zusätzliche Ressourcen beanspruchen, wenn dafür separate Hardware benötigt wird. Dies ist zum Bei-

spiel der Fall, wenn Queuing-Services oder Caching-Server eingesetzt werden. Weiterhin ist neben den Kosten für Hardware und Energie auch der höhere Entwicklungsaufwand, der durch die steigende Komplexität der Systeme entstehen kann, zu beachten.

1.4.4 Entwicklungsaufwand

Die Performance einer Webanwendung bzw. deren Optimierung ist eine nicht funktionale Anforderung, die mitunter in vielen Projekten zu spät als eine wichtige Anforderung erkannt und definiert wird. In vielen Unternehmen entstehen Anwendungen jedoch meist geprägt von wirtschaftlichen Faktoren. Neuentwicklungen werden oftmals durch die Anforderung einer kurzen »Time to Market«[3] bestimmt. Weiterentwicklungen sind häufig von Wünschen des Kunden geprägt, weshalb die Optimierung oft aufgrund der Kosten und der Zeit nicht durchgeführt werden kann. Bugfixes müssen ebenso, je nach Schwere, schnell ausgerollt werden. Generell bleibt in wenigen Unternehmen Zeit für die Optimierung bestehender Anwendungen und es wird auch kaum Zeit für die Performance-Optimierung im Projektplan berücksichtigt. Änderungen im Hinblick auf die Performance werden oft erst vorgenommen, wenn es bereits zu einem kompletten Ausfall gekommen ist oder Engpässe die Nutzung deutlich einschränken.

Dabei müssen Performance-Optimierungen nicht unbedingt zum Ende des Projekts, sondern können bereits während des Projekts durchgeführt werden. Ein erfahrener Entwickler kann bereits in der Entwicklungsphase Performance-Schwachstellen entdecken, diese gegebenenfalls markieren oder gleich verbesserten Programmcode integrieren. Durch entsprechende Ergänzungen im Programmcode bei der Entwicklung ist es sogar sehr einfach möglich, Performance-Flaschenhälse (engl. »Bottlenecks«) frühzeitig zu identifizieren. Verbesserter Programmcode reduziert insgesamt den Nachbearbeitungsaufwand und sorgt bereits frühzeitig im Projektverlauf für eine hohe Performance der Anwendung und damit für eine hohe Kundenzufriedenheit und geringere Kosten.

1.4.5 Kundenzufriedenheit

Beim Besuch einer Webseite steht die Kundenzufriedenheit für die Erreichung eines Ziels (Information, Kauf etc.) im Vordergrund. Dies kann unter anderem durch ein ansprechendes Design der Seite erreicht werden. Viele Webseitenbetreiber vergessen jedoch häufig, dass die vom Nutzer wahrgenommene Ladezeit einer Webseite auch zur Kundenzufriedenheit beiträgt.

[3] Unter »Time to Market« ist der Druck einer schnellen Markteinführung eines Produkts zu verstehen.

Eine schnelle Auslieferung der Inhalte ist deshalb substanziell. Die Ladezeit einer Webseite wird dabei von vielen Faktoren beeinflusst:

- Die **Größe der Dateien**, darunter HTML-, Bild-, Stylesheet-, JavaScript-Dateien, beeinflusst die Zeit, die der Browser des Besuchers benötigt, um diese zu laden und zu verarbeiten. Durch die hohe Verbreitung von Breitbandanschlüssen in Deutschland wird der Faktor häufig nicht mehr beachtet, obwohl dieser weiterhin ausschlaggebend ist.
- Durch die **zunehmende Verwendung von multimedialen Inhalten**, Animationen und Effekten sowie die Ladezeit von nachgeladenen Inhalten durch JavaScript kann die Anwendung oder Webseite sich langsam anfühlen, sofern die Verarbeitung und Darstellung viel Rechenzeit benötigt.
- Ist eine Webseite schnell geladen, kann trotzdem eine **ungünstige Benutzerführung** oder Webseitenstruktur dazu beitragen, dass der Benutzer schnell das Interesse an der Webseite verliert. Die Benutzerführung (engl. »Usability«) einer Webseite ist deshalb ebenfalls essenziell für die Zufriedenheit des Webseitenbesuchers.

Diese Faktoren können durch Änderungen, wie zum Beispiel die Reduzierung und Minimierung der ausgelieferten Daten, optimiert werden. Zudem kann eine verbesserte Benutzerführung auf der Webseite dafür sorgen, dass der Benutzer schneller an sein Ziel kommt und eine Aktion, z.B. der Kauf in einem Webshop, schneller durchgeführt werden kann. Dadurch ist der Benutzer insgesamt zufriedener und die Anwendung erzeugt dadurch weniger Last, da der Benutzer ggf. weniger Seiten durchlaufen muss, um sein Ziel zu erreichen. Wie sich solche Verbesserungen einer Webseite in der Praxis auf die Kundenzufriedenheit auswirken, zeigen die folgenden Beispiele:

- *Mozilla* konnte durch Optimierungen die Start- und Downloadseite für den Webbrowser Firefox um 2,2 Sekunden schneller ausliefern, wodurch Firefox im Jahr 2010 60 Millionen Mal heruntergeladen wurde gegenüber 8 Millionen Downloads im Jahr 2008. Diese Verbesserung wurde durch die Optimierung von eingebundenen externen Dateien erreicht [Cut10].
- *Amazon* hat das Verhältnis zwischen Ladezeit, Kundenzufriedenheit und Verkäufen näher untersucht. Dazu wurde die Auslieferung der Amazon-Webseite künstlich um 100 ms verzögert. Das Resultat war, dass der Umsatz sich um 1 % verringert hat. Bei einem Jahresumsatz von 17,43 Milliarden US-Dollar im vierten Quartal 2011 sind dies beeindruckende 174 Millionen US-Dollar [Lin06].
- *Google* hat bei einer Untersuchung der Benutzerzufriedenheit in Abhängigkeit der Lade- und Darstellungsgeschwindigkeit von Web-

seiten einem Teil der Suchmaschinenbenutzer die Suchergebnisse mit einer Verzögerung von 500 ms ausgeliefert. Bei diesen Benutzern ging die Zahl der weiteren Suchanfragen um 20 % zurück, was auf eine steigende Unzufriedenheit der Nutzer schließen lässt [May08].

Die Beispiele zeigen, dass zufriedene Kunden eine Webseite häufiger besuchen, weiterempfehlen und länger auf dieser verweilen. Eine höhere Kundenzufriedenheit kann zu einer größeren Anzahl an Downloads oder einem gesteigerten Verkauf von Produkten führen. Zu bedenken ist auch, dass eine Webseite das Online-Aushängeschild eines Unternehmens darstellt. Ist die Webseite eines Onlineshops sehr träge und nur mühsam zu bedienen, kann dies damit einen schlechten Einfluss auf das Image des Unternehmens haben. Der Kunde könnte vermuten, dass die Bearbeitung und Lieferung seiner Bestellung ebenso langwierig ablaufen könnte. Als Nebeneffekt kann eine beschleunigte Auslieferung der Daten die Last auf den Servern reduzieren. Dies führt wiederum zur Einsparung von Ressourcen, Energie und den damit verbundenen Kosten. Eine interne Studie bei Google kam 2010 zu ähnlichen Ergebnissen:

> *Faster sites create happy users and we've seen in our internal studies that when a site responds slowly, visitors spend less time there. But faster sites don't just improve user experience; recent data shows that improving site speed also reduces operating costs.*
> *— Amit Singhal, Matt Cutts [SC10]*

1.4.6 Suchmaschinen-Ranking

Das Geschäftsmodell von Suchmaschinenanbietern besteht darin, dem Benutzer auf seine Anfrage hin möglichst viel Werbung anzuzeigen. Um möglichst viele Benutzer zu erreichen, müssen wertvolle und relevante Inhalte präsentiert werden, damit die Benutzer häufig wiederkehren. Dadurch legen Betreiber wie Google großen Wert auf die Geschwindigkeit des Suchvorgangs. Da es den Nutzer frustriert, wenn nach einem schnellen Suchvorgang die gefundenen Webseiten langsam geladen werden, bewertet Google seit April 2010 neben vielen anderen Faktoren auch die Performance der Webseite und lässt diese mit in ihren Ranking-Algorithmus einfließen. Das bedeutet letztendlich, je schneller eine Webseite lädt, desto höher ist die Wahrscheinlickeit, das Ranking in den Suchergebnissen von Google zu verbessern [SC10]. Da die Ranking-Algorithmen der Suchmaschinen jedoch gut gehütete Geheimnisse sind, lässt sich über die tatsächliche Gewichtung des Performance-Faktors nur spekulieren. Im Zuge einer Suchmaschinenoptimierung sollte eine geringe Ladezeit des Angebots allerdings immer auch angestrebt werden.

1.4.7 Kapazitätsplanung

Bei der Kapazitätsplanung geht es um die Planung der Last, die ein System oder eine Anwendung verarbeiten muss und sie zielt auf die Entscheidung ab, welche Architektur geeignet ist, um diese Last zu verarbeiten. Ausgangspunkt der Kapazitätsplanung ist die Bestimmung der Last durch Lasttests, wodurch gezeigt wird, wie viel Last die aktuelle Architektur imstande ist zu verarbeiten. Die Performance-Optimierung betrifft somit immer auch die Kapazitätsplanung, da durch die Tests die Belastbarkeitsgrenze der Anwendung ermittelt werden kann, um darauf aufbauend eine Abschätzung über die notwendige Hardware treffen zu können.

Bei der Kapazitätsplanung passiert es häufig, dass die zukünftige Kapazität zu hoch oder zu niedrig geschätzt wird. Eine zu hohe Schätzung bedeutet, dass unter Umständen mehr Hardware als benötigt eingesetzt wird. Wird die Kapazität zu niedrig eingeschätzt, reicht die Hardware womöglich nicht aus. Ein Ziel der Kapazitätsplanung ist demzufolge, die Hardwareausstattung eines Systems weder zu groß noch zu gering zu dimensionieren. Die Optimierung der Performance beeinflusst die künftige Leistungsfähigkeit der Anwendung und damit die entstehenden Kosten und Hardwareanforderungen.

1.5 Performance-Bereiche

Performance-Optimierungen werden inkrementell durchgeführt und sind immer auf einen bestimmten Bereich der Software fokussiert. Diese verschiedenen Bereiche werden im Folgenden als »Performance-Bereiche« bezeichnet, da diese die Performance der Software beeinflussen. Für jeden Bereich müssen unterschiedliche Kriterien festgelegt werden, die Aussagen über die Performance treffen können. Performance-Bereiche sind nicht immer voneinander abgegrenzt, sondern können auch voneinander abhängen und zusammenspielen. Ein Beispiel ist die Nutzung der Hardware, die Konfiguration der Hardware und die damit verbundene Performance abhängig vom Betriebssystem, auf dem die Software ausgeführt wird. In Abbildung 1.2 ist ein Überblick über die verschiedenen Performance-Bereiche und deren Zusammenhänge dargestellt.

1.5.1 Hardware

Die unterste Ebene im Performance-Bereich, auf der alle weiteren Komponenten aufsetzen, ist die Hardware. Software ist immer auf die Hardware abgestimmt, daher findet sich auf dem Großteil der Software im Handel Hardwareanforderungen. In den meisten Fällen werden dem

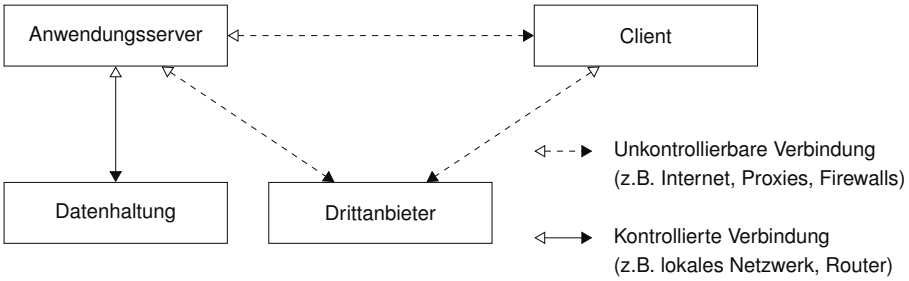

Abbildung 1.2: Überblick der Performance-Bereiche und ihrer Zusammenhänge

Anwender oder Betreiber von Software bezüglich der Hardware Vorschläge gemacht. Diese Vorschläge können Speicher (RAM), Prozessor (CPU), Festplatte (HDD) und Speicherkapazität sowie die Netzwerk-/Internetgeschwindigkeit (z.B. 10 Mbit/s oder DSL 16.000) umfassen. Bei Webanwendungen im Anwenderbereich (zum Beispiel *Wordpress*, *TYPO3*, *Magento*) werden üblicherweise keine expliziten Hardwareanforderungen gestellt, allerdings existieren auch für Webanwendungen Mindestanforderungen. Die Mindestanforderungen richten sich aber nach den Zielen, die die Anwendung erreichen soll. Beispielsweise ist die Installation des Shopsystems *Magento* auf schwacher Hardware mit wenig RAM und wenig Prozessorleistung zwar grundlegend funktionstüchtig, jedoch wird der Seitenaufbau langsam und mit wahrnehmbarer Verzögerung stattfinden. Eine gute »User Experience« wird dieser Shop einem Besucher damit nicht bieten können.

Generell sollte auch darauf geachtet werden, systemrelevante und ausfallsichere Hardware nicht aus gleichen Chargen zu verbauen. Bei Ausfällen in RAID-Systemen mit Festplatten aus der gleichen Charge kommt es des Öfteren vor, dass die Festplatten zum gleichen Zeitpunkt ausfallen. Gründe hierfür sind der Einsatz von ähnlichen Drittkomponenten und gleichen Ausgangsmaterialien.

Die Hardware ist ein grundlegender Performance-Bereich, weshalb dieser im Hinblick auf Performance-Tests und -Optimierung in der Software als konstant betrachtet werden sollte. Durch die Veränderung der Hardware während der Softwareoptimierung können keine Aussagen über die Steigerung der Performance in der Software getroffen werden. Weiterhin sollte aus Kostengründen immer vermieden werden, Softwareprobleme durch mehr Hardware zu beheben. Grundlegend gilt für Tests, dass immer weniger starke Hardware eingesetzt werden sollte, da oftmals dadurch Bottlenecks deutlicher und vor allem schneller erkennbar sind und letztendlich die Optimierungen an der Software größere Auswirkung zeigen.

1.5.2 Netzwerk

Ein Netzwerk ist als Zusammenschluss von verschiedenen Komponenten zu verstehen. Im Serverumfeld bezieht sich das auf die Vernetzung von Servern über Hochgeschwindigkeitsnetzwerke (100 Mbit/s, 1 Gbit/s, 10 Gbit/s), wohingegen im Client-Server-Umfeld damit meist die Internetanbindung gemeint ist, deren Übertragungsgeschwindigkeit von Anwender zu Anwender stark variiert (1 Mbit/s DSL 1.000, ..., 50 Mbit/s VDSL 50.000 und mehr). Server in Rechenzentren sind dabei untereinander mit leistungsfähigen Anschlüssen vernetzt, die üblicherweise über mehrere redundante Anbindungen über verschiedene weitere Anbieter oder direkt mit einem großen Knotenpunkt, wie dem *DE-CIX*[4] oder dem *AMS-IX*[5], vernetzt sind. Jedoch ist die letztendliche Übertragungsleistung immer nur maximal so schnell wie die Internetverbindung des Benutzers.

Der Besuch eines Anwenders auf einer Webseite läuft auf Netzwerkebene schematisch immer gleich ab:

1. Der Besucher ist Kunde eines Telekommunikationsanbieters.
2. Der Telekommunikationsanbieter betreibt direkt oder über andere Telekommunikationsanbieter Pairing[6] mit einem Knotenpunkt (z.B. mit dem DE-CIX).
3. Das Rechenzentrum der Zieladresse ist Kunde mindestens eines weiteren Telekommunikationsanbieter.
4. Dieser Telekommunikationsanbieter betreibt wiederum auch direkt oder indirekt Pairing.
5. Durch das Pairing im Knotenpunkt kann also die Anfrage vom Besucher aus dem Netz des Telekommunikationsanbieters zu einem Netz eines anderen Telekommunikationsanbieters weitergereicht werden.

Das folgende Beispiel zeigt die Verfolgung der Netzwerkroute mittels »traceroute« unter Linux. Unter den Punkten 6 und 7 ist das Pairing zu sehen. Das Rechenzentrum des Telekommunikationsanbieters *ge3-0-0-pr2.fra.router.colt.net* ist am DE-CIX mit dem Rechenzentrum des Zielservers *decix-gw.hetzner.de* verknüpft.

[4] German Commercial Internet Exchange

[5] Amsterdam Internet Exchange

[6] Das Teilen eines Netzwerkknotenpunktes zwischen mehreren Teilnehmern.

```
# > traceroute 88.*.*.*
 1 10.244.*.*  (10.244.*.*)   1.253 ms
   ...
 6 ge3-0-0-pr2.fra.router.colt.net  (212.74.76.191)  10.399 ms
 7 decix-gw.hetzner.de              (80.81.192.164)  9.398 ms
   ...
10 88.*.*.*  (88.*.*.*)   13.851 ms
```

Das Pairing und das Routing zwischen den Knotenpunkten und Routern läuft in der Regel so schnell ab, dass es vom Benutzer überhaupt nicht wahrgenommen wird. Dazu kommt, dass Netzwerke durch die darüberliegenden Protokolle wie ATM oder TCP sehr fehlertolerant sind und Fehler ohne Zutun des Benutzers korrigieren. Anwender bekommen von Fehlübertragungen in den Softwarestacks im Betriebssystem oft nur dann etwas mit, wenn die Fehler gravierend sind.

Durch dieses »Blackbox«-Verhalten wird die Netzwerkkomponente von Anwendern bzw. Entwicklern überhaupt nicht wahrgenommen und fälschlicherweise werden folgende Eigenschaften als gegebenen betrachtet [Deu94]:

1. Das Netzwerk ist zuverlässig.
2. Die Latenz ist gleich null.
3. Die Bandbreite ist unbegrenzt.
4. Das Netzwerk ist ausfallsicher.
5. Die Topologie wird sich nicht ändern.
6. Es gibt genau einen Netzwerkanbieter.
7. Die Transportkosten sind gleich null.
8. Das Netzwerk ist homogen.

Diese Annahmen führen oft zu Performance-Problemen, insbesondere die Latenz (Punkt 2), die Bandbreite (Punkt 3) und die Transportkosten (Punkt 7) verursachen bei webbasierten Systemen die größten Probleme. Oft wird kein Gedanke an die Größe der Dateien, die Bandbreite und die Latenz verschwendet, da davon ausgegangen wird, dass diese nur bei Besuchern mit einem Internetzugang mit geringer Bandbreite zu einem langsamen Seitenaufbau führen können. Werden im Serverumfeld zusätzlich Caches oder Queuing-Services verwendet, können die Bandbreite und die Zuverlässigkeit des Netzwerkes ebenfalls eine wichtige Rolle spielen, z.B. wenn mehr Daten übertragen werden müssen, als Bandbreite verfügbar ist.

1.5.3 Betriebssystem & Virtualisierung

Insbesondere das Betriebssystem, das Zugriffe auf die Systemressourcen verwaltet, ist neben der Hardware ein wichtiger Faktor. Zwischen einzelnen Versionen von Betriebssystemen können große Performance-Unterschiede herrschen. Als Beispiel konnte ein großes Unternehmen aus der Online-Werbebranche 10 von 25 veralteten SuSe 11 Server durch Wechsel auf eine neuere Linux-Distribution (Debian Squeeze) abschalten, da die verbleibenden 15 Server leistungsfähiger waren als die vorigen 25 Server – bei gleich bleibender Hardware.

Das Betriebssystem ist die Schnittstelle zur Hardware und wird in den meisten Fällen bei der Performance-Optimierung außer Acht gelassen. Es kann sich jedoch lohnen, einen Test mit einer neueren Betriebssystemversion zu wagen. Neben der Wahl des Betriebssystems sind die Einstellungen im Betriebssystem ein wichtiger Faktor. Da über das Betriebssystem die Hardwarekomponenten justiert werden, können Änderungen an den Festplatten-Caches, Sektorgrößen, Swap-Größe, Scheduler und die Wahl des Dateisystems essenzielle Auswirkungen haben.

Des Weiteren sind Betriebssysteme im Serverumfeld meist virtualisiert. Die Virtualisierung von Betriebssystemen bringt Vorteile bei der Verwaltung von Anwendungsservern und eine effizientere Ausnutzung der Hardware, abstrahiert allerdings die Systemressourcen vom Betriebssystem und zieht somit eine weitere Ebene zwischen Hardware und Software ein. Diese zusätzliche Ebene kann sich unter Umständen negativ auf die Performance auswirken. Versuche mit verschiedener Virtualisierungssoftware können sich daher durchaus lohnen.

1.5.4 Programmiersprache, Frameworks & Bibliotheken

Die Aktualität der Version der eingesetzten Programmiersprache und der darauf aufbauenden Tools ist wichtig, da diese Bugfixes und neue Features enthalten, die die Performance positiv beeinflussen können. Gerade bei jungen Sprachen wie z.B. *Ruby* werden von Release zu Release viele Performance-Verbesserungen veröffentlicht. Abgesehen von der Version der Programmiersprache ist die Wahl einer für die Anforderung geeigneten Programmiersprache ebenfalls ausschlaggebend für die Performance. Verschiedene Programmiersprachen lösen Anforderungen auf unterschiedliche Weise und eignen sich daher besser oder weniger gut für eine spezielle Aufgabe. Dem gegenüber steht sicherlich immer das Portfolio der vorhandenen Entwickler und die eventuell anstehenden Kosten für weitere Entwickler, die die entsprechende Programmiersprache beherrschen.

In der Softwareentwicklung werden sowohl bei Web- als auch bei Desktop-Anwendungen server- und clientseitigen Frameworks und Biblio-

theken eingesetzt. Frameworks haben das Ziel, immer wieder auftretende Tätigkeiten zu abstrahieren und dem Entwickler eine einfache Möglichkeit zu geben, diese zu verwenden. Die Abstraktion von speziellen Aufgaben auf einen gemeinsamen Nenner bedeutet allerdings oftmals, dass Einbußen in der Performance zugunsten der Strukturierung und der einfachen Verwendung hingenommen werden müssen. Durch die Verwendung von Frameworks können einzelne Teile in der Anwendung durch die Abstraktion weniger performant sein als eine Implementierung ohne Framework. Manche Frameworks sind jedoch auch flexibel und bieten die Möglichkeit, für diverse Aufgaben den Standardweg, den das Framework vorgibt, zu umgehen. Dieses Vorgehen wird für performanceintensive Bereiche in der Software empfohlen. Allgemein betrachtet ist der Einsatz eines Frameworks bei der Entwicklung immer von Vorteil, bei der Performance-Optimierung sollten jedoch Framework-Funktionen hinsichtlich ihrer Ausführungsgeschwindigkeit evaluiert werden.

Neben der Version der Programmiersprache ist auch auf die Aktualität der Versionen von Frameworks, Bibliotheken und Treibern zu achten. Bei Treibern für die Kommunikation mit anderen Systemen wie Queues, Caches oder ähnlichen Diensten können die Verbesserungen in der Performance zum Teil gravierend sein. Gleiches gilt für die Clientseite: Neue Browserversionen bringen häufig auch neue performantere Funktionen mit. Um diese nutzen zu können, müssen die verwendeten Bibliotheken, wie zum Beispiel die JavaScript-Bibliothek *jQuery*, auch in einer aktuellen Version eingesetzt werden. Als Beispiel sei die in Firefox 3.1 eingeführte native `document.querySelectorAll`-Methode zu nennen. Durch diese können JavaScript-Bibliotheken DOM-Elemente per CSS-Selektor 2- bis 6-mal schneller auffinden als die bisherigen eigenen Implementierungen der Bibliotheken [Res08].

Allerdings ist zu beachten, dass neue Versionen auch neue Fehler enthalten können oder sich Zugriffe auf Funktionen der Bibliotheken geändert haben können. Vor dem Einsatz dieser sollten zuerst die Änderungen der Entwickler in den entsprechenden Blogs oder Change-Logs überprüft werden.

1.5.5 Architektur & Implementierung

Die Architektur und die Implementierung einer Software sind ein ausschlaggebendes Kriterium für die Performance. Ist eine Software unstrukturiert und ineffektiv, ist sie langsam und verbraucht viele Systemressourcen. Beispiele aus der theoretischen Informatik zeigen, wie Teile von Software einfach optimiert werden können.

Folgender Beispielcode vergleicht zwei Arrays von Produktobjekten und gibt die Übereinstimmungen beider Arrays in einem dritten Array

arrayFound zurück. Codebeispiel 1.1 zeigt eine schlechte Implementierung, die zwei ineinander geschachtelte For-Schleifen enthält. Die Ausführungskomplexität beträgt $n*m$ (Länge von arrayProductSearch multipliziert mit der Länge von arrayProductList) und der Vorgang kann je nach Größe der Arrays sehr lange dauern.

```
var arrayProductSearch
var arrayProductList
var arrayFound
FOR EACH product1 IN arrayProductList
  FOR EACH product2 IN arrayProductSearch
    IF product1.productid == product2.productid
      PUSH product1 TO arrayFound
    END IF
  END FOR
END FOR
```

Codebeispiel 1.1: Beispiel mit einer Komplexität von $n * m$

Sind alle gesuchten Produkte in arrayProductSearch enthalten, kann als kleine Verbesserung die For-Schleife abgekürzt werden, indem die Länge von arrayProductSearch und arrayFound verglichen wird. Stimmt die Länge der Arrays überein und wurden alle Produkte gefunden, kann die Suche durch BREAK beendet werden (siehe Codebeispiel 1.2).

```
var arrayProductSearch
var arrayProductList
var arrayFound
FOR EACH product1 IN arrayProductList
  FOR EACH product2 IN arrayProductSearch
    IF product1.productid == product2.productid
      PUSH product1 TO arrayFound
      IF LENGTH OF arrayProductSearch == LENGTH OF arrayFound
        BREAK
      END IF
    END IF
  END FOR
END FOR
```

Codebeispiel 1.2: Beispiel mit einer Komplexität von $n*m$ und Abbruchbedingung

Da die Verbesserung nur für den Fall zutrifft, wenn alle gesuchten Produkte im Array arrayProductSearch enthalten sind, ist diese Optimierung nur bedingt wirksam. Im folgenden Codebeispiel 1.3 wird ein Ansatz gezeigt, der die Ausführungszeit auf $n + m$ reduziert und damit im Vergleich sehr effizient ist. Dafür werden zuerst alle Produkt-IDs (hier: *productid*) in

einem assoziativen Array[7] gespeichert. Im zweiten Schritt wird über die Produktliste einmal iteriert und jede Produkt-ID im assoziativen Array `arraySearchAsoc` gesucht. Wenn die Produkt-ID vorhanden ist, wird sie zum Array `arrayFound` hinzugefügt. Wie im vorigen Beispiel kann eine Abbruchbedingung hinzugefügt werden. Um welchen Faktor die Effizienz gesteigert werden kann, hängt hier jedoch von der internen Implementierung von assoziativen Arrays der Programmiersprache ab.

```
var arrayProductSearch
var arrayProductList
var arrayFound
var arraySearchAsoc

FOR EACH product2 IN arrayProductSearch
  arraySearchAsoc[product2.productid] = product2
END FOR

FOR EACH product1 IN arrayProductList
  IF arraySearchAsoc HAS KEY product1.productid
    PUSH product1 TO arrayFound
    IF LENGTH OF arrayProductSearch == LENGTH OF arrayFound
      BREAK
    END IF
  END IF
END FOR
```

Codebeispiel 1.3: Beispiel mit einer Komplexität von $n + m$

Dieser Beispielcode zeigt anhand einer einfachen Implementierung eines naiven Suchalgorithmus von Produkten in Arrays, wie dieser optimiert werden kann. Die Optimierung der Implementierung und Architektur ist sehr vielseitig und aufwendig. Die Optimierungsmöglichkeiten hängen vom Anwendungsfall ab, weshalb es keinen generellen Ansatz gibt. In den Folgekapiteln werden allerdings einige Möglichkeiten zur Identifizierung von langsamen Codebereichen vorgestellt.

1.5.6 Datenhaltung

Die Datenhaltung in Webanwendungen ist vielseitig. Serverseitig werden Datenbanken oder Dateien verwendet, um die Daten der Anwendung zu speichern. Diese Datenbanken oder Dateien werden von der Webanwendung gelesen und aufbereitet an den Client übertragen. Im Frontend-Bereich werden clientseitige Datenbanken oder Dateien verwendet, die sich auf der lokalen Festplatte des Benutzers befinden.

[7] *<product-id, product-object>* $\Longleftrightarrow$ *<key, value>*

Die meisten Webanwendungen benutzen heutzutage relationale Datenbanken. Die bei Webanwendungen am häufigsten eingesetzten (relationalen) Datenbanken sind *MySQL* und *Postgres*. Beide Datenbanken sind in einer kostenlosen Version verfügbar und zeichnen sich durch eine gute Performance und eine langjährige Marktreife aus. Im kostenpflichtigen Bereich sind *MsSQL*, *Oracle* und *Informix* zu finden, die meist in Webanwendungen in Unternehmen eingesetzt werden, wenn auf bestehende Daten zugegriffen werden soll. Die Geschwindigkeit und Tools zur Optimierung der kostenpflichtigen Datenbanken sind nicht zu verachten.

Neben den relationalen Datenbanken gibt es auch die schemalosen Datenbanken, die oftmals als »NoSQL« referenziert werden. Als große Vertreter der schemalosen Datenbanken gelten *MongoDB* (dokumentenorientiert), *CouchBase* (dokumentenorientiert), *Riak* (Key-Value Store) und *Cassandra* (Key-Value Store). Schemalose Datenbanken haben im Vergleich zu den genannten relationalen Datenbanken noch nicht die volle Marktreife erreicht, da diese vergleichsweise jung sind und erst in den letzten Jahren eine breite Anerkennung in Entwicklerkreisen gefunden haben. Nichtsdestotrotz haben schemalose Datenbanken ein unglaubliches Potenzial und sind den relationalen Datenbanken in puncto Performance und Skalierbarkeit durch den Paradigmenwechsel von festen, relationalen Strukturen hin zu flexiblen, schemalosen Strukturen, die durch die großen »Success Stories« wie Twitter, Facebook und Co. beeinflusst wurden, einen Schritt voraus.

Bei großen Anwendungen, die viele Daten sammeln und bearbeiten, kann die Datenbank schnell zum Flaschenhals werden. Die Flaschenhälse ergeben sich allerdings oftmals aus ineffizienten Zugriffen auf die Datenbank oder durch fehlerhafte Strukturierung des Datenbankschemas. Hierbei ist es egal, ob relationale oder schemalose Datenbanken verwendet werden, da die Art des Zugriffs und die Struktur der Daten für die Performance ausschlaggebend sind. Dazu finden sich in Kapitel 4 viele weitere Informationen zu den Optimierungsmöglichkeiten.

1.5.7 Drittanbieter

Jede Komponente in einer Webanwendung kann zum Flaschenhals werden. Beim Einsatz von Drittanbieterdiensten, d.h. Schnittstellen (APIs) sowohl zu Services wie Twitter, Facebook, Geo-Anbietern etc. als auch Hostern, Cloud-Services oder Content-Delivery-Networks muss damit gerechnet werden, dass diese zum Flaschenhals werden können. Drittanbieter müssen daher wie »Blackboxes« behandelt werden. Die Performance und Verfügbarkeit des Anbieters muss als unbekannt angesehen werden. Verspricht ein Anbieter 99,999 % Verfügbarkeit, kann es trotzdem zu tagelangen Ausfällen kommen, was sich auf das eigene Businessmodell und Kundenvertrauen

negativ auswirken kann. Die Gewährleistung der Verfügbarkeit ist meist nur eine rechtliche Gewährleistung im betriebswirtschaftlichen Sinne.

Die Performance von Drittanbietern kann gemessen werden, allerdings kann sie in den seltensten Fällen optimiert werden. Geeignete Fehlerbehandlung, Fehlermodelle und Backup-Anbieter sollten in der Anwendung vorgesehen werden.

1.5.8 Frontend & Usability

Eine schlechte Usability einer Webanwendung wirkt sich nicht nur negativ auf die Kundenzufriedenheit aus, sondern hat ebenfalls auch nachteilige Auswirkungen auf die Performance. Kann ein Anwendungsbenutzer sein Ziel nur über Umwege erreichen, erzeugt er durch den Umweg unnötige Last auf den Webservern. Werden bei dem Umweg beispielsweise jedes Mal intensive Berechnungen oder diverse Datenbankzugriffe ausgeführt, kostet dies unnötige Ressourcen (Rechenleistung, Bandbreite, Energie etc.). Weiteren Webseitenbesuchern werden dadurch eventuell andere Inhalte ebenfalls verlangsamt ausgeliefert. Die Optimierung der Benutzerführung kann einen großen Performance-Schub der Anwendung bewirken.

Neben der Usability kann der clientseitige Teil der Webanwendung selbst zum Flaschenhals werden. Sind serverseitige Operationen zügig abgeschlossen und die Daten schnell an den Client übertragen (vgl. Abschnitt 1.5.2), kann es trotzdem auf der Clientseite zu Performance-Problemen kommen. Eine langsame Verarbeitung und zu häufiges Aktualisieren der Daten vom Server, ohne einen client- oder serverseitigen Caching-Mechanismus, kann zu einem langsamen Verhalten der Anwendung führen. Zudem können komplexe Seitenstrukturen und -inhalte die Ausführungsgeschwindigkeit im Browser bremsen und die für den Nutzer wahrgenommene Lade- und Reaktionszeit der Seite dadurch negativ beeinflussen.

1.6 Best Practices

Performance-Optimierungen sind nicht immer einfach umzusetzen, da mitunter sehr mühselig die eingesetzte Software, Schnittstellen zu weiteren Komponenten oder Codefragmente getestet und ggf. verändert werden müssen. Beim Optimierungsvorgang sollten deshalb folgende Schritte eingehalten werden:

- **Kein frühzeitiges Optimieren:** Ein Entwickler sollte zuerst funktionale Anforderungen der Applikation programmieren und sich zunächst weniger auf zum Teil komplizierte Optimierungen stürzen. Optimierte Anwendungsteile, Codesegmente oder Code von Dritten

kann dazu führen, dass die Applikation unnötig unverständlich für den/die Entwickler wird. Für Anfänger ist es empfehlenswert, die Anwendung und den Programmcode am Anfang einfach zu halten, mögliche Performance-Schwachstellen zu markieren[8] und erst zu einem späteren Zeitpunkt komplexe Optimierungsschritte vorzunehmen.

- **Blindes Optimieren nützt wenig:** Zuallererst sollte mithilfe eines Profilers oder einfacher Zeitmessungen die kritische Schwachstelle ermittelt werden. Ist die Schwachstelle gefunden, kann diese mit den entsprechenden Mitteln angepasst und optimiert werden. Zudem hat es sich bewährt, Optimierungen in einem *A*/*B-Test* gegeneinander zu vergleichen. Hierbei wird die Performance des ursprünglichen Systems *A* mit einer optimierten Version *B* verglichen und evaluiert, ob die Performance sich verbessert hat. Optimierungen sollten immer schrittweise durchgeführt und für sich getestet werden.
- **Programmcode von Dritten birgt Risiken:** Wie schon im ersten Punkt beschrieben, kann Programmcode von Dritten die Anwendung unnötig unverständlich machen. Bevor Programmcode von Dritten eingesetzt wird, sollte Zeit investiert werden, um diesen exakt zu verstehen. Ein Codesegment, das komplex und schlecht kommentiert ist, kann viel Einarbeitungsaufwand verursachen und ungewollte Nebeneffekte hervorrufen.
- **Einfacher Programmcode erleichtert die Arbeit:** Der entwickelte Programmcode sollte immer so einfach und verständlich wie möglich gestaltet werden, auch wenn dabei mehr Zeilen Code nötig sind. Weiterhin tragen Kommentare zu einem besseren Verständnis bei. Für zukünftige Entwickler sind einfacher Programmcode und Kommentare nur von Vorteil und erleichtern die Einarbeitung sowie die Wartung und Optimierung.
- **Wenig Programmcode läuft nicht unbedingt schnell** und viel Programmcode nicht unbedingt langsam. Es kommt immer auf die verwendeten Strukturen, logischen Bausteine und den eigentlichen Algorithmus an, der ein Problem lösen soll. Die Größe des Programmcodes gibt also nur bedingt Auskunft über die Laufzeit. In Voraussicht auf die Datenübertragung zum Client ist kurzer und möglichst schneller HTML-, CSS- und JavaScript-Code zu bevorzugen.

[8] Beispielmarkierungen für spätere Optimierungen: `@TODO` oder `@FIXME`.