Seminar
Grundlagen webbasierter Informationssysteme
– Web-Caching –

Sebastian Adam

kontakt@sebastian-adam.de

Zusammenfassung. Die derzeitige Nutzung des Internets führt zu einer immer höheren Rechen- und Netzwerkbelastung, welche größere Maße angenommen hat, als alleine durch moderne Hardware zu bewältigen ist. Web-Caching hat sich daher als Technik etabliert, Webserver und Netzwerke zu entlasten und HTTP-Requests über schnelle Zwischenspeicher zu beantworten. Die Probleme bei dynamischen Inhalten stellen dabei besondere Herausforderungen dar, welchen mittels moderner Implementierungskonzepte begegnet werden kann.

1 Motivation

Mit über 4,25 Mrd. in Google erfassten Webseiten [1], 22 Mio. permanent am Netz angeschlossenen Rechnern [2] und mehr als 34 Mio. allein deutschen Nutzern [3] hat das Internet noch lange nicht sein Wachstum beendet.

Trotz dieses gigantischen Angebotes im World Wide Web entfällt jedoch der größte Anteil aller Seitenzugriffe nur auf eine relativ geringe Anzahl von Webseiten. Solche Online-Plattformen wie in Deutschland beispielsweise eBay, Amazon, Google, GMX, T-Online werden täglich von vielen tausend Nutzern aufgerufen, während die mit Abstand meisten anderen Auftritte deutlich weniger als tausend Besucher pro Woche zu verzeichnen haben.[1]

Trotz permanentem Geschwindigkeitszuwachs der Serverrechner und breiterer Netzwerkanbindungen sind jedoch selbst modernste Systeme heute noch nicht in der Lage, die Flut von Anfragen rein hardwaretechnisch zu bewältigen. Die derzeitigen Lösungen sind zwar für die parallele Nutzung durch einige hundert Nutzer gut geeignet, nicht aber für einige hunderttausend. Folgendes Beispiel verdeutlicht dieses Problem:

Eine durchschnittliche eBay-Seite besteht aus insgesamt mehr als 220 KB HTML-, JavaScript- und Grafikdaten. Neben der Tatsache, dass bei dieser Seitengröße ein Surfen auf eBay fast nur für Breitband-Nutzer in akzeptabler Qualität möglich ist, stellt insbesondere die enorme Netzbelastung ein wesentliches Problem dar. Mit monatlich 7,3 Mrd. Seitenaufrufen [4] (rund 3000 Aufrufe pro Sekunde) entsteht dadurch – vorausgesetzt, der Server kann diese Anfragelast überhaupt bearbeiten – bei typischen HTTP-Requests (siehe Abb. 1) eine Transferlast von mehr als 4,6 GBit pro Sekunde. Dieser Netzbelastung sind selbst große Backbone-Leitungen kaum gewachsen.

[1] eBay Deutschland verzeichnet über 1 Mrd. mehr Seitenzugriffe pro Monat als alle knapp 1,5 Mio. von Marktführer 1&1 gehosteten Web-Auftritte zusammen [4, 5].

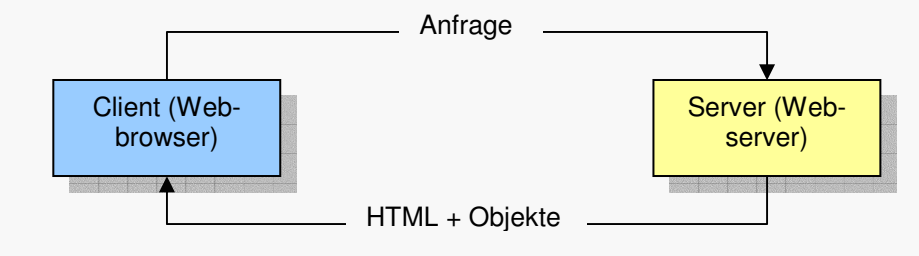


Abb. 1. Klassisches HTTP-Request-Response

Bei der genauen Analyse der übertragenen Daten lässt sich aber erkennen, dass nur ein relativ geringer Anteil den tatsächlich anfragespezifischen Inhalt umfasst. Die Mehrzahl der Daten (im eBay-Beispiel rund 160 KB) werden auf fast jeder Seite der Internetpräsenz verwendet und sind von dem konkreten Request, beispielsweise der Suchanfrage, weitestgehend unabhängig.

Der einfachste Weg, eine Netzüberlastung und Übertragungsverzögerung zu vermeiden, lässt sich daher durch die Eliminierung von unnötigem Datentransfers erzielen. Dies kann dadurch erreicht werden, dass eine Kopie aller unveränderlichen Bestandteile (z.B. von Bildern) außerhalb des Rechenzentrums in einem schnellen, clientnahen Zwischenspeicher abgelegt wird.

Beim Stellen eines HTTP-Request wird dann der Server lediglich mit der Auslieferung ungespeicherter Inhalte beauftragt, während bereits vorhandene Seiten(-Elemente) aus dem schnellen Zwischenspeicher bezogen werden können (siehe Abb. 2). Je nach Standort dieses Speichers lässt sich dadurch die Netzlast zum Server reduzieren, wodurch ein höherer Durchsatz möglich wird [7].

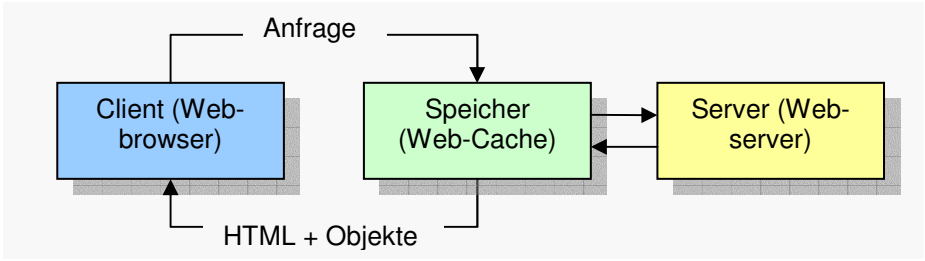


Abb. 2. Grobes Prinzip einer Zwischenspeicher-Nutzung (Proxy-Prinzip)

Eine solche Zwischenspeicherung von Web-Inhalten (neben Grafiken auch von Multimedia-Objekten, ganzen HTML-Seiten, etc.) bezeichnet man als Web-Caching und den jeweiligen Speicher als (Web-)Cache.

Die Anwendung von Web-Caching und die unterschiedlichen Platzierungs- und Implementierungsmöglichkeiten zielen aber neben der Reduzierung des Netzwerk-Transfers auch auf die Minimierung der Rechenlast der beanspruchten Server ab. Wie im obigen Beispiel skizziert, wird von manchen Systemen gefordert, mehr als 3000 Seiten pro Sekunde zu generieren und auszuliefern, was selbst moderne Applikationsserver kaum erreichen. Eine Zwischenspeicherung von fertig generierten Seiten oder -fragmenten kann in diesem Fall jedoch selbst solch hoch frequentierte Systeme befähigen, zahlreiche Anfragen parallel zu beantworten (Skalierbarkeit).

Nicht zuletzt auch im Hinblick auf den Übertragungsverlauf, insbesondere bei multimedialen Inhalten, kann Web-Caching die Qualität erhöhen. Bei einer klassischen Verbindung vom Browser-Client zum Zielserver läuft die Verbindung in der Regel über mehrere Netzverbindungsknoten. Dadurch wird die gesamte Übertragungsgeschwindigkeit durch das langsamste Teilnetz bestimmt, was bei hoch belasteten Netzknoten eine Reduzierung der Wiedergabequalität bedeuten kann [7].

Ein Web-Cache gewährleistet in diesem Fall eine zuverlässigere Übertragung, da die angeforderten Daten näher am Client gehalten werden, wodurch die Anzahl der zu überwindenden Teilstrecken und somit das Verzögerungsrisiko abnehmen.

Web-Caching bietet also auf Serverseite den Vorteil der Skalierbarkeit, während den Clients eine Reduzierung der Antwortzeit/Ladezeit ermöglicht wird, was auf beiden Seiten mit einer Qualitäts- und Zuverlässigkeitssteigerung einhergeht. Nicht zuletzt im Hinblick auf wirtschaftliche Ziele ist dieser Aspekt für die Betreiber von großen Onlineshops ein wichtiges Kriterium.

Trotz der zahlreichen Vorteile von Web-Caching stellen sich bei der praktischen Umsetzung einige Probleme, die es zu lösen gilt. Nachstehend seien vier prominente exemplarisch aufgeführt:

- Wie kann überhaupt gewährleistet werden, dass meine Daten in einem Cache vorhanden sind?
- Wie wird erreicht, dass die aus dem Zwischenspeicher bezogenen Daten zu diesem Zeitpunkt mit den neusten Daten des Zielservers übereinstimmen?
- Welche Inhalte dürfen aus Gründen der Sicherheit und Personalisierbarkeit überhaupt zwischengespeichert werden?
- Wie kann man personalisierte und andere dynamische Inhalte cachen?

Aufgrund der Wichtigkeit von Web-Caching für die effiziente Arbeite des modernen Webs möchte ich in dieser Seminararbeit die grundlegende Funktionsweise von Web-Caching-Systemen beleuchten und konkrete Lösungskonzepte für die skizzierten Probleme vorstellen.

Kapitel 2 behandelt dabei allgemeine Aspekte, während Kapitel 3 die Probleme und Ansätze im Hinblick auf dynamische Webseiten diskutiert. Kapitel 4 stellt zwei kommerzielle Systeme vor und Kapitel 5 schließt mit der Betrachtung verwandter und fortgeschrittener Arbeiten.

Da das Gebiet „Web-Caching“ sehr breit gefächert ist und es, begonnen bei Netzwerkstrukturen über Algorithmen bis hin zu Protokoll- und Sprachkonstrukten, zahlreiche Aspekte beleuchtet, habe ich in dieser Arbeit bewusst den Fokus auf die Themen, die zum grundlegenden Verständnis und zur Anwendung aus Sicht des Entwicklers nötig sind, gelegt.

2 Grundlagen des Web-Caching

Dieses Kapitel erläutert neben der grundlegenden Funktionsweise eines Web-Caches und dessen möglicher Platzierung insbesondere das in der Praxis verwendete Verfahren zur Gewährleistung von Datenkonsistenz zwischen Webserver und -Cache.

2.1. Grundlegende Funktionsweise eines Web-Caches

Ein Cache ist im Allgemeinen ein Zwischenspeicher, der aktuell benötigte Daten enthält und diese schneller ausliefern kann, als dies dem eigentlichen Quellspeicher möglich ist.

Bei einem Web-Cache handelt es sich dabei konkret um ein System[2], welches oft gefragte Web-Inhalte zwischenspeichert, um dadurch die eigentlichen Webserver von unnötigen Anfragen zu entlasten und gleichzeitig den Nutzern eine schnellere Bereitstellung dieser Inhalte zu ermöglichen [9]. Um diesen Vorteil jedoch auch nutzen zu können, muss sowohl die Einbeziehung des Web-Cache in die Client/Server-Kommunikation, als auch eine relativ hohe Hit-Rate[3] gewährleistet werden.

---

[2] In der Regel als Software aufbauend auf Standardbetriebssystemen implementiert. Alternativen bestehen in Hardwarelösungen mit Spezialsoftware.

[3] Die gewünschten Informationen liegen im Cache vor. Das Gegenteil wird als „Miss“ bezeichnet.

Die erste Leistungsanforderung an einen Web-Cache wird hierbei durch seine Platzierung auf dem Kommunikationspfad zwischen Client und Server erreicht. Dafür bieten sich Netzwerkknoten an, die bei einem HTTP-Request in jedem Fall durchlaufen werden, wie beispielsweise die Schnittstelle des Client-Rechners zum Netzwerk, die Schnittstelle des Service Provider[4] zum Internet oder die des Zielservers zum nächsten Backbone.

Die zweite Anforderung wird in der Praxis implizit durch das in Kapitel 1 dargestellte Zugriffsverhalten (viele Zugriffe auf relativ wenige Seiten) erfüllt: Während in cachefreien Systemen eine hohe Anzahl von ähnlichen Internetanfragen zu Engpasssituationen führen kann, stellt der gleiche Sachverhalt in cachebasierten Netzwerken die Grundlage dafür dar, dass häufig gebrauchte Ergebnisse in einen Web-Cache gelangen (siehe Beschreibung in Abb.3).

Zusammenfassend stellt sich daher die Frage nach der idealen Platzierung eines Web-Caches.

Je näher ein Cache beim Client platziert wird, desto kürzer ist der Verbindungspfad und umso geringer die Antwortzeit bei Anfragen, die vom Cache selbst beantwortet werden können. Dem gegenüber enthält dieser Cache aber potentiell weniger Inhalte als ein clientferner, da nur ein geringer Kreis an Nutzern für dessen Inhalt „verantwortlich“ ist. Die Bestimmung einer einzigen optimalen Cache-Platzierung ist daher nicht möglich.

Einlagerung von Inhalten in einen Web-Cache[5]

1. Ein HTTP-Request eines Clients zu einem Webserver erreicht einen Netzknoten mit darauf installiertem Caching-System. Der Cache vergleicht die Anfrage mit seinem Datenbestand. Besitzt er eine gültige Version der angeforderten Inhalte führt er Schritt 5 aus, sonst Schritt 2.
2. Der Request wird an den zuständigen Server weitergeleitet.
3. Mittels einer Ersetzungsstrategie (siehe Abschnitt 2.3.) wird innerhalb des Cache freier Speicherplatz zur Verfügung gestellt.
4. Der Webserver liefert die angeforderten Daten an den Netzknoten zurück. Dieser legt die Daten in seinem Cache ab.
5. Die angeforderten Inhalte werden an den Client ausgeliefert.

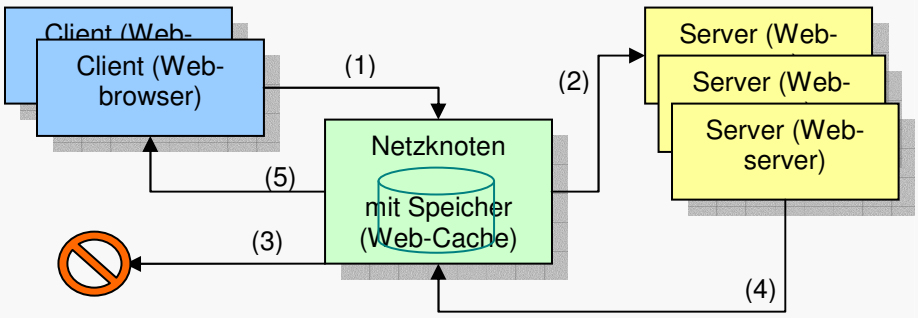


Abb. 3. Prinzip der Einlagerung von Inhalten in einen Web-Cache

Insbesondere im Hinblick auf die unterschiedlichen Performance-Ziele (Serverentlastung vs. Reduzierung der Antwortzeit) ist daher die Anwendung mehrerer Web-Caches entlang eines typischen Verbindungspfades sinnvoll.

[4] Einwahlprovider, z.B. T-Online, AOL, …

[5] Vereinfachte Darstellung ohne Cache-Hierarchie und “not-modified”-response.

2.2. Arten und Platzierungen von Web-Caches

Der Pfad einer Verbindung zwischen Client und Server beginnt im Webbrowser einer Workstation und endet in der Applikation des aufgerufenen Zielservers. Die einzelnen Caches und ihre Aufgaben seien nachstehend kurz aufgezeigt (siehe Abb. 4):

Browser-Cache: Im Rechner des Nutzers starten die HTTP-Requests durch die Eingabe einer URL oder der Auswahl eines Hyperlinks. Hierher gelangen letztendlich die angeforderten Inhalte zwecks Darstellung auch zurück. Der Rechner des Nutzers ist daher der nächstmögliche Ort für das Caching von Web-Inhalten. Ziele sind dabei, ein schnelles Navigieren zwischen bereits besuchten Seiten zu unterstützen und das zeitintensive Mehrfach-Laden bereits erhaltener Elemente zu vermeiden. Der Nachteil eines Browser-Caches besteht allerdings darin, dass die Inhalte des Caches nur auf dem entsprechenden Rechner nutzbar sind, wodurch keine wesentliche Netzentlastung oder Antwortzeitminimierung erreicht werden kann.

Lokaler Proxy-Cache: Insbesondere in Organisationen, vermehrt aber auch in privaten Haushalten, existieren Netzwerke. Die einzelnen Workstations der Nutzer verfügen hierbei nicht über eigene Internetzugänge durch Modems, sondern erhalten über ihre lokale Netzwerkschnittstelle Zugang zum Web. Insbesondere in größeren Netzwerken werden hierfür Proxy-Server eingesetzt, welche als Stellvertreter für die einzelnen Clientrechner die Internetkommunikation abwickeln und dadurch die lokale Netzstruktur verbergen. Durch diese zentralisierte Zugriffskontrolle bieten solche Firewall-Proxy-Server neben hoher Sicherheit insbesondere die Möglichkeit, bezogene Web-Inhalte lokal zwischenzuspeichern und für alle Netzwerkrechner verfügbar zu machen. Da in Organisationen viele Nutzer oft auf dieselben Informationen im Internet zugreifen, ermöglicht dies in der Praxis die oft schmalen Verbindungen ins Internet zu entlasten und Inhalte schneller bereitzustellen. In großen Unternehmen werden darüber hinaus Proxy-Caches bereits auf Abteilungsebene installiert, um ebenfalls das Unternehmensnetzwerk, insbesondere den Firewall-Proxy, zu entlasten. [8]

ISP-Proxy-Cache: Die eigentliche Verbindung zum Internet stellt die Mehrzahl der privaten und gewerblichen Nutzer über Zugänge von Internet Service Providern (ISP) her. In der Regel handelt es sich dabei um Breitband-Anschlüsse wie DSL oder schmale Standleitungen von wenigen MBit/s. In den Einwahlzentren der Anbieter, die die Schnittstelle zwischen Telefonnetz und Internet bilden, laufen die Anfragen von mehreren hunderttausend Nutzern zusammen. Durch diese hohe Anzahl von Clients und das zu Beginn beschriebene Zugriffsverhalten erzielen hier platzierte Proxy-Caches Hitraten von bis zu 50% [9]. Dadurch können Antwortzeit und Netzwerktransfer deutlich reduziert werden.

Nationale & internationale Caches: An wichtigen nationalen und internationalen Internetknoten sind ebenfalls Web-Caches installiert, welche häufig frequentierte Inhalte von den daran angeschlossenen Rechenzentren enthalten.

Proxy-Cache im Rechenzentrum (Reverse-Cache): Webserver sind in der Regel nicht direkt am Internet angeschlossen, sondern in Netzwerke großer Rechenzentren integriert. Die Kommunikation mit der Außenwelt erfolgt auch hier mit Hilfe von Proxy-Servern. Diese nehmen stellvertretend für die eigentlichen Server die Anfragen entgegen und beantworten sie, soweit möglich, aus ihrem Cache. Dadurch wird eine hohe Anfrage- und Rechenlast für die Web- oder Applikationsserver vermieden.

Serverseitiges Caching: Innerhalb des eigentlichen Servers kann ebenfalls ein Caching erfolgen [10]. Hierbei handelt es sich neben den eigentlichen Web-Inhalten auch um Daten oder Fragmente, die zur Erzeugung der Seiten notwendig sind. Dies können insbesondere Ergebnisse von Instruktionsausführungen oder Datenbankabfragen sein.

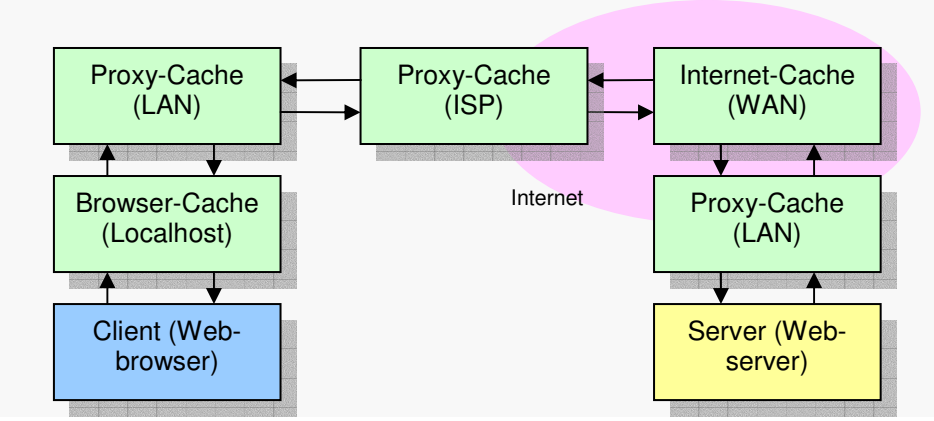


Abb. 4. Aufbau einer Web-Caching-Hierarchie

Die Beschreibung der unterschiedlichen Caching-Standorte hat gezeigt, dass abhängig von der Platzierung auch die zugemessenen Aufgaben variieren. Während die so genannten Forward-Caches zwischen Client und Internet eine Reduzierung der Antwortzeit und des Netztransfers beabsichtigen, dienen die Reverse-Caches zwischen Internet und Zielserver einer Minimierung der Anfrage- und Rechenlast.

Trotz der Unterschiede in Architektur und Zielsetzung unterscheidet sich hierbei der Ablauf eines HTTP-Request nur unwesentlich von dem in Abb. 3. beschriebenen Vorgehen.

Beim Stellen einer Anfrage prüft zunächst der Browser, ob er aus seinem eigenen Cache die gewünschten Daten bereitstellen kann. Ist dies nicht gewährleistet, leitet er den Request an den nächst höheren Knoten weiter, welcher selbst diesen Vorgang wiederholt, bis schließlich ein Knoten erreicht ist, der die Anfrage beantworten kann. Der angeforderte Inhalt wird daraufhin in der Hierarchie nach unten zurückgegeben, wobei jeder zwischenliegende Knoten diese Daten in seinem Cache ablegt und gegebenenfalls zuvor alte Einträge entfernt.

Parallel zu dieser in der Praxis vorherrschenden Web-Caching-Architektur existieren am Markt kommerzielle Content Delivery Networks (CDN) [11]. Hierbei handelt es sich um ein interagierendes Netz von über die gesamte Welt verteilten Edge-Servern, welche die Verfügbarkeit von Inhalten ihrer betreuten Webseiten an allen wichtigen Internetknoten gewährleisten. Anders als herkömmlichen Web-Caches werden hierbei die Daten koordiniert auf die Edge-Server geleitet und nicht erst bei Anfrage eingelagert. Dadurch ist eine höhere Aktualität, ständige Verfügbarkeit und größere Sicherheit gewährleistet, als dies unter Verwendung der öffentlichen Netzkomponenten möglich wäre. Die nutzenden Kunden wie beispielsweise Amazon und MSNBC können dadurch, ohne ihre eigenen Rechenzentren zu skalieren, 100% Verfügbarkeit auch bei hoher Anfragelast gewährleisten. Die Aufgaben bezüglich der weltweiten Bereitstellung (insbesondere auch von Multimedia-Inhalten) werden von den CDN-Betreibern übernommen, während der eigentliche Anbieter sich nur noch um die Generierung der Inhalte zu kümmern braucht.

2.3. Qualitätsaspekte des Web-Caching

Durch (Forward-)Caching werden Kopien von Inhalten entfernter Zielserver in clientnahen Zwischenspeichern gehalten, um Anfragen schneller beantworten zu können. Die dadurch gewonnenen Leistungsvorteile bringen jedoch Probleme im Hinblick auf die Qualität, Speicherbelegung und Sicherheit der gefragten Inhalte mit sich.

Speicherbelegung: Trotz des enormen Wachstums von Speicherkapazitäten in den letzten Jahren haben Web-Caches nur eine begrenzte Speichergröße. Es ist daher erforderlich, dass Algo-

rithmen den vorhandenen Speicher effektiv nutzen, um die meistfrequentierten Angebote auch stets zur Verfügung stellen zu können.

Als eine der näherungsweise idealen Ersetzungsstrategien hat sich innerhalb der Betriebssystemwelt der Least-Recently-Used-Algorithmus (LRU) etabliert. Hierbei werden so lange die am längsten unbenutzten Einträge gelöscht bis ausreichend Speicherplatz zur Verfügung steht, um neue Daten einzulagern. Der LRU-Algorithmus liefert aber in Web-Caching-Systemen oft keine idealen Ergebnisse, da er neben dem Problem der Speicherfragmentierung insbesondere die Gültigkeit und Wichtigkeit eines Eintrags nicht berücksichtigt und auch auf Speicherfragmentierung keine Rücksicht nimmt.

In der Praxis hat sich daher der Time-To-Live-Algorithmus (TTL) durchgesetzt. Dabei werden alle Einträge bei Einlagerung mit einer Lebensdauer (Frist) versehen und gelöscht, sobald diese Frist überschritten ist. Steht nach einer solchen Löschung immer noch nicht ausreichend Speicherplatz zur Verfügung verwenden viele Caching-Systeme als sekundäre Strategie das LRU-Verfahren. Um die zeitintensiven Löschvorgänge zu optimieren und nicht permanent Löschaktionen starten zu müssen, setzen die Systeme dabei oft obere und untere Marken *high* und *low* ein, die den Start und das Ende des Löschvorgangs bestimmen. Erst sobald ein Cache mehr als *high* MB Daten enthält, werden gemäß den Algorithmen so lange Einträge gelöscht, bis nur noch *low* MB im Cache enthalten ist [6].

Da jedoch auch dieses Verfahren rechen- und zeitintensiv ist, ist der ideale Fall, eine Ersetzung nur dann vorzunehmen, wenn Einträge des Caches ohnehin nicht mehr gültig sind. Mit den steigenden Speicherkapazitäten wird sich dies in Zukunft auch annähernd realisieren lassen.

Eintragsgültigkeit: Ein damit eng verbundenes Problem, sicherlich das Hauptproblem des Web-Caching, ist die Gefahr, dass Einträge im Cache nicht mehr den aktuellen Daten auf dem Zielserver entsprechen. Dies kann beispielsweise bei Aktienkursen oder dem aktuellen Höchstgebot einer eBay-Auktion zu Irritationen führen.

Um dieses Problem zu vermeiden, muss der Web-Cache jederzeit gewährleisten, dass die von ihm bereitgestellten Daten zu diesem Zeitpunkt noch Gültigkeit besitzen. Die ständige Rückfrage bei der ursprünglichen Datenquelle bringt zwar diesbezügliche Gewissheit, allerdings widerspricht dieses Vorgehen den Zielen, die mit Caching erreicht werden sollen.

Der Web-Cache muss daher alleine durch Informationen in seinen Dateneinträgen wissen, ob diese zum aktuellen Zeitpunkt noch potentiell mit den Daten des Zielservers übereinstimmen. Die hierfür notwendigen Regeln können zum einen über die Einstellungen des Caches, zum anderen über Parameter im HTTP-Protokoll festgelegt werden.

Drei einfache Grundregeln ermöglichen auf Basis dieser Parameter eine schnelle Überprüfung der Gültigkeit [9].

- Wurde im HTTP-Header des Eintrages eine Ablauffrist angegeben, so gilt das Datum als gültig, falls diese Frist nicht überschritten wurde.
- Wurde der Browser-Cache so konfiguriert, dass er pro Session bzw. Zeiteinheit nur einmal die Gültigkeit überprüft und hat er ein gewisses Datum innerhalb dieser Zeit bereits neu erhalten, so gilt es als gültig.
- Wurde ein Datum erst relativ kürzlich in einen (Proxy-)Cache eingelagert und liegt die im HTTP-Header vermerkte letzte Änderung länger zurück, so gilt das Datum ebenfalls als gültig.

Um diese Gültigkeitskontrollen den Web-Caches zu ermöglichen, ist es wichtig, dass entsprechende HTTP-Informationen seitens des anbietenden Webmasters gesetzt werden. Diese Angaben können in der Konfiguration des Webservers für unterschiedliche Inhaltstypen feingranular oder über ein `http-equiv`-Meta-Tag im HTML-Dokument für die gesamte Seite eingestellt werden. [12].

Als wichtige Grundlage in HTTP 1.0 dient dabei der `Expires`-Header. Dieser teilt dem Cache mit, wie lange ein Objekt als gültig angesehen werden kann, im nachstehenden Beispiel also bis zum 31. Mai 2004, 14:19:41 Uhr.

```
Expires: Mon, 31 May 2004 14:19:41 GMT
```

Der Webserver kann für diesen Parameter, der in jedem Fall als absolute Datumsangabe festgelegt werden muss, entweder fixe Ablaufdaten angeben oder diese relativ zum letzten Änderungs- oder Zugriffsdatum berechnen. Wurde das `Expires`-Datum nicht gesetzt, kann der Cache aufgrund gewisser Vorgaben aus dem Zeitpunkt der letzten Änderungen dafür einen Schätzwert berechnen.

HTTP 1.1. bietet darüber hinaus flexiblere Möglichkeiten zur Cache-Kontrolle mittels des `Cache-Control`-Header. Einige wichtige Parameter seien kurz vorgestellt.

`max-age:`	Maximale Dauer in Sekunden, wie lange ein Eintrag nach Start des Requests als gültig betrachtet werden darf.
`public:`	Die Daten können in jedem Cache abgelegt werden, auch wenn dies standardmäßig nicht vorgesehen ist (beispielsweise authentifizierte Daten).
`private:`	Das Speichern der Daten ist nur auf dem Client erlaubt.
`no-cache:`	Zwingt den Cache, in jedem Fall vor Auslieferung des Eintrags eine Gültigkeitsanfrage an den Server zu stellen oder gar die Daten von dort zu beziehen[6]. Sinnvoll insbesondere bei authentifizierten Anfragen, welche über `public` gecached werden dürfen.
`no-store:`	Daten müssen immer frisch vom Zielserver bezogen werden.
`must-revalidate:`	Dieser Parameter zwingt den Cache die strengen Regeln der Gültigkeitsprüfung in jedem Fall einzuhalten. Ohne diesen Parameter haben Caches einen gewissen Entscheidungsspielraum.

Hat die Auswertung mittels der Gültigkeitsregeln schließlich ergeben, dass ein Cache-Eintrag veraltet ist, wird ein HTTP-Request an den Zielserver gestellt, der entweder eine neue Version des Eintrags liefert oder dem Cache mitteilt, dass trotz den Regelverletzungen das Datum nach wie vor Gültigkeit besitzt[7] (siehe Abb. 5). Exemplarisch sieht eine solche Anfrage wie folgt aus:

```
GET /produkte.html HTTP/1.1 If-Modified-Since: Sun, 30 May 2004
18:30:32 GMT
```

Diese Gültigkeitsanfragen laufen stets unter Verwendung so genannter Validatoren ab. Bei einem Validator handelt es sich um einen Parameter, welcher dem Server eine eindeutige Unterscheidung zwischen verschiedenen Versionen eines Eintrags ermöglicht. In der Praxis wird dafür meist der `Last-Modified`-Parameter (im Beispiel der 30. Mai 2004 18:30:32) oder seit HTTP 1.1. eine eindeutige Versionsnummer, das ETag, verwendet.

Inhalte ohne Validator werden grundsätzlich nicht in einen Cache aufgenommen.

Abb. 5. Validierung von Cache-Einträgen durch den Webserver (passiv)

[6] Das konkreter Verhalten kann im Cache konfiguriert werden.
[7] Die Gültigkeitsfrist wird daraufhin neu gesetzt.

Datensicherheit: Das wahrscheinlich kritischste Problem beim Caching ist die Gefahr, dass persönliche und vertrauliche Daten in öffentlichen Speichern eingesehen werden können. Dem wird jedoch durch verschiedene Sicherheitsmaßnahmen entgegengewirkt. Grundsätzlich sind zwar alle authentifizierten oder verschlüsselten Inhalte von einem Caching ausgeschlossen, durch den `Cache-Control`-Parameter `public` können jedoch auch authentifizierte Seiten (HTTP-passwortgeschützt) gecached werden. Durch eine Verbindung mit dem Parameter `no-cache` ist jedoch gewährleistet, dass der Cache erst die Daten ausliefert, nachdem der Client seinen Authentifizierungsrequest an den Server gestellt hat. Dies gewährleistet die nötige Sicherheit, verzögert jedoch die Datenübertragung, wodurch insbesondere die Abspeicherung unkritischer Inhalte, beispielsweise von Grafiken, in ungeschützten Verzeichnissen empfohlen wird [9].

Anders als authentifizierte Inhalte werden SSL-verschlüsselte Inhalte von einem Cache nicht gespeichert, wodurch hierbei größte Sicherheit gewährleistet bleibt. Bei unverschlüsselten Seitenaufrufen ist jedoch die Gefahr gegeben, dass unseriöse Web-Cache-Administratoren Zugang zu persönlichen Inhalten gewinnen. Dies ist insbesondere bei den in URLs übergebenen Kennwörtern kritisch, allerdings ein allgemeines Sicherheitsproblem und nicht speziell auf Caches zu beziehen.

3 Caching dynamischer Inhalte

Die im vorhergehenden Kapitel vorgestellten Verfahren haben allgemeingültig gezeigt, wie Web-Caching in der Praxis funktioniert. Zur Vereinfachung wurde dabei stets davon ausgegangen, dass es sich bei den Web-Inhalten um klassische Dokumente wie Texte, statische HTML-Dateien oder Grafiken handelt. Im heutigen Internet spielen aber dynamische Inhalte eine entscheidende Rolle. Dabei handelt es sich um Webseiten, die nicht als fester Quellcode auf einem Webserver abgelegt werden, sondern dessen HTML-Code erst beim Aufruf erzeugt wird.

Grund für diese Anwendung sind zum einen interaktive Seiten wie Online-Shops, zum anderen personalisierte Inhalte wie Mitgliedslogins auf Community-Seiten (z.B. Mein eBay). Hier können die Inhalte nur dynamisch und abhängig von den Benutzerangaben erzeugt werden und ändern sich schnell im Laufe der Zeit.

Die Erzeugung dynamischer Inhalte wird durch verschiedene Architekturmodelle unterstützt. Das klassische Prinzip (siehe Abb. 6.) besteht darin, dass anstelle statischer HTML-Dateien Skriptdateien auf dem Webserver (1, 2) aufgerufen werden. Diese werden in einem Prozess der Skriptengine gestartet (3) und führen entsprechende Datenbankanfragen (4) durch. Aus diesen Rückgabewerten erzeugen sie anschließend HTML-Code (5), welcher direkt ohne vorherige Speicherung per HTTP an den Client gesendet wird (6).

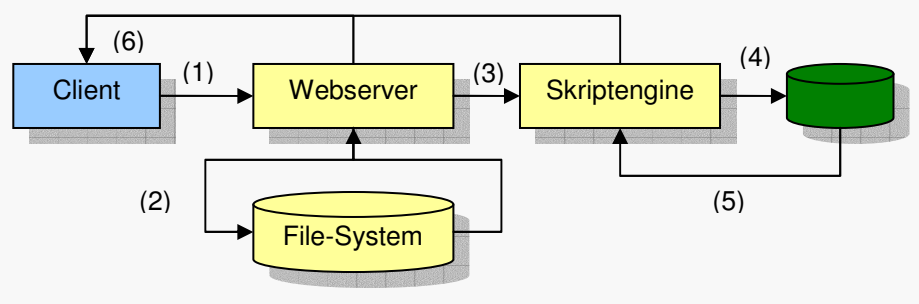


Abb. 6. Aufruf dynamischer Inhalte auf Web- oder Applikationsservern

Durch diese zahlreichen Operationen und Komponentenaufrufe innerhalb des Servers wird dieser bei hohen Anfragezahlen neben der Netzanbindung auch im Hinblick auf seine Rechenkapazität stark belastet.

In diesem Kapitel werden daher Verfahren vorgestellt, wie innerhalb des Webservers, aber auch innerhalb der in Kapitel 2 vorgestellten Architektur effizientes Caching dynamischer Inhalte unterstützt wird.

3.1. Proxy-Caching dynamischer Inhalte

Proxy-Caches sind in der Lage, dynamische Inhalte zu cachen, soweit bei der Bereitstellung durch den Zielserver HTTP-Validatoren mitgesendet werden. Um dies sicherzustellen, müssen daher die generierenden Skripte neben dem HTML-Code auch die nötigen HTTP-Header erzeugen, da dies im Gegensatz zu statischen Webseiten nicht automatisch vom Webserver vorgenommen werden kann.

Die Entscheidung, welche Seiten in einem Cache überhaupt abgelegt werden sollen, kann daher leicht von den Skripten zur Laufzeit entschieden werden und richtet sich nach Art und Zielgruppe des Inhalts.

Dynamische Webseiten lassen sich grob darin unterscheiden, ob die Inhalte aus Gründen der leichten Wartbarkeit nicht statisch programmiert sind oder ob dies erforderlich ist, um personalisierte und interaktive Angebote zu offerieren. Ein Produktkatalog wird beispielsweise von allen Kunden unabhängig von deren Benutzerprofil über einen längeren Zeitraum gleich erscheinen, während die Seite „Meine Käufe“ stets aktuelle und individuelle Informationen darstellt.

Daher ist es im Hinblick auf die Ziele, welche mit Web-Caching erreicht werden sollen, unwirtschaftlich, eine personalisierte Seite, die ohnehin nur von einem einzigen Nutzer weltweit verwendet wird, in einem öffentlichen Proxy-Cache abzulegen.

Im Gegensatz dazu können jedoch erhebliche Leistungssteigerungen erzielt werden, wenn der HTML-Code dynamischer Seiten, wie beispielsweise des Produktkataloges, in einem Cache gespeichert wird. Diese Inhalte sind für jeden Nutzer zugänglich und über einen relativ langen Zeitraum gültig. Eine ständige Neugenerierung und Auslieferung direkt vom Zielserver würde nur zu einer unnötigen Rechen- und Netzbelastung führen.

Neben diesen zum einen streng personalisierten und zum anderen öffentlichen und langlebigen Inhalten gibt es auch solche, die zwar nicht personalisiert sind, dafür aber einer hohen Änderungshäufigkeit unterliegen. Ein Beispiel hierfür sind Aktienkurse oder Suchergebnisse von Auktionsseiten, welche den aktuellen Höchstpreis beinhalten. Auch wenn ein Großteil solch einer Seite aus langlebigen Rahmeninhalten besteht, ändern sich einige Teile der Seite innerhalb kurzer Zeit. Dadurch verliert innerhalb des Caches der gesamte Eintrag seine Gültigkeit, auch wenn nur ein Bruchteil des Inhalts ungültig wird. Die daraus resultierende Neueinlagerung des Inhaltes innerhalb kurzer Zeitabstände stellt daher für Cache, Zielserver und Netzwerk eine nicht wünschenswerte Belastung dar.

Das kritischste Problem des Proxy-Caching dynamischer Inhalte wird jedoch durch die Art der Cache-Einträge verursacht. Vereinfacht gesagt bestehen solche Einträge aus einem HTTP-Request der Form `GET Url` und der entsprechenden Rückgabe. Die Identifikation eines Inhaltes erfolgt dabei ausschließlich über die URL, wobei darin übergebene Variablen ebenfalls berücksichtigt werden[8].

Zur Authentifizierung von Nutzern auf personalisierten Seiten sowie der Speicherung des Sitzungskontextes werden in der Praxis jedoch häufig Header-Parameter wie Cookies und andere, nicht URL-basierte Authentifikationsmechanismen verwendet. Webseiten, die auf dieser Technik beruhen, können daher unter einer einheitlichen URL verschiedene, benutzerspezifische Inhalte bereitstellen, was bei der Verwendung von Web-Caching die Auslieferung falscher Daten zur Folge haben kann [13].

Beispiel (siehe Abb. 7)*:* Client-1 sendet eine Anfrage bestehend aus URL und Cookie (1). Da der Cache unter dieser URL keinen gültigen Eintrag gespeichert hat, sendet er die Anfrage an den Zielserver weiter (2). Dieser liefert aufgrund des Cookies eine persönliche Seite für Client-1

[8] So werden beispielsweise produkte.jsp?art=Schuhe und produkte.jsp?art=Hosen gesondert geführt.

zurück (3). Der Cache legt diese Seite in seinem Speicher ab (da sie durch den HTTP-Header gespeichert werden darf) und leitet das Ergebnis an den Client-1 weiter (4). Client-2 stellt eine Anfrage an dieselbe URL (5), übermittelt aber kein Cookie. Nach Programmierung der Webapplikation sollte dafür eine nicht personalisierte Standardseite ausgeliefert werden. Da diese URL samt Inhalt aber bereits in gültiger Form im Cache vorhanden ist, erhält Client-2 dieselbe Seite wie zuvor Client-1 (6).

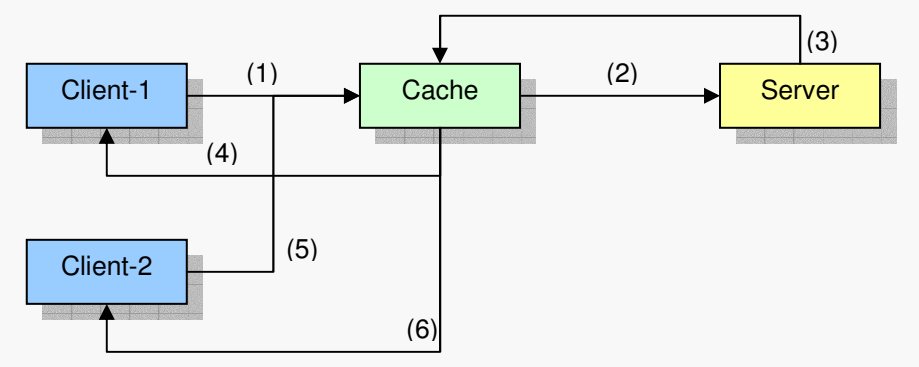


Abb. 7. Probleme beim (herkömmlichen) Caching personalisierter Inhalte

Zusammenfassend kann man daher Proxy-Caches im dynamischen Umfeld nur eingeschränkt verwenden. Dort wo Inhalte öffentlich und langlebig sind, sollten die erzeugenden Skripte entsprechende HTTP-Header generieren, um durch Einlagerung dieser Seiten eine Transfer- und Rechenentlastung zu erzielen. In Fällen, wo kurzlebige oder personalisierte Inhalte eine Rolle spielen, ist die Anwendung herkömmlicher Proxy-Caches jedoch zu vermeiden.

3.2. Serverseitiges Caching

Durch diese eingeschränkte Anwendbarkeit der Proxy-Server können im dynamischen Umfeld die Zielserver nur schwer mit Hilfe der vorgestellten Cache-Hierarchie entlastet werden. Hinzu kommt im Gegensatz zu statischen Seiten ein vermehrter Rechenaufwand durch die ständige Generierung von Seiten. Serverseitiges Caching versucht daher mit internen Optimierungsverfahren, diesen Anforderungen standzuhalten.

Moderne Web- oder Applikationsserver sind meist in einer 2- bis 3-Schichtenarchitektur realisiert. Die unterste Ebene stellen dabei Datenbanksysteme dar, die die Inhalte, aber auch Zusatzinformationen wie Benutzerdaten enthalten. Solche Datenbanksysteme werden in dynamischen Projekten stark gefordert, da sie oft mehr als eine Anfrage pro Seite beantworten müssen. Trotz des allgemein hohen Aktualisierungsgrades von Datenbankinhalten bleiben in typischen Webanwendungen die Ergebnisse vieler SQL-Anfragen über einen bestimmten Zeitraum gleich. Durch das Zwischenspeichern von häufig verwendeten Abfrageergebnissen können daher bereits im Datenbanksystem Leistungsvorteile durch eingesparte Operationen erreicht werden.

Auf den höheren Serverebenen wird aus den Ergebnissen der Datenbankabfragen und anwendungslogischen Berechnungen der HTML-Code generiert. Trotz personalisiertem Inhalt bleibt dieser in vielen Praxisfällen während einer Benutzersitzung nahezu unverändert. Um daher die ständige Neugenerierung auch bei unverändertem Datenbestand zu vermeiden, kann durch Caching der erzeugten Seiten im Sitzungskontext der Bereitstellungsaufwand deutlich reduziert werden. In Abb. 8 ruft ein Client eine personalisierte Seite auf, die abhängig vom Sitzungskontext jeweils anders erscheint. Wurde diese Seite während der Sitzung bereits erzeugt und gespeichert, so kann die Anfrage direkt aus dem serverseitigen Cache beantworten werden ohne eine erneute Seitengenerierung starten zu müssen [10].

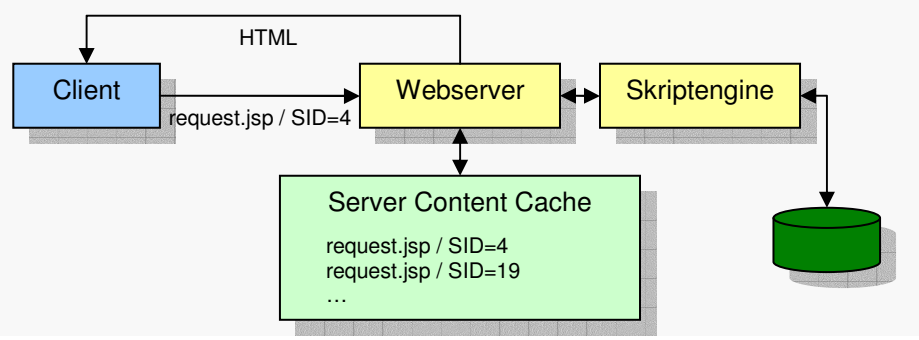


Abb. 8. Serverseitiges Caching personalisierter Inhalte

Dadurch wird im Gegensatz zur Speicherung dynamischer Inhalte in Proxy-Caches stets eine korrekte und individuelle Auslieferung der Inhalte gewährleistet. Die Aktualisierung des Datenbestands erfolgt dabei je nach Gültigkeitsdefinition bei Aufruf (on-demand), bei Änderung der zugrunde liegenden Daten (on-change) oder in periodischen Zeitabständen (periodic) [14].

Durch die Anwendung dieses Konzepts auf die Ebene kleinerer Inhaltsfragmente lässt sich weiterer Rechenaufwand einsparen. Anstatt für jeden Benutzer dessen personalisierte Seiten von Grund auf neu zu erzeugen, bietet es sich an, in einer tieferen Ebene bereits einzelne, nutzerunabhängige Seitenelemente zu erzeugen, welche vielfach wieder verwendet werden können. Dieses Verfahren wird als Fragment-Caching bezeichnet.

Zusammengefasst können also die vorgestellten Möglichkeiten des serverseitigen Caching innerhalb des Zielrechners eine deutliche Leistungssteigerung gewährleisten. Für die Reduzierung der externen Netzbelastung leisten sie allerdings keinen Beitrag, wodurch auf Proxy-Cache-Verfahren trotz deren Nachteilen nicht verzichtet werden kann.

3.3. Edge-Side-Includes-Ansatz (ESI)

Im Abschnitt 3.1 (Proxy-Caching dynamischer Inhalte) wurde das Problem diskutiert, dass in vielen Seiten geringe Inhaltsfragmente einer hohen Änderungshäufigkeit unterliegen, während der Großteil des Seiteninhaltes über einen längeren Zeitraum unverändert bleibt. Ständig müssen daher auch unveränderte Inhalte ersetzt werden, da der Cache nur auf der Granularität eines ganzen Objekts (Seite, Bild, etc.) arbeiten kann. Dies führt zu einer unnötigen Belastung für Proxy- und Zielserver.

Um diesem Phänomen entgegenzuwirken, macht es Sinn, dynamische Webseiten nicht als Ganzes zu betrachten, sondern in Einzelfragmente zu zerlegen. Innerhalb einiger Applikationsserver wird die Nutzung kleiner HTML-Bausteine bereits verwendet, um mehrfache Wiederverwendung über viele Seiten zu ermöglichen.

Ein Ansatz, der dieses Vorgehen auch in externen Cache-Servern anwenden möchte, ist das standardisierte Konzept[9] der Edge-Side-Includes (ESI) [15]. ESI bietet dafür XML-basierte Sprachkonstrukte, um Entwicklern dynamischer Seiten eine Fragmentierung der Inhalte zu ermöglichen.

Ein ESI-Dokument bildet dabei ein Layout, das die zu generierende Seite in einzelne Bestandteile zerlegt. Für jedes dieser Fragmente kann eine eigene Gültigkeitsfrist angeben werden, beziehungsweise entschieden werden, ob dieser Teilinhalt überhaupt gespeichert werden soll (siehe Abb. 9). Dadurch brauchen nur noch die Seitenteile vom Zielserver erneut angefordert zu werden, die zum aktuellen Zeitpunkt nicht mehr gültig oder vorhanden sind. Der restliche Teil der

[9] standardisiert vom W3C im Sommer 2001

Seite kann hingegen nach wie vor aus dem Cache bezogen werden. Aufwändige Kompletterset-zungen entfallen dadurch.

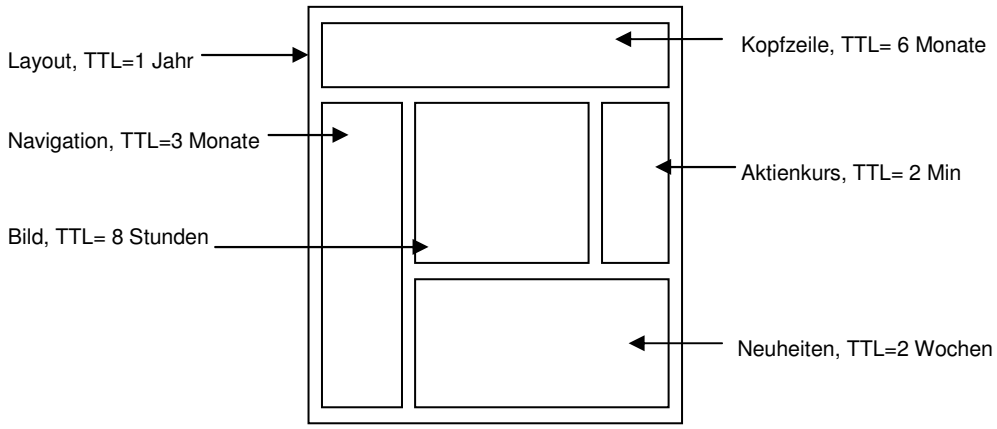


Abb. 9. Fragmentierung einer dynamischen Webseite mittels ESI

Die Verwendung von ESI setzt jedoch neben der Anwendung der Sprachkonstrukte auch die Existenz entsprechender Applikations- und Cache-Server voraus. Diese müssen über unterstützende Softwarekomponenten wie beispielsweise einen ESI-Prozessor verfügen.

Durch die größere Skalierbarkeit aufgrund der Trennung von Inhaltsbereitstellung und Inhaltsgenerierung spielt ESI insbesondere im Umfeld der Content Delivery Networks (CDN) eine Rolle, auch wenn es innerhalb herkömmlicher Cache-Hierarchien unterstützt werden kann. Den Cache-Servern kommt dabei neben den allgemeinen Caching-Diensten die Aufgabe zu, die Layouts mit den einzelnen Inhaltsfragmenten zu füllen und die fertige Seite an den Client zu senden.

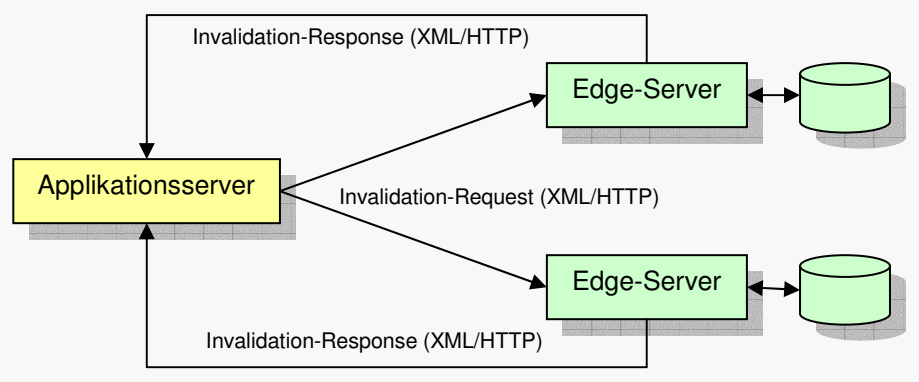
Folgendes Codefragment zeigt die prinzipielle Anwendungsweise von ESI.

```
1  <html><head><title> ESI Beispiel Datei </title><body>
2  <esi:try>
3  <esi:attempt>
4  <esi:include src=”www.site.de/go.htm” onerror="continue"/>
5  </esi:attempt>
6  <esi:except>
7  Alternativer HTML-Code
8  </esi:except>
9  </esi:try>
10 static content
11 </body></html>
```

Das ESI-Tag `try` bereitet den ESI-Prozessor auf das Einfügen eines Fragmentes vor. Als Sub-Tags sind `attempt` und `except` (Ausnahmebehandlung) erlaubt. Das `include`-Tag beauftragt den ESI-Prozessor die angegebene URL als Fragment zu holen. `onerror` schreibt hierbei vor, eine eventuell zurückgegebene Fehlermeldung zu ignorieren und das Parsing fortzusetzen. Tritt ein solcher Fehler auf, gibt die `except`-Anweisung zwischen Zeile 6 und 8 ein alternatives Vorgehen an. Außerhalb des `try`-Blocks kann regulärer HTML-Code oder herkömmlicher Skriptcode integriert werden.

Durch die Verwendung bedingter Anweisungen und der Unterstützung von HTTP-Headervariablen, beispielsweise Cookies, können auch personalisierte Seiten vom Cache korrekt bereitgestellt werden.

Wie auch beim klassischen Web-Caching wird die Gültigkeitsdauer der einzelnen Fragmente über entsprechende Angaben im HTTP-Header definiert. Da die genaue Vorhersage der `Expires`-Werte jedoch oft schwierig ist, bietet die ESI-Spezifikation neben den Sprachkonstrukten zur Seitenerstellung auch ein Konzept zur Interaktion zwischen Applikations- und Edge-Servern an. Dadurch kann die Ungültigkeit eines Eintrags umgehend publiziert werden. Der Applikationsserver sendet hierzu eine XML-Nachricht, die die zu entfernenden Einträge aufführt, per HTTP an die entsprechenden Caches. Konnte der Invalidierungsauftrag für alle Einträge erfolgreich durchgeführt werden, wird dies durch ein Änderungsprotokoll im XML-Format quittiert. Andernfalls wird eine Fehlerbeschreibung übersandt.



Durch die Integration des ESI-Konzeptes in JavaServerPages (JSP) mittels des ESI for Java (JESI) ist dessen Anwendung insbesondere bei der Webseiten-Entwicklung auf Basis der Java-Technologie interessant. JESI generiert innerhalb des HTML-Outputs automatisch die nötigen ESI-Tags und HTTP-Header, um ein Speichern der Fragmente in ESI-unterstützenden Caches zu ermöglichen. Weiterhin wird durch JESI die partielle Ausführung einer JSP ermöglicht, um bei Anfrage nur die geforderte Fragmente generieren zu müssen [16, 19].

ESI bietet also eine zuverlässige und sicherere Möglichkeit, dynamische Inhalte auch außerhalb der Web- oder Applikationsserver zu cachen und dadurch zu einer Entlastung der Zielserver beizutragen. Die Tags können dabei in jegliche Skripte integriert werden; ESI ist selbst keine Skriptsprache.

Durch die Existenz ESI-fähiger (Edge-)Server an vielen wichtigen Knoten des Internets ist eine schnellere Auslieferung der Inhalte an die Nutzer möglich, wodurch zusammenfassend beide Ziele des Web-Caching erreicht werden können.

ESI ist allerdings in der derzeitigen Praxis nur begrenzt einsetzbar, da nur spezielle Cache- und Applikationsserver die Auslieferung und das Zusammensetzen solcher Seiten unterstützen. Da weiterhin der gesamte Web-Auftritt unter Verwendung der ESI-Sprachkonstrukte von Grund auf neu zu gestalten ist, zerstört im Gegensatz zum klassischen Web-Caching der ESI-Ansatz die bisher wünschenswerte Transparenz.

3.4. Motivation und Beispiel neuerer Konzepte

Auch wenn in der aktuellen Praxis das ESI-Konzept zunehmend an Bedeutung gewinnt, stellt es noch keine Optimallösung dar und bedarf Verbesserungen. Neben der mangelnden Transparenz im Hinblick auf die freie Seitengestaltung wird der Aufbau auf statischen Layouts kritisiert [13].

Das ESI-Dokument, somit das Layout der personalisierten Seiten, wird ebenso wie mögliche Fragmente in die Caches eingelagert. Ist es zum Zeitpunkt des Aufrufs gültig, so werden lediglich nach Bedarf die ungültigen Fragmente neu eingelagert. Viele Webseiten unterstützen heute aber neben personalisierten Inhalten auch personalisierte Layouts. Da eine variablenbasierte

Personalisierung mittels ESI zwar innerhalb eines Layouts möglich ist, dieses selbst allerdings nur über die URL identifiziert wird, kann dieser Sachverhalt nicht berücksichtigt werden, solange eine gültige Layoutversion im Cache vorhanden ist.

Ein weiterer Kritikpunkt ist die Tatsache, dass sich nicht alle Seiten in unabhängig-langlebige Fragmente zerlegen lassen. Wird beispielsweise in einem großen Onlineshop dem authentifizieren Nutzer neben einer persönlichen Begrüßung die Auflistung seines Warenkorbinhaltes präsentiert, so wären dies inhaltlich zwar zwei verschiedene Fragmente, doch beide sind abhängig vom Profil des Nutzers. Da diese Inhalte in der Regel keine Lebensdauer besitzen, müssten beide daher permanent beim Zielserver angefordert werden. Das ESI-Konzept bietet sich in der derzeitigen Form also eher für Seiten an, die ein statisches Layout und unabhängige Fragmente besitzen.

Der Ansatz des dynamischen Proxy-basierten Caching [13] versucht daher, eine Lösung auf Basis erweiterter Proxy-Caches zu finden, welcher die Fehler bisheriger Ansätze eliminieren soll.

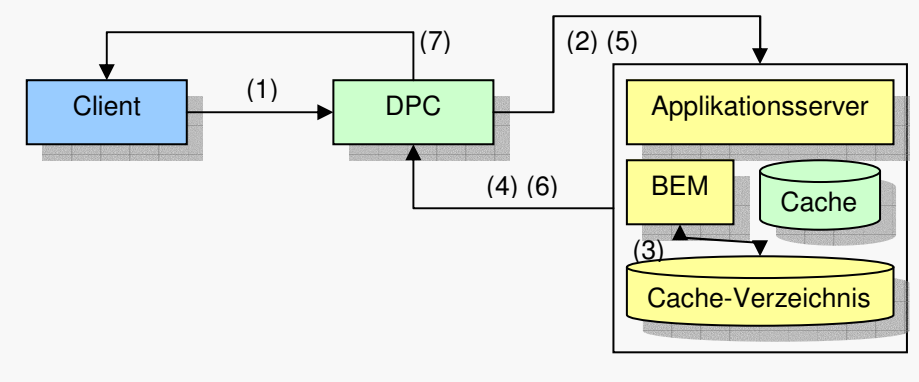


Abb. 10. Architektur des Dynamic-Proxy-Caching

Die Architektur dieses Ansatzes (siehe Abb. 10) besteht aus einem dynamischen Proxy-Cache (DPC) und einem Back-End-Monitor (BEM) auf dem Zielserver. Der DPC hat die Aufgabe, analog zum ESI-Ansatz Layouts mit einzelnen Fragmenten zu füllen und diese an die Clients zu liefern (7). Anders als bei ESI wird jedoch jede Anfrage (1) zunächst an den Zielserver beziehungsweise den BEM durchgereicht (2). Das dort ausgeführte Skript generiert ein stets individuelles Layout ohne Inhalt und gibt dieses an den DPC zurück (4).

Während der Abarbeitung des Skripts prüft der BEM durch Anfrage in seinem Cache-Verzeichnis (3), ob die zugehörigen Fragmente bereits an die DPC ausgeliefert wurden oder noch gültig sind. Wird die Anfrage negativ beantwortet, schreibt das Skript in die entsprechenden Stellen des generierten Layouts eine SET-Instruktion, welche den DPC beauftragt, sich das entsprechende Fragment vom Zielserver anzufordern (5, 6). Andernfalls erscheint eine Instruktion, die den DPC das Fragment aus seinem Cache laden lässt.

Neben der Verwaltung der Fragmente bildet der BEM einen serverseitigen Fragmentcache um immer gültige Versionen ohne erneuten Rechenaufwand ausliefern zu können. Das Konzept verbindet daher die Stärken von Proxy- und Server-Caching und versucht deren jeweiligen Nachteile zu reduzieren.

Einen erheblichen Schwachpunkt stellt allerdings der Ausfall des Zielrechners dar. In diesem Fall wären keine Replikate vorhanden, die die Verfügbarkeit der Inhalte auch nur temporär gewährleisten könnten[10].

[10] Diese Anmerkung basiert auf eigenen Überlegungen.

4 Kommerzielle Caching-Lösungen

Dort, wo große Internetapplikationen laufen, sind aus Sicherheits- und Leistungsgründen meist keine einfachen Webserver und Skriptengines, sondern moderne Applikationsserver großer Hersteller im Einsatz.

Aufgrund der Bedeutung des Themas bieten diese daher innerhalb ihrer Systeme oder als zusätzliche Produkte Caching-Lösungen an, um eine hohe Verfügbarkeit der angeforderten Inhalte zu ermöglichen. Im Folgenden werden daher eine serverseitige, integrierte Lösung von IBM, sowie ein externer Proxy-Cache von Oracle kurz vorgestellt.

4.1. Serverseitiges Caching in IBM WebSphere

Der IBM WebSphere Application Server [17] umfasst einen integrierten Cache für dynamische Inhalte, der sich nahtlos über die Administrationsschnittstelle konfigurieren und in bestehende Netzwerke auf Basis des WebSphere Edge Server oder des IBM HTTP-Server integrieren lässt.

Durch diese vollständige Integration in die IBM-Produktfamilie ermöglicht dieser Caching Service[11] die Speicherung auf unterschiedlichen Ebenen und Granularitäten innerhalb und außerhalb des Applikationsservers (siehe Abb. 11).

Zur Replikation der Inhalte zwischen den einzelnen Servern sowie zur Invalidierung der Einträge verwendet der Caching Service den Java Message Service (JMS), welcher eine effiziente Inhaltsverwaltung innerhalb des gesamten Netzwerks ermöglicht. Die Invalidierung erfolgt dabei nach Regeln oder Zeitperioden, kann aber auch explizit programmiert werden.

Die Cache-Komponenten innerhalb des Applikationsservers sind als hauptspeicherbasierte Systeme realisiert, wodurch schnelle Antwortzeiten möglich sind. Je nach Granularität erfolgt das Caching in den unterschiedlichen Schichten des Servers.

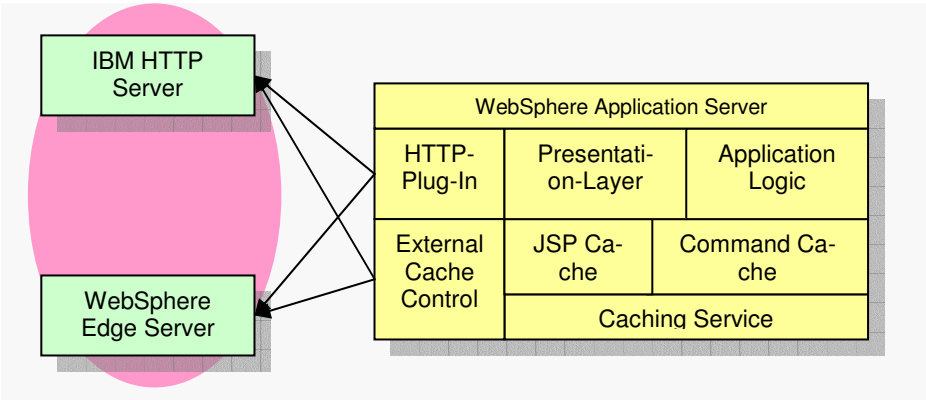


Abb. 11. Wichtige Caching-Komponenten von IBM WebSphere

JSP/Servlet Result Cache: Innerhalb der Präsentationsschicht ist das System in der Lage, Ausgaben von Servlets oder JSPs im Kontext der Anfrageparameter und Sitzungsinformationen zu cachen. Nur wenn eine Anfrage nicht aus diesem Cache bedient werden kann, wird das JSP/Servlet gestartet und das Ergebnis neu generiert. Da sich komplette JSP-Seiten oft aus einzelnen Fragmenten zusammensetzen, unterstützt dieses System auch Fragment-Caching.

Durch das integrierte Zusammenspiel mit anderen Netzkomponenten kann der Caching Service parallel dazu diese Fragmente und Seiten in externe Caches replizieren, um die Verfügbarkeit

[11] vollständig: WebSphere Application Server Dynamic Cache Service.

näher am Client zu gewährleisten. Eine spezielle Programmierung der Skripte ist zur Nutzung des JSP-Caching dabei nicht nötig.

Command Cache: Auf der Ebene der Anwendungslogik kann ein weiterer Cache Daten speichern, die von Anfragen auf externen Systemen wie Datenbanken oder Remoteservern stammen oder die eine intensive Berechnung erfordern.

Wird entsprechende Anwendungslogik innerhalb der WebSphere Command Framework API entwickelt, ist der Command Cache in der Lage, die entsprechenden Ergebnisse einzelner Execute-Methoden[12] zu speichern. Das konkrete Verhalten dieses Caches lässt sich mittels eines XML-Cache-Policy-Dokuments genau definieren.

External Cache: Durch Installation des WebSphere HTTP-Plug-In kann der Application Server die Konzepte der ESI-Spezifikation implementieren und mit externen Servern wie dem Edge Server oder dem IBM HTTP-Server auf deren Kommunikationsansätzen interagieren. Parallel dazu kann auch durch Einsatz des External Cache Control die Einlagerung und Zusammensetzung der Inhalte sowie deren Gültigkeitskontrolle im gesamten Serververbund koordiniert und die Bereitstellung an den Client optimiert werden.

4.2. Reverse Proxy Caching mit Oracle Web Cache

Oracle Web Cache [16] ist ein speziell auf dynamischen Inhalt zugeschnittener Proxy-Cache. Anwendung findet er insbesondere als Reverse Proxy, aber auch innerhalb von CDNs.

Oracle Web Cache implementierte eine Reihe von Lösungsansätzen für die typischen Probleme im dynamischen Umfeld: Eindeutigkeit der Cacheeinträge, Verwaltung von Sitzungskontexten, die damit verbundenen Aspekte der Personalisierung, Datenkonsistenz und Fragmentverwaltung.

Um beispielsweise die Eindeutigkeit der Cacheeinträge zu gewährleisten, setzt sich im Oracle Web Cache der Eintragsschlüssel aus URL und einem individuellen HTTP-Header zusammen. Dadurch ist es im Gegensatz zu herkömmlichen Proxy-Caches möglich, zu einer URL je nach Anfrage verschiedene Versionen auszuliefern. Die Auswahl entsprechender Auslieferungen wird dabei über die Werte oder die Existenz spezieller Header bestimmt. Dies ist insbesondere beim Caching von Seiten innerhalb einer Benutzersitzung interessant.

Für die Bereitstellung und Zusammensetzung möglichst vieler Inhalte direkt aus dem Cache und einer dafür effizienten Organisation implementiert Oracle Web Cache die oben beschriebenen Ansätze der Edge-Side-Includes (ESI).

Die Kommunikation mit anderen Servern und Applikationen erfolgt ausschließlich per HTTP.

Insbesondere ist die Lösung so ausgelegt, dass neben herkömmlichen Web-Inhalten auch Applikationsdaten wie SOAP-Nachrichten verwaltet werden können.

5 Neuere Aspekte des Web-Caching

Web-Caching ist ein sehr weitgefächertes Thema angefangen von Netzwerkstrukturen und Netzprotokollen über Algorithmen bis hin zu Sprachkonstrukten.

In meiner Arbeit habe ich mich daher bewusst auf die grundsätzliche Ansätzen des klassischen und dynamischen Caching beschränkt und dabei einzelne Protokoll- oder Sprachkonstrukte vorgestellt. Der Fokus sollte auf die Themen gesetzt werden, die dem Verständnis und der Anwendung von Web-Caching-Technologien hilfreich sind.

Dieses letzte Kapitel möchte ich daher nutzen, um kurz Einblicke in neuere Fragestellungen hinsichtlich Web-Caching zu gewähren.

---

[12] Methoden innerhalb einer Applikation, welche Daten verändern oder neu generieren.

5.1. Aktives Caching und Prefetching

In Kapitel 2 wurde der typische Ablauf einer Cache-Einlagerung dargestellt. Der entsprechende Cache-Server kontrolliert, ob er einen ankommenden Request beantworten kann und fordert, soweit dies nicht gewährleistet ist, frische Inhalte beim Zielserver an. Diese legt er in seinem Speicher ab. Die Inhalte der Caches werden daher dynamisch durch das Aufrufverhalten der Endbenutzer bestimmt. Diese Einlagerungsart wird daher als anziehendes (pull), passives Caching (siehe Abb. 12 links) bezeichnet.

Der Vorteil dieses Ansatzes besteht darin, dass immer nur die auch geforderten Inhalte in einen Cache eingelagert werden und keine „uninteressanten" Daten Speicherplatz belegen. Der Nachteil jedoch ist, dass im Falle eines ungültigen Eintrags der Web-Cache erst einen Anfrage an den Zielserver leiten muss, bevor er die gewünschten Inhalte dem Client bereitstellen kann. Bei einer durchschnittlichen Hitrate von 50% müsste also immer noch jede zweite Anfrage direkt vom Zielserver beantwortet werden, was sicherlich nicht der gewünschten Ideallösung entspricht.

Um dieses Problem zu lösen, muss der Cache also stets im Besitz einer gültigen Version sein, ohne sie erst bei Bedarf anfragen zu müssen.

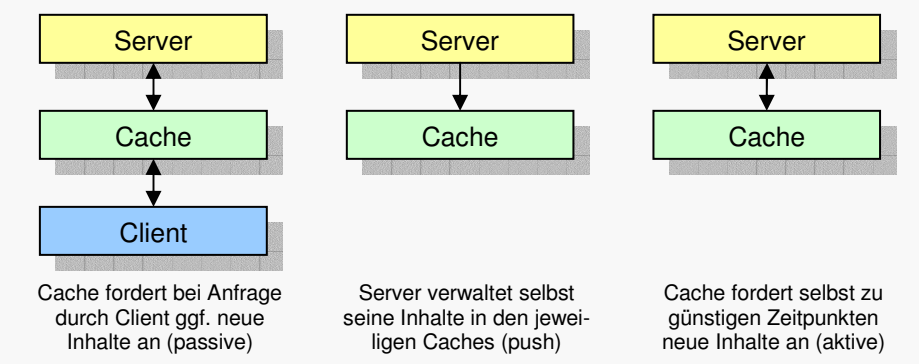


Abb. 12. Strategien zur Einlagerung gültiger Inhalte

In kommerziellen CDNs wird dies durch die koordinierte Replikation und Invalidierung von Inhalten zwischen Applikationsserver und den verteilten Edge-Servern erreicht. Die Inhalte der Caches werden zentral durch den Zielserver verwaltet (siehe Abb. 12 Mitte). Die Applikation schiebt stets die jeweils frischen Inhalte in alle angeschlossenen Caches (push), wodurch es nie zu einer Rückfrage am Server wegen Gültigkeitsüberschreitung kommen muss[13]. Dieser Ansatz lässt sich jedoch nicht auf offene Architekturen übertragen, da hier die Inhalte der Caches über das Zugriffsverhalten verteilter Endnutzer bestimmt werden.

Um dennoch auch hier die Anfragen an den Zielservern stärker reduzieren zu können, gibt es den Ansatz des aktiven Caching (siehe Abb. 12 rechts). Der Cache führt auf seinem Datenbestand Buch über die Häufigkeit von Änderungen sowie das Zugriffsverhalten auf einzelne Inhalte. Daraus errechnet er optimale Zeiträume innerhalb von Leerlaufperioden, in denen er sich die aktualisierten Inhalte selbst anfordert. Für den im Cache befindlichen Inhalt kann dadurch eine Rückfrage bei den Zielservern eingespart werden, was mit deutlichen Leistungssteigerungen einhergeht. Während bei passiven Web-Caches die Hitrate im Schnitt bei rund 35% liegt, beträgt sie in einigen aktiven Caching-Systemen über 70% [21].

In der Industrie ist dieser Ansatz daher weit verbreitet, weshalb viele Hersteller entsprechende Lösungen anbieten. Da es jedoch für diese Algorithmen noch keinen Standard gibt, variieren die Implementierungen und Leistungen erheblich.

[13] Lediglich nicht-cachebare Inhalte müssen nachgeliefert werden.

Neben Web-Caching in unterschiedlichen Ausprägungen gibt es in der Praxis auch den Ansatz des Prefetching, um die Latenzzeit der Nutzer zu reduzieren. Der grobe Ablauf besteht darin, dass beim Zugriff auf eine Seite häufig von dieser aus nachfolgend geforderte Inhalte mitgeladen werden, um zum Zeitpunkt der eigentlichen Anfrage diese bereits im Client zu haben. Lädt beispielsweise ein Nutzer die Startseite eines Online-Shops und werden erfahrungsgemäß im Anschluss daran die Seiten „Computer“, „Hi-Fi“ und „TV“ angefordert, so überträgt das System automatisch in der Zeit, in der sich der Nutzer die Startseite ansieht und sein weiteres Navigieren abwägt (Leerlaufzeit) diese Inhalte zum Client. Klickt der Nutzer anschließend auf eine weiterführende Seite ist sie bereits vorhanden und kann angezeigt werden. Der Nachladevorgang wird für diese Seite dann erneut angestoßen.

Um diese Funktionalität zuverlässig zu offerieren, verwendet Prefetching statistische Daten über das Zugriffsverhalten der Nutzer. Der Ansatz lässt sich analog zu Caching auf unterschiedlichen Netzebenen betreiben, wie nachstehend kurz skizziert wird:

In der clientbasierten Implementierung wird die Browser-Historie analysiert, woraus Querverbindungen zwischen einzelnen Inhalten erkennbar werden. Durch die Häufigkeit einzelner Seitenaufrufe kann daraus das Prefetching-System die individuelle Wichtigkeit für einzelne Inhalte ermitteln und demnach ein entsprechendes Laden von verknüpften Seiten beim Aufruf einer gewissen Seite vornehmen.

In Proxy- aber auch Zielservern kann Prefetching dahingehend betrieben werden, dass bei der Anforderung von Inhalten die erfahrungsgemäß häufig anschließend geladenen Inhalte gleich mit an den Client gesendet werden. Um die dadurch jedoch großen Auslieferungsmengen zu optimieren, bietet es sich an, diesen Ansatz mit Caching zu kombinieren.

5.2. Caching von Webservices

Bei den bisherigen Betrachtungen in dieser Arbeit lag der Fokus ausschließlich bei Inhalten, welche mittels eines Webbrowser dargestellt werden konnten. Die vorgestellten Ansätze beschrieben, wie typische Webobjekte, also Bilder, Flashanimationen, HTML-Seiten u.ä. ideal zwischengespeichert werden können, um eine Netzentlastung und schnelle Auslieferungen an die Endnutzer zu erzielen.

In der heutigen Praxis spielt aber eine weitere Art von Inhalt eine wichtige Rolle im Web. Innerhalb der Business-to-Business-Welt laufen Geschäftsprozesse längst nicht mehr nur durch manuelle Bestellung mittels Onlineshops, sondern durch die Verwendung von Webservices ab. In diesem Abschnitt werden daher kurz Ansätze für das Caching dieses Inhaltstyps vorgestellt.

Webservices sind eine Art der extern aufgerufenen Methoden. Oft wird auch von einem XML-basierten Remote-Procedure-Call gesprochen. Durch die Integration solcher Webservices in Anwendungen ist das Kriterium „Antwortzeit“ noch wichtiger als im herkömmlichen Webumfeld. Während 10 Sekunden Ladezeit für eine sichtbare Webseite beim Client noch als akzeptabel gilt, sollte ein Webservice innerhalb weniger als 3 Sekunden abgearbeitet sein [25]. Dabei setzt sich diese Zeit aus drei Teilen zusammen: Der Zeit der Datenübertragung, der des Parsens beziehungsweise des Codieren von XML und der für die eigentliche Ausführung der Methode.

Um eine effiziente und unternehmensübergreifende Interaktion auf Basis von Webservices zu unterstützen, wird daher zunehmend Caching als unterstützende Technik diskutiert. Allerdings gibt es mehrere Probleme, die das Caching in einer herkömmlichen Weise nicht erlauben und neue Ansätze verlangen. Der Grund dafür liegt darin, dass die Identifikation bei herkömmlichen (dynamischen) Webinhalten mittels einer URL und ggf. der HTTP-Header-Parameter erfolgt. Da sich bei Webservices aber die Parameter innerhalb des XML-Körpers befinden, ist dieser Ansatz nicht geeignet, Webservices zuverlässig zu cachen.

Viele Ansätze des SOAP[14]-Caching erfolgen daher auf Clientseite, da hier entsprechende Softwarekomponenten vorhanden sind, die XML-Nachrichten der Webservices zu verstehen oder

[14] SOAP ist ein Protokoll zum Methoden-Aufruf von Webservices

zu transformieren. Primitive Ansätze implementieren dabei einen Cache als Klasse aufbauend auf einer Hash-Table und verwenden als Schlüssel den Endpoint und die übergebenen Parameter [20, 25].

Ein aktueller Ansatz von IBM [24] versucht, einen Cache für Webservices innerhalb der clientseitigen Middleware zu integrieren, wodurch hinsichtlich der Antwortzeit erste Leistungsverbesserungen möglich sind.

Wie auch beim herkömmlichen Web-Caching lassen sich die Antwortnachrichten in solche unterscheiden, die in einem Cache abgelegt werden dürfen, und solche, für die dies nicht möglich ist. Als Schwierigkeit hat sich hierbei erwiesen, dass noch kein Standard existiert, welcher diese Entscheidung seitens des Servers an die Clients übermitteln kann. Derzeit müssen also Konfigurationen des Caches entsprechende Unterscheidungen treffen.

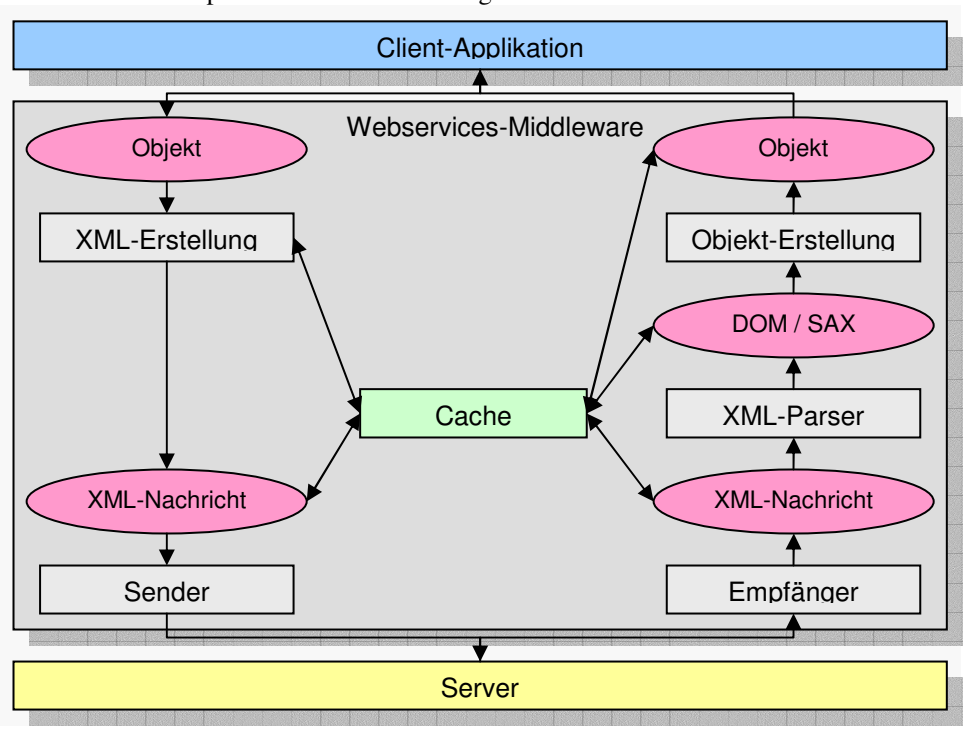


Abb. 13. Integration eines Caches für Webservices

Grundsätzlich lassen sich reine Anfrageoperationen cachen, während Operationen mit Änderungsfunktionalität nicht gespeichert werden. Für diese speicherbaren Antwortnachrichten muss daher analog zum klassischen Caching eine Gültigkeitsfrist existieren. Auch hier existiert noch kein Standard, der serverseitiges Konsistenzmanagement unterstützt. Abhängig von der Semantik eines Webservices muss daher der TTL vom Cache-Administrator geschätzt werden oder unter Verwendung des HTTP 1.1. Protokolls erfolgen. Aufgrund der eigentlichen Unabhängigkeit von SOAP und HTTP ist diese Verwendung allerdings vorsichtig zu genießen.

Die entscheidende Idee dieses Ansatzes von IBM ist, Caching auf unterschiedlichen Repräsentationsebenen abhängig vom Typ der übermittelten Objekte durchzuführen. Grund dafür sind Abwägungen von Leistungsverbesserungspotential, technischer Machbarkeit und Sicherheit hinsichtlich von Seiteneffekten.

Die Webservice-Caching-Architektur dieses Ansatzes lässt sich daher wie folgt skiziieren (siehe Abb. 13): Eine Clientanwendung ruft ein Objekt auf, welches einen Webservice repräsentiert. Das Objekt wird zu einer XML-Nachricht transformiert, aus der sich ein Cache-Schlüssel (analog zur URL in herkömmlichen Web-Caches) ableiten lässt. Die Nachricht wird an den Server gesendet und eine XML-Antwortnachricht empfangen. Diese kann im Cache unter dem zuvor

generierten Schlüssel abgelegt werden. Das XML-Dokument wird – je nach Implementierung – mittels DOM- oder SAX-Parser analysiert, wodurch entsprechende Objekte entstehen. Auch diese können im Cache abgelegt werden. Schließlich wird durch Transformation ein Anwendungsobjekt gewonnen, welches ebenfalls im Cache abgespeichert werden kann.

Bei einem nachfolgenden Request wird im Falle eines Hit, der entsprechender Cache-Eintrag in entsprechender Repräsentationsform zur Verfügung gestellt. Die Aufbereitung zu einem Anwendungsobjekt erfolgt im Anschluss automatisch.

Die Entscheidung, ob eine Serverantwort als XML-Dokument, als SAX- (oder DOM-) Objekt oder als Anwendungsobjekt im Cache abgelegt wird, richtet sich nach dem Typ des Rückgabewertes.

Die Speicherung als XML-Dokument birgt den Nachteil, dass bei jedem Aufruf das Dokument geparst und in ein Anwendungsobjekt umgewandelt werden muss. Der Vorteil besteht darin, dass auch nach Modifikation des generierten Anwendungsobjektes die Serverantwort aus dem Cache erneut gewonnen werden kann. Diese Art der Speicherung bietet sich daher für alle Objekttypen an.

Auf der nächsthöheren Ebene ist eine Speicherung als SAX- oder DOM-Objekt möglich. Bei Nutzung eines solchen Cache-Eintrages ist ein Parsen des XML-Dokuments nicht mehr nötig, allerdings die Transformation in ein Anwendungsobjekt. Auch diese Art des Caching eignet sich für alle Objekttypen und bietet darüber hinaus den gleichen Vorteil wie das Speichern der ursprünglichen XML-Nachricht.

Im Hinblick auf die Bereitstellungskosten ist das Caching eines fertig generierten Anwendungsobjektes die optimale Lösung. Es bedarf keines Parsens oder Transformierens und kann daher sehr schnell erfolgen. Der Nachteil besteht darin, dass bestimmte Kopierverfahren zum Speichern gewählt werden müssen, um eine Modifikation durch den parallelen Zugriff verschiedener Anwendungen zu vermeiden. Die einfache Rückgabe einer Referenz durch den Cache (call-by-reference) scheidet somit aus, soweit es sich nicht um Objekte handelt, die nur lesenden Zugriff unterstützen.

In diesem Konzept werden daher drei verschiedene Arten des Objekt-Caching vorgeschlagen. Die erste Möglichkeit besteht im Abspeichern der Serialisierung eines Objektes. Hierunter versteht sich die Speicherung der gesamten Objektstruktur inklusive verbundener Referenzobjekte (ggf. rekursiv) innerhalb eines Byte-Arrays. Bei Bereitstellung durch den Cache wird ein Objekt aus diesen Informationen generiert und ausgeliefert. Der Nachteil dieses Ansatzes ist der Aufwand für die Serialisierungs- und Rückgewinnungsoperationen sowie die Einschränkung, dass sich nicht alle Objekte serialisieren lassen.

Als zweite Möglichkeit bietet sich die Erstellung einer neuen Objektinstanz an, welche alle Attributwerte des gespeicherten Objektes einkopiert bekommt. Dies ist in der Regel allerdings nur für so genannte Bean-Objekte (Attributspeicher) und Arrays möglich. Für selbstentwickelte, anwendungsspezifische Objektstrukturen ist dieses Verfahren allerdings nur schwer einsetzbar, da es eine eigene Implementierung der Kopierfunktionalität erwartet.

Die dritte Option ist die Verwendung der `clone`-Methode, welche standardmäßig von allen Objekten unterstützt wird. Die internen Attribute eines Objektes werden dabei in eine neue Instanz kopiert, allerdings bleiben Beziehungen zu anderen Objekten als Referenzen bestehen, wodurch dieser Ansatz Seiteneffekte nur ausschließt, wenn es sich bei den referenzierten Objekten um unveränderliche Objekte handelt. Andernfalls ist ein rekursives Clonen notwendig.

Zusammenfassend kann man diesen Ansatz zum Caching von Webservices innerhalb einer Middleware-Architektur als weit fortgeschritten betrachten. Die Auswahl der geeigneten Speicherungsart innerhalb des Caches ermöglicht die stets effizienteste Bereitstellung bereits empfangener Service-Nachrichten.

5.3. Verbleibende Herausforderungen

Die Gründe für die ursprüngliche Erforschung des Gebietes „Web-Caching“ waren der Wunsch nach geringerer Latenzzeit und die Entlastung von Servern und Netzwerken. Durch die ausgereiften Konzepte des klassischen Web-Cachings wurde hierfür bisher ein effektiver Beitrag geleistet. Mit der Standardisierung des Konzeptes der Edge-Side-Includes sind in den letzten Jahren auch erhebliche Leistungssteigerung für dynamische Web-Inhalte erzielt worden.

Die Motivation, die Forschung im Bereich „Web-Caching“ weiterhin voranzutreiben, besteht in den Schwachstellen der bisher hilfreichen, aber längst nicht optimalen Ansätze.

Wie beispielsweise in Abschnitt 3.4. motiviert wurde, sind Modifikationen von ESI nötig, um auch dynamische Layouts unterstützen zu können.

Auch im Bereich des aktiven Web-Caching und des Prefetching, welche durch intelligente Bereitstellung gültiger Inhalte die Hitraten erfahrungsgemäß auf 70% steigern können, besteht insbesondere Standardisierungsbedarf, um bereits etablierte Konzepte weitläufig zugänglich zu machen.

Letztlich bedarf es beim Caching von Webservices einer Erweiterung von SOAP und WSDL, um Cachekonsistenz zu gewährleisten und Klarheit bezüglich der Eignung für Caching zu bringen. Solche Verbesserungen könnten zu einer Weiterentwicklung bestehender Ansätze, auch im Hinblick auf Webservice-Caching in Proxy-Servern, beitragen [26].

6 Zusammenfassung

Ohne die Existenz von Web-Caches an vielen wichtigen Netzwerkknoten würde das Internet in der heutigen Form vermutlich nicht existieren. Server, insbesondere die von viel besuchten Seiten, würden unter der hohen Anfrage- und damit verbundenen Rechenlast zusammenbrechen, und die Antwortzeiten wären um ein Vielfaches höher.

Durch die Anwendung von Web-Caching können der Netztransfer und die Anfragen an die eigentlichen Zielserver deutlich reduziert werden, wodurch die Funktionalität des Webs auch bei immer weiter wachsender Nutzerzahl und Informationsmenge gewährleisten bleibt.

Die Idee und Funktionsweise eines Web-Caches und dessen hierarchische Anordnung innerhalb existierender Netzwerke ermöglichen, dass eine hohe Anzahl von wichtigen Informationen meist in schnellen Speichern nahe dem Client vorhanden ist. Dadurch werden Antwortzeiten und Netztransfer gleichermaßen eingespart.

Die Frage, ob im Cache befindliche Inhalte mit denen auf den Zielservern liegenden Quelldaten zum Zeitpunkt der Anforderung noch übereinstimmen, gilt als das qualitative Hauptproblem des Caching. Konstrukte des HTTP-Protokolls wie `Expires`, `Last-modified` und `Not-modified` gewährleisten hier eine schnelle Überprüfung und eventuelle Neuanforderung der Inhalte.

Die Bedeutung von dynamischen Seiten im heutigen Internet stellt das klassische Web-Caching vor neue Probleme und Herausforderungen. Aufgrund der ausschließlichen Identifikation von Inhalten über die URL in herkömmlichen Proxy-Caches ist eine einfache Anwendung dieser Ansätze im Hinblick auf personalisierte Inhalte unter einer gemeinsamen Adresse nicht möglich.

Um das Caching von dynamischen Inhalten dennoch zu gewährleisten, existieren Konzepte des serverseitigen Caching sowie der Edge-Side-Includes (ESI).

Serverseitiges Caching ermöglicht die Zwischenspeicherung von häufig geforderten Inhaltsfragmenten oder gar ganzen Seiten im eigentlichen Applikationsserver, um diese bei Anfrage ohne vorherige Neugenerierung direkt ausliefern zu können. Dadurch kann die Rechenbelastung auch bei dynamischen Seiten deutlich reduziert werden.

Der ESI-Ansatz hingegen ermöglicht ein effizientes Caching auch in der Architektur der Proxy-Caches. Die zugrunde liegende Idee besteht darin, dass dynamische Seiten aufgrund unter-

schiedlicher Gültigkeitszeiten in Fragmente zergliedert werden, welche erst beim Aufruf zu einer fertigen Seite zusammengesetzt werden. Durch die Verwendung von HTTP-Variablen kann dieser Ansatz darüber hinaus auch das Caching personalisierter Seiten durch erweiterte Identifikationsmechanismen unterstützen.

Da diese zwischenzeitlich weit verbreiteten und hilfreichen Ansätze noch keine Ideallösungen darstellen, werden nach wie vor neue Ansätze und Fragestellungen diskutiert. Insbesondere im Bereich des Caching von Webservices und dem aktiven Web-Caching, welches eine weitere Leistungssteigerung mit sich bringen kann, existiert Forschungs- und Standardisierungsbedarf.

7 Referenzen

Nachstehend ein Überblick über verwendete Quellen. Stand aller Onlinequellen: 19.06.2004.

1. Google Deutschland: Startseite.
   http://www.google.de
2. Denic: RIPE-Hostcount.
   http://www.denic.de/de/domains/statistiken/hostentwicklung/hostcount.html
3. forsa: Anzahl Internetnutzer in Deutschland, August 2003.
   http://www.digitale-chancen.de/transfer/downloads/MD612.pdf
4. eBay Deutschland: Mediadaten.
   http://pages.ebay.de/community/aboutebay/contact/mediadaten.html
5. 1&1 Internet AG: Das 1&1 Hochleistungs-Rechenzentrum.
   http://hosting.1und1.de/xml/order/DataCenter
6. Wendel, U.: Application Based Caching.
   http://caching.ulf-wendel.de/
7. Caching.com: Caching 101, General Info.
   http://www.caching.com/caching101.htm
8. Fischer, A.: Proxy, Grundlagen, Konzepte, Beispiele.
   http://www.htw-saarland.de/fb/gis/people/wpauly/vortraege_internet/ws99_00/Proxys/index.html
9. Web-Caching.com: Caching Tutorial for Web Authors and Webmasters.
   http://www.web-caching.com/mnot_tutorial/intro.htm
10. Mulzer, M.: Using server-side caching to increase web site performance and scalability.
    http://www1.us.dell.com/content/topics/global.aspx/power/en/ps4q01_mulzer?c=us&cs=555&l=en
11. Informationweek: Content Delivery Infrastruktur für das Web.
    http://www.informationweek.de/index.php3?/channels/channel03/020240.htm
12. Apache: Apache modules.
    http://httpd.apache.org/docs/mod/mod_expires.html
13. Datta A., Dutta K, Thomas H..: Proxy-based Acceleration on Dynamically Generated Content on the world wide web. SIGMOD Conference 2002, [97-108]
14. Garg P., Eshghi K, Gschwind T.: Enabling Network Caching of Dynamic Web Objects. Computer Performance Evaluation / Tools 2002, [329-338]
15. <esi> Edge Side Includes: Overview.
    http://www.esi.org/overview.html
16. Anton J., Jacobs L., Liu X.: Web Caching for Database Applications with Oracle Web Cache, SIGMOD Conference 2002, [594-599]
17. IBM High-Volume Web Site Team: How WebSphere Caches Dynamic Content for High Volume Web Sites. 2003.
    http://www7.software.ibm.com/vadd-bin/ftpdl?1/vadc/wsdd/hvws/cache12_15.pdf
18. Jones, A.: Proxy Server Caching.
    http://www.winnetmag.com/Web/Article/ArticleID/8502/8502.html
19. Oracle: JESI Tags for Edge Side Includes.
    http://otn.oracle.com/docs/tech/java/oc4j/Jsp1131/jesitags.htm
20. Azim O.: Cache SOAP Services on the Client Side.
    http://www.javaworld.com/javaworld/jw-03-2002/jw-0308-soap.html

21. Zhou T.: Surfing Web-Caching Technology.
http://www.winnetmag.com/Windows/Article/ArticleID/7199/7199.html
22. Pierzchala S.: Caching for Performance.
http://www.webcaching.org/caching_for_performance.pdf
23. Mohan C.: Caching Technologies for Web Applications. VLDB 2001,
http://www.almaden.ibm.com/u/mohan/Caching_VLDB2001.pdf
24. Takase T., Tatsubori M.: Efficient Webservice Response Caching by Selecting Optimal data Representation. http://www.research.ibm.com/trl/people/mich/pub/200403_icdcs2004.pdf
25. Goodman B.: Accelerate your Web services with Caching.
http://www-106.ibm.com/developerworks/webservices/library/ws-cach1/?ca=degrL19wscaching
26. Terry D.: Issues in Caching Webservices.
http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=38&page=2