Datenbanksysteme und World Wide Web: Caching für Informationsverteilung im Netz

Marc Schanne

Eine Seminararbeit am
Institut für Programmstrukturen und Datenorganisation
Seminar: Datenbanksysteme und World Wide Web, Sommersemester 1997
Betreuerin: Jutta A. Mülle

Universität Karlsruhe

1. Juli 1997

Zusammenfassung

Seit dem Beginn, 1990, zeigt das World Wide Web (WWW oder Web) ein exponentiales Wachstum. Dieses hält an, und niemand weiß, ob, oder wann sich das ändert.

Um die nur begrenzten Netzkapazitäten zu entlasten, hat sich die Technik des Cachings, des Zwischenspeicherns von entfernten Dokumenten und Objekten in der Nähe, etabliert. Diese Arbeit gibt einen Überblick über die aktuellen Techniken, die damit verbundenen Probleme und offenen Fragestellungen. Dies beinhaltet sowohl Client- als auch Proxy-Caching. Besonders Fragen der Kohärenz und Effizienz sind Teil der Überlegung.

In einem weiteren Teil werden neue Entwicklungen auf dem Gebiet des Web-Cachings sowie einige Zukunftsvisionen vorgestellt und bewertet. Die Brauchbarkeit in Bezug auf die steigende Nutzung von Datenbanken im WWW wird abschließend betrachtet.

In eodem prato bos herbam quaerit, canis leporem, ciconia lacertam.[1] *- Seneca*

[1] Auf derselben Wiese sucht die Kuh Gras, der Hund den Hasen, der Storch die Eidechse.

Inhaltsverzeichnis

1 Caching Konzepte **5**
1.1 Client Caching 5
1.2 Proxy Caching 6

2 Offene Fragen und Probleme **6**

3 Aktuelle Techniken im Internet **8**
3.1 Hypertext Transfer Protocol 8
3.1.1 Hierarchisches Caching 8
3.1.2 Harvest und Squid Cache Server 8
3.2 Mirrors 10

4 Aktuelle Ansätze bei Caching Problemen **10**
4.1 Kohärenz 10
4.2 Effizienz 12

5 Mittel- und längerfristige Überlegungen **15**
5.1 HTTP-NG, IIOP 15
5.2 Replikationen, Preloading 16
5.3 Semantisches Caching von Datenbankzugriffen im WWW 17

6 Zusammenfassung und Ausblick **18**

Abbildungsverzeichnis

1 Caching im WWW 5
2 Proxy Anordnungen im Vergleich 9
3 Kohärenz-Mechanismen: validation check *vs.* callback 10
4 Conditional GET, eine Erweiterung von HTTP/1.0 11
5 Derzeit übliche Konfiguartion eines HTTP-Server 16
6 Server und Client beim Internet Inter-ORB Protocol (IIOP) 17
7 Replikation durch Web Caches 18

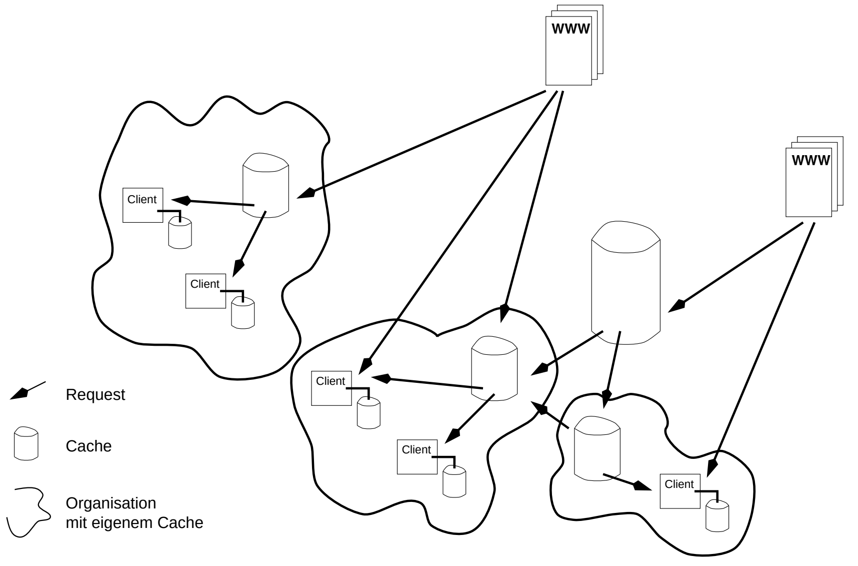


Abbildung 1: Caching im WWW

1 Caching Konzepte

Es existieren zwei unterschiedliche Möglichkeiten des Cachings von Web Seiten[2] im Internet: Client Caching und Proxy Caching. Die Abbildung 1 zeigt das Zusammenspiel der beiden Ansätze.

1.1 Client Caching

Client Caches sind Teil der Anwendersoftware, z.B. von WWW Browsern, wie *Mosaic* oder dem *Netscape Navigator*. Der Client Cache hält nicht nur das aktuell angezeigte Dokument, sondern auch früher angeforderte. Grundsätzlich kann man zwei Formen des Client Caching unterscheiden: *persistent* und *nicht-persistent*. Persistente Caches verwahren die WWW Dokumente auch zwischen den Aufrufen des Web Browsers. Nicht-persistente Caches löschen ihren Inhalt beim Beenden der Anwendung.

Der Zugriff auf beide Typen von Caches unterscheidet sich nicht. Wenn der Benutzer eine bestimmte WWW Seite anfordert, prüft die Anwendung zuerst ob sich die gewünschte Seite bereits im Cache befindet. Bei Erfolg kann diese Seite dem Benutzer schnell angezeigt werden und muß nicht über das Netz geladen werden.

Als Methoden der Kohärenzprüfung bieten Client Caches derzeit nur sehr rudimentäre Möglichkeiten. Der Netscape Navigator bietet hier z.B. drei Techniken zur Verifikation eines Dokuments: *Once per Session*, *Every Time* und *Never*. Abhängig von dieser Einstellung wird entschieden, ob eine im lokalen Cache befindliche Kopie erst auf Aktualität geprüft oder sofort angezeigt wird. Konsistenzprüfungen werden z.B. mit einem conditional GET an einen HTTP

[2] Wenn im Folgenden von WWW Seiten oder Dokumenten gesprochen wird, sind stets auch sonstige Objekten wie Grafiken, Audio-, Videodaten oder Programme (Applets) gemeint.

oder Proxy Server (vgl. nächsten Abschnitt) unternommen. Mehr hierzu findet sich im Kapitel 4.1 ab Seite 10.

Als grundsätzliche Regel für den Cache gilt: Je größer er ist, desto größer ist die Wahrscheinlichkeit, daß er die gewünschten Daten enthält.

1.2 Proxy Caching

Proxy Caches sind Teil des mit World Wide Web benutzen Netzwerkes. Sie sind auf strategisch günstigen Maschinen plaziert (typischerweise Gateways). Ein Proxy Cache bedient, anders als ein Client-Cache, mehrere Clients.

Eine derzeit häufige weitere Einsatzmöglichkeit für Proxy Caches ist die Funktion eines Firewalls für Sicherheitsaspekte. Der Proxy ist ein Server, durch den der gesamte Zugriff auf Daten außerhalb des Firmen-/Organisationsnetzes erfolgen muß. Ein direkter Zugriff ist nicht erlaubt.

Wenn ein Benutzer eine bestimmte WWW Seite anfragt, geht diese Anfrage an den Proxy Server, und dieser prüft, ob er das entsprechende Objekt bereits gespeichert hat. So beantwortet ein Proxy Server die Anfragen von vielen lokalen Clients - durchaus auch mit eigenem Client Cache.

Üblicherweise besteht zwischen den Clients eine Beziehung, sie sind Teil der gleichen Organisation oder werden zu ähnlichen Zwecken verwendet. Deshalb kann man davon ausgehen, daß von verschiedenen Clients gleiche WWW-Objekte angefordert werden und so ein gemeinsames Caching sinnvoll ist. So eingesetzte Proxies werden auch als *forward proxies* bezeichnet.

Proxies, die hingegen nur eine geringe Anzahl von Dokumenten (z.B. die einer einzigen Organisation) cachen, aber weltweit von Clients benutzt werden, nennt man *reverse proxies*. Der zentrale WWW-Server der Uni-Karlsruhe (www.uni-karlsruhe.de) zum Beispiel ermöglicht einen zugriffstransparenten, einheitlichen Zugang auf den Inhalt vieler Server. Diese Organisationsform eines Proxy Caches ergibt grundlegend andere Anforderungen und Probleme und ist nicht Teil dieser Betrachtungen.

2 Offene Fragen und Probleme

Ziel des Cachings ist es, das Datenaufkommen im Netz zu verringern. Im einzelnen kann man die Vorteile für Einzelpersonen, Anwender und Internet-Provider kurz zusammenfassen:
Caching ...

... verbessert die Antwortzeit für den Benutzer

... verringert den Bandbreitenbedarf jedes einzelnen Benutzer

... reduziert die Beanspruchung des Internets insgesamt

... beschränkt den Aufwand beim Remote Server

Es ergeben sich sowohl für den Privatanwender als auch für die Gemeinschaft Vorteile. Allgemein verbessert Caching das Internet für jedermann.

Bei allen gegebenen Vorteilen sind aber besonders die noch offenen Fragen und Probleme mit Caching für diese Betrachtungen hier interessant. Das exponentiale Wachstum legt außerdem die Forderungen an Skalierbarkeit und Einfachheit des Caching-Verfahrens fest. Erfolgreiche Lösungen für mittlere und kleine LANs sind im Bereich von derzeit 3.000 neuen Domains täglich[3] nicht mehr praktikabel.

Ein Problem, das in diesem Zusammenhang besondere Betrachtung erfordert, ist die Frage wie gecachete Objekte aktuell gehalten werden können, d.h. bei Änderung der Orginalobjekte die Kopien erneuert werden. In der Terminologie traditioneller Verteilter Systeme wird dies *Kohärenz* genannt. Um effizient Netzressourcen zu schonen, muß versucht werden, statt des Originaldokuments möglichst oft eine lokale Kopie zu verwenden. Algorithmen und Möglichkeiten, die das Internet derzeit vorsieht, werden im Kapitel 4.1 näher betrachtet.

Die aktuellen, relativ begrenzten Techniken bedingen aber, daß existierende Web Caches derzeit zu oft veraltete Informationen liefern und außerdem den Benutzer länger warten lassen als wirklich nötig [DP96]. Das Kapitel 5 skizziert einige Ansätze, die dieses Problem in Zukunft besser lösen sollen.

Die weitere Entwicklung des Internets ist nur schwer abzuschätzen, sicher ist aber ein weiteres Wachstum. Auch eine Entwicklung hin zu mehr veränderlichen Elementen und die Integration bestehender Datenbasen ins WWW deutet sich an.

In diesem Zusammenhang stellt sich die Frage: Wieviel ist cachebar?

Derzeit wird oft aus Mangel an geeigneten Mechanismen Caching für diese variablen Darstellungen völlig ausgeschaltet. CGI-Programme erzeugen dynamisch bei jedem Aufruf eine neue HTML-Seite. Hier muß überlegt werden, inwieweit sich diese dynamisch generierten Objekte in kleinere statische, cachebare Teile zerlegen lassen. Die mit zunehmender Kommerzialisierung des Internets immer wichtiger werdende Frage nach sicheren statistischen Zahlen über Zugriffe auf einzelne Web-Inhalte fällt in den gleichen Zusammenhang. Auch hier wird bisher Caching unterdrückt, um unverzerrte Statistiken zu erhalten.

Veränderungen sind auch bei den derzeit üblichen Anfragesystemen für relationale Datenbanken im WWW, bei den verwendeten Cache-Mechanismen und der Datenausgabe nötig. Das Kapitel 5.3 ab Seite 17 stellt hierzu einen Ansatz semantischen Cachens kurz dar.

Die praktische Leistungsfähigkeit von Cache Servern hängt aber zusätzlich zu dieser theoretisch möglichen Treffer-Rate[4] auch von der realen, optimalen Ausnutzung des Cache-Speichers ab. Da die Speicherkapazität von Proxies nach oben beschränkt ist, kann die Entscheidung, welches Objekt ersetzt wird, die Effizienz des Caches massiv beeinflussen. Die Frage nach der optimalen Ersetzungsstrategie für Web Caches versucht das Kapitel 4.2 zu klären.

Wo ein Cache im Netz platziert werden sollte, ist nicht ohne Berücksichtigung der verwendeten Cache-Struktur, hierarchisch oder Harvest/Squid, zu klären. In Zusammenhang mit der Harvest Cache Server Software stellt [Fis97] die nötigen Voraussetzungen dar.

Aspekte der Sicherheit, des Datenschutzes und der Möglichkeit eines *Pay-per-View* -Services bei Caching im Netz sind ebenfalls noch nicht optimal gelöst. Auch hier wird Caching derzeit oft mangels anderer Möglichkeiten umgangen. Diese Arbeit beschränkt sich aber im

[3]iX Magazin für professionelle Informationstechnik 5/1997, S. 40

[4]Cache-Treffer oder Cache-Hit: Das geforderte Dokument befindet sich im Cache und muß nicht wie bei einem Cache-Miss vom Original-Server geladen werden.

Wesentlichen auf die Möglichkeiten bei den Problemen der Kohärenz und Effizienz von Web-Caches. Für speziellere Problemstellungen sind [Mar96], [MLB95], [Nab97] und [Nea96] zu empfehlen.

Die weniger technische Frage des Copyrights bei Caching und Replizieren wird in [San96] näher betrachtet.

3 Aktuelle Techniken im Internet

Dieses Kapitel stellt einige im WWW verwendete Techniken dar, die sich Cachings bedienen oder Caching unterstützen. In diesem Zusammenhang werden die Probleme, die einzelne Methoden lösen oder erst aufwerfen, kurz umrissen. Im nächsten Kapitel wird dann gesondert auf die Hauptaspekte bei Caching (Kohärenz und Effizienz) eingegangen.

3.1 Hypertext Transfer Protocol

Derzeit aktuelles Protokoll von WWW Clients und Servern ist das Hypertext Transfer Protocol in der Version 1.0 (HTTP/1.0)[5]. Aufbauend auf einer TCP/IP[6] Verbindung bietet es die Möglichkeit über URL[7] eindeutig deklarierte Objekte im WWW zu adressieren und zu beziehen.

Die Möglichkeiten für Caching sind eingeschränkt. Erst mit der verbesserten abwärtskompatiblen Version 1.1 wurden die offensichtlichsten Mängel von HTTP beseitigt. Im Kapitel 4.1 werden die Veränderungen im Zusammenhang mit Kohärenz-Mechanismen erklärt.

3.1.1 Hierarchisches Caching

Der ursprünglich vom *CERN* entwickelte HTTP-Dämon *httpd* bietet neben der Web Server Funktionalität auch die Option als Proxy mit lokalem Cache zu fungieren. Mittlerweile leistet das *W3C* den Support. Erfahrungen mit diesem Server sind in die Neuentwicklung, den *Jigsaw*[8], eingeflossen. Ähnliche Funktionalität wie der W3C Server bieten derzeit auch einige kommerzielle Anbieter, wie z.B. der *Netscape Proxy Server* [Net97].

Um die Effizienz einfacher Caches zu steigern, haben sich hierarchische Strukturen mit Caches auf verschiedenen Ebenen entwickelt. Bei einer reinen Baumstruktur haben sich aber die Verbindungen zwischen den Servern weiter oben in der Wurzel als problematisch, weil leicht überlastbar, erwiesen. Abbildung 2 zeigt die hierarchische Struktur im Vergleich zu der im nächsten Kapitel beschrieben Harvest Anordnung, die dieses Problem behoben hat.

3.1.2 Harvest und Squid Cache Server

Anders als der CERN Server ist die *Harvest Cache Software* nur als Caching-Server entwickelt worden. Sie bietet WWW Proxy Caching mit Unterstützung von FTP, Gopher und HTTP Objekten, die wesentlich schneller ist als die CERN Software. So muß der CERN Server

[5] beschrieben im RFC 1945 [Ber93]
[6] Transmission Control Protocol/Internet Protocol
[7] Uniform Resource Locator
[8] Referenz-Server für HTTP/1.1, vollständig modular und in Java geschrieben

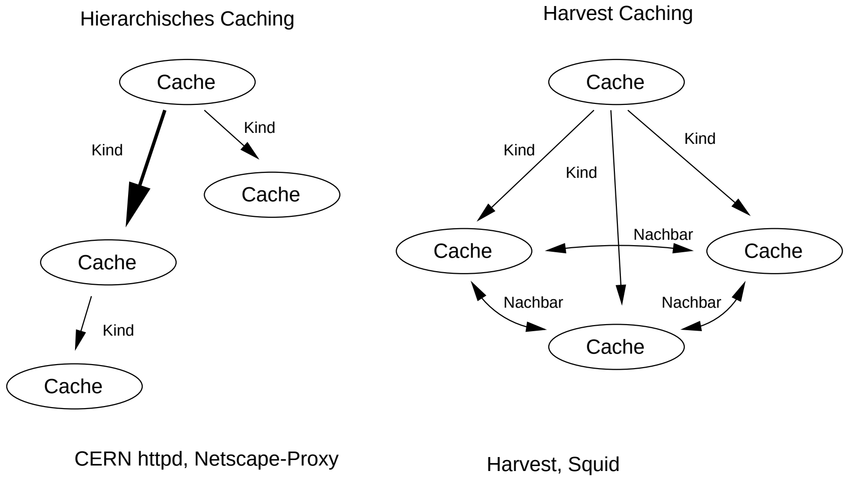


Abbildung 2: Proxy Anordnungen im Vergleich

für jede eingehende Anfrage einen neuen Prozeß starten und der Harvest-Server nur bei FTP Transfer.

Die Harvest-Software ist bis zur Version 1.4 frei erhältlich. Die aktuelle Version 2.0 ist mit Support als kommerzielles Produkt zu beziehen. Als freie Weiterentwicklung des Harvest Caches gibt es den Squid beim *National Laboratory for Applied Network Research*. In Funktionalität und Stabilität entspricht er etwa dem kommerziellen Produkt.

Beide Proxy Caches - Harvest und Squid - unterstützen, anders als der Server vom W3C (CERN), nicht nur hierarchisches Caching, sondern auch eine komplexe Strukturierung über Nachbarn und Eltern (vgl. Abbildung 2). So ist es einem Benutzer möglich, ein Dokument über den gesamten Cache-Verbund zu erhalten. Bei einem Cache-Miss[9] werden zuerst die Nachbarn über ein optimiertes, auf UDP[10] aufgesetztes Protokoll befragt, und nur wenn keiner erreichbar (Timeout) oder es nicht vorhanden ist, wird das Objekt über den Parent-Cache bezogen.

Der Cache kann auch *negativ cachen*. Wenn ein Server kurzfristig tot ist, wird (auf Basis eines definierbaren Timeouts) nicht ständig Bandbreite verschwendet, um ihn zu erreichen [Sch96].

Derzeit baut der *Verein zur Förderung eines Deutschen Forschungsnetzes e.V.* (DFN-Verein) in einem vom *Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie* (BMBF) geförderten Projekt einen nationalen Cache-Server-Verbund [Pra95].

Weitere Informationen hinsichtlich Einsatz und Funktionalität von Harvest Cache Software bietet [Fis97].

[9] das Objekt ist nicht im lokalen Cache

[10] User Datagram Protocol

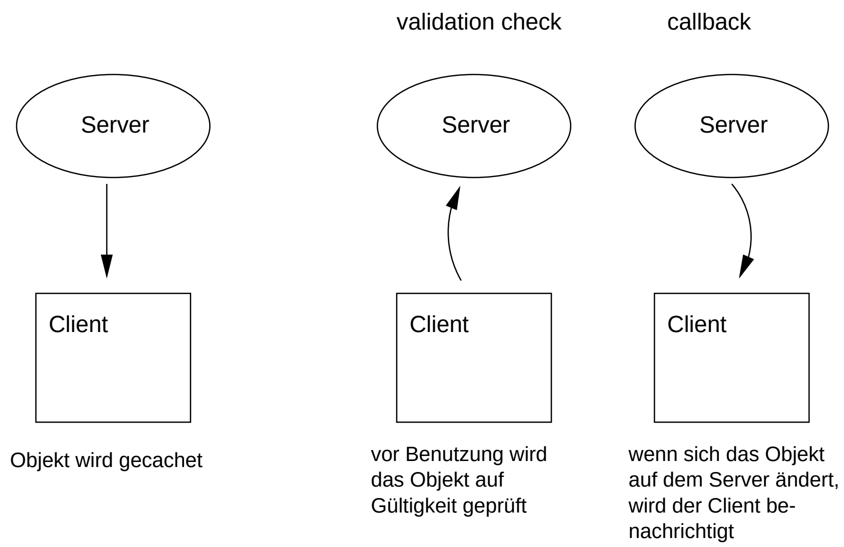


Abbildung 3: Kohärenz-Mechanismen: validation check *vs.* callback

3.2 Mirrors

Besonders aus dem Anonymous FTP[11] Bereich ist die Technik der Replikation entfernter Server bekannt. Sogenannte Mirrors oder Spiegelungen entfernter FTP Server verringern internationale Netzlast. Um lokale Kopien der gesuchten Objekte zu finden werden Service wie Archie angeboten, die über Indizes der einzelnen Server diese Objekte lokalisieren.

Einige Server verteilen die Benutzer auch selbst dynamisch auf den nächsten freien Knoten im Verbund. Dies funktioniert ebenfalls über vorher eingerichtete Indizes, die noch per Hand gepflegt werden können. Für den Einsatz im World Wide Web müßte dieser Ansatz vereinfacht und automatisiert werden. Ideen in diese Richtung werden im Kapitel 5.2 näher betrachtet und auf ihre Leistungsfähigkeit getestet.

4 Aktuelle Ansätze bei Caching Problemen

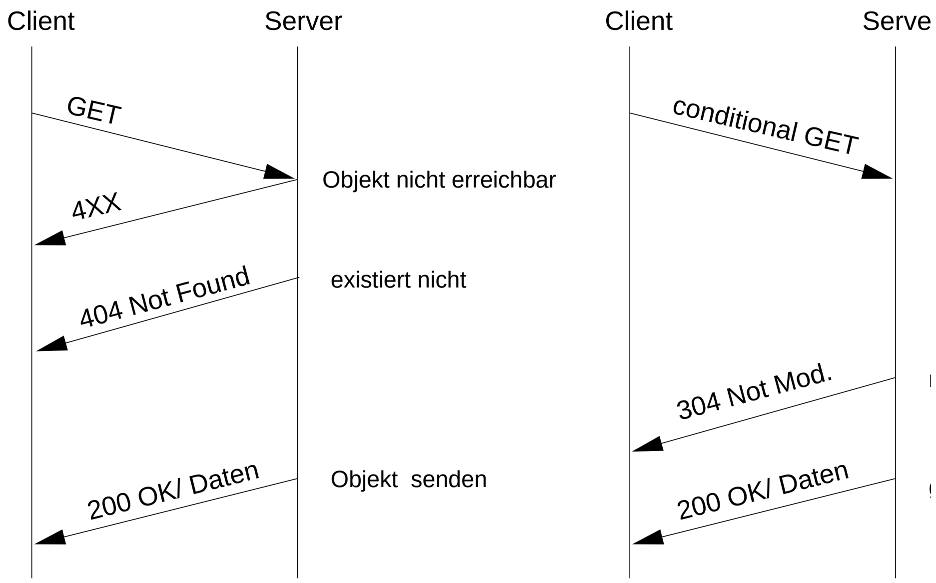
4.1 Kohärenz

Alle Web Caches müssen versuchen die gespeicherten Objekte aktuell zu halten. Dieses Problem ist aus traditionellen verteilten Systemen bekannt. Prinzipiell ist das Web nur ein größeres verteiltes Dateisystem und verteilte Dateisysteme werden seit Jahren erfolgreich eingesetzt.

Kohärenz-Mechanismen kann man in zwei generelle Klassen einteilen: *validation check* und *callback* Mechanismen. Die Abbildung 3 zeigt die zwei grundlegenden Techniken, mit der die Korrektheit und Aktualität eines gecacheten Objekts garantiert werden kann.

Die Größenordnung des WWW bedingt, daß dort eigentlich nur die Kohärenzprüfung über validation check praktikabel ist. Eine einzelne Seite kann in Tausend oder Millionen

[11] File Transfer Protocol, beschrieben in RFC 959



conditional GET: GET /page.html HTTP/1.0
If-Modified-Since: Thu, 17 Jun 1997 15:57:41 GMT

Abbildung 4: Conditional GET, eine Erweiterung von HTTP/1.0

Caches (jeder Web-Browser ist einer) gespeichert werden. Dies macht es für einen Server unmöglich, alle Caches zu benachrichtigen. Durch eine hierarchische Struktur könnte man zwar die Zahl der zu benachrichtigenden Caches verringern, aber es bleibt ineffektiv bei jeder Änderung irgendeiner Seite im WWW eine Notify-Lawine zu starten.

Naive Kohärenz-Mechanismen sind die am Beispiel heutiger Browser schon angesprochenen *Check-Every-Time* und *Check-Never* Mechanismen. Im ersten Fall wird immer, wenn ein *GET* oder *conditional GET*[12] für eine gecachete Seite eingeht, ein conditional GET an den nächsthöheren Cache oder an den Server gesendet. Das conditional GET ergänzt das normale GET des HTTP/1.0 um ein `Not Modified`. Der Client erweitert die GET-Anfrage um eine `If-Modified-Since` Zeile mit dem Datum der im Cache gespeicherten Seite. Ist das Originaldokument unverändert, antwortet der Server mit einem `304 Not Modified` und der Client zeigt die lokale Kopie an. Im anderen Fall sendet der Server ein `200 OK` und die neue HTML-Seite. Der Check-Every-Time-Ansatz in Zusammenhang mit einem conditional GET war der erste eingesetzte Kohärenz-Mechanismus. Für den umfangreichen Einsatz im World Wide Web ist er allerdings wenig brauchbar. Wenn jeder Cache mit einem conditional GET an einen übergeordneten Cache bzw. an den Server selbst reagiert, wird jede Anfrage bis zum Server durchgereicht. Dies führt, selbst wenn keine HTML-Daten übertragen werden müssen, zu übermäßiger Netzlast und reduziert die Anzahl der Anfragen an den Remote Server nicht.

Der zweite naive Kohärenz-Mechanismus, der oben angesprochen wird: Never-Check, überprüft nie, ob die gecachete Seite noch aktuell ist. Um eine Seite explizit neu zu laden, muß

[12]Die Abbildung 4 zeigt das Zusammenspiel von Client und Server bei einem klassischem GET und die Ergänzungen, die ein conditional GET bietet.

der User einen Header mit `Pragma:no-Cache`-Zeile verwenden. Diese Technik verringert zwar die Netzlast, gibt aber keinerlei Garantie für die Aktualität des übertragen Dokuments.

HTTP/1.1[13] ist die Weiterentwicklung des Hypertext Transfer Protocol 1.0 und bietet neben Fortschritten, wie Pipelining und Kompression, auch hinsichtlich des Cachings einige zusätzliche Möglichkeiten [Fie97]. Caching in HTTP/1.1 basiert auf zwei unterschiedlichen Ansätzen: dem *Expiration-* und dem *Validation-Modell*.

Expiration

Das Expiration-Modell definiert ein Verfalldatum für eine Seite. So können Zugriffe auf den Server eingespart werden, wenn ein aktuelles Dokument im Cache-Speicher existiert.

Entweder gibt ein Server für eine Seite explizit eine Verfallzeit an, oder der Proxy Cache berechnet eine heuristische. Für den zweiten Fall gibt die HTTP/1.1-Spezifikation zwar keine Algorithmen, aber Rahmenvorgaben für ihr Ergebnis an. Alle Server sind aufgefordert, so oft wie möglich eine explizite Expiration-Zeit anzugeben, da die heuristischen Grenzzeiten nur mit Vorsicht zu betrachten sind [Fie97].

Validation

Unter dem Validation-Modell faßt das RFC 2068 die Techniken des Hypertext Transfer Protocols zur Prüfung von Kopien im Cache auf Gültigkeit (unverändert gegenüber dem Original) zusammen. Sowohl das vom conditional GET bekannte `Last-Modified`-Datum, als auch zusätzliche *Entity Tags*, sind hier zum Abgleich zwischen Cache und Server vorgesehen.

Allgemein

Zusätzlich zu den beiden oben angegebenen Modellen werden in der HTTP/1.1-Spezifikation, in einem eigenen Kapitel zu Caching weitere Rahmenbedingungen bezüglich Korrektheit und Transparenz für Server, Clients und Caches festgelegt. Implementiert ist der neue Standard aber erst bei wenigen Referenzprodukten. Einige Teile, insbesondere Entwicklungen der Caching-Möglichkeiten, sind aber schon bei mehreren HTTP/1.0-Anwendungen hinzugefügt.

Weitere Überlegungen [DP96] betonen die Notwendigkeit, dem Benutzer eine Art Gültigkeitsgarantie in Prozent zu bieten. Zu jedem gesendeten Dokument sollte dem Benutzer zusätzlich das Expiration-Datum oder die spekulative Berechnung des Caches offengelegt werden. Wenn das so angezeigte Dokument nicht den Bedingungen des Benutzers genügt, kann er explizit eine „bessere“ Kopie aus einem höheren Cache oder direkt vom Orginal-Server anfordern.

4.2 Effizienz

Die zweite Hauptforderung neben Kohärenz an Web Caches ist Effizienz. Dies kann nur über eine hohe Treffer-Quote erreicht werden. Und da derzeitige Caches nur einen relativ beschränkten Speicherraum besitzen, besteht die Notwendigkeit, ältere Dokumente aus dem Speicher zu löschen. Der von Festplatten und RAM-Caches bekannte, und dort optimale

[13] beschrieben in RFC 2068

Algorithmus *Least Recently Used* (LRU), ist beim Caching von WWW Objekten nicht optimal [Abr95]. Im Folgenden wird ein kurzer Überblick über die Problematik gegeben und ein Vergleich verschiedener denkbarer Ersetzungsstrategien versucht.

LRU - Least Recently Used

Der klassische LRU-Ersetzungsalgorithmus löscht solange das least recently used Objekt aus dem Cache, bis genügend freier Platz für das neue Objekt geschaffen ist. LRU ersetzt so vielleicht auch mehrere kleine Dokumente, um Raum für ein großes zu schaffen.

LRU/MIN

Diese Variante des LRU-Algorithmus versucht die Anzahl der zu ersetzenden Dokumente zu minimieren. Hier eine kurze formale Beschreibung analog [Abr95]:
Liste L, Integer T, Größe des neuen Objekts S

1. T = S
2. L = Liste alle Dokumente im Cache gleich oder größer als T (L kann leer sein)
3. Löschen des LRU-Dokuments von der Liste, bis L leer oder der freie Cache Platz größer oder gleich T
4. Falls freier Cache-Speicher $< S$, dann T = T/2 und weiter mit 2.

LRU/THOLD

Diese Ersetzungsstrategie ist identisch zu LRU, nur werden Objekte, die größer als eine festgelegte Threshold-Größe sind, nicht gecachet. So wird verhindert, daß im Verhältnis zur Cache-Größe sehr große Dokumente viele kleinere Dokumente verdrängen. Selbst wenn genügend freier Platz im Cache-Speicher existiert wird das Objekt nicht gespeichert.

LFU - Least Frequently Used

LFU ersetzt das *least frequently used* Dokument im Cache. Hier wird nicht über die Zeit seit dem letzten Zugriff, sondern über die Häufigkeit des Zugriffs entschieden. Besonders populäre Dokumente werden so bevorzugt. Insbesondere für die Verwendung bei Proxies ist die LFU-Ersetzungsstrategie günstiger als der LRU-Algorithmus [Wil96].

SIZE

Proxies können aber auch versuchen, die nötige Netzwerkbandbreite zu minimieren. SIZE ist ein Ersetzungsalgorithmus, der mit Häufigkeit und Größe über die Ersetzung von Cache-Objekten entscheidet.

LAT - Latency Estimation Algorithm

Dieser Algorithmus, näher beschrieben in [WA97], macht die Ersetzung eines Cache-Objekts direkt von der Download-Zeit abhängig. Eine WWW-Benutzer Umfrage des Graphic, Visualization & Usability Centers (GVU) ergab, daß *Geschwindigkeit* das Hauptproblem von Web-Benutzern ist. Der LAT-Algorithmus ist direkt von diesem Ergebnis motiviert und ersetzt die Dokumente mit der kürzesten Ladeverbindungszeit. Die durchschnittlichen Verbindungszeiten werden serverweise auf Proxyseite gespeichert und so für einzelne Dokumente unterschiedlicher Größe die Download-Dauer vorausberechnet.

HYB - Hybrid Algorithm

Auch der HYB-Algorithmus wird in [WA97] eingeführt und ausführlich verglichen. Er generiert nicht nur aus Ladezeit, sondern auch aus Größe und Häufigkeit der Zugriffe einen Entscheidungsfaktor. In [WA97] wird dieser mit dem folgenden Ausdruck berechnet:

$$\frac{\left(c_{ser(i)} + \frac{W_B}{b_{ser(i)}}\right)\left(n_i^{W_N}\right)}{s_i}$$

mit:
n_i Anzahl der Zugriffe auf das Dokument i seit es zuletzt in den Cache geladen worden ist
s_i Größe des Dokuments i in Byte
$c_{ser(i)}$ Zeit zum Öffnen einer TCP-Verbindung zum Server des Dokuments i
$b_{ser(i)}$Bandbreite zum Server in Byte/Sec
W_B, W_N Konstanten für die relative Gewichtung der Eingangsvariablen n_i und $b_{ser(i)}$

Web-Caches besitzen eine Obergrenze, die auf die Anforderungsrate aus statistischen Untersuchungen zurückzuführen ist, für die Hit-Rate von 30%-50% bei theoretisch unendlicher Cache-Größe [Abr95]. Ziel der Ersetzungsstrategie ist es, diese theoretischen Werte auch bei realen, kleineren Cache-Speichern zu erreichen. [Abr95] vergleicht die drei LRU Varianten bei ihrem Verhalten Cache-Speicher zu befreien und kommt zu dem Ergebnis, daß der klassische LRU schlechter als LRU/MIN und LRU/THOLD abschneidet. Dies liegt daran, daß er unabhängig von der Größe eines Objekts z.B. für ein großes Video von einem Benutzer viele kleinere Dokumente im Cache ersetzt. Daraus resultieren Cache-Fehler für viele Benutzer. LRU-THOLD hält die Cache-Größe relativ klein, nur ist es schwierig, die effektivste Threshold-Grenze zu finden. Das Ergebnis hinsichtlich der optimalen Ersetzungstrategie fällt bei [Abr95] folgendermaßen aus: *use LRU-MIN until the cache size approached 100% of the available disk size, and the switch to LRU-THOLD with a threshold that is gradually reduced, until the cache size reaches a low-water mark.*

Ein weiteres Ergebnis ist die Empfehlung, nonpersistente Client-Caches zusammen mit einem gemeinsamen Proxy zu verwenden. Bei persistenten Caches der einzelnen Clients fällt die Trefferrate des Proxy Caches mit der Zeit, da sich die Caches der einzelnen Client füllen und nur noch Fehlanfragen dort an den Second-Level-Proxy-Cache weitergereicht werden. Mit einer besseren Nutzung des Proxy-Caches kann Speicherplatz gespart werden, und aufgrund der größeren Anzahl der Zugriffe kann der Proxy Server besser über das Caching einzelner Dokumente entscheiden [Abr95].

Im Vergleich zwischen LRU- und LFU-Strategien zeigt sich, daß für den Einsatz bei Proxy-Caches die Anzahl der Zugriffe auf ein Dokument statt der Zeit seit dem letzten Zugriff das bessere Entscheidungskriterium ist [Wil96]. Deshalb verwendet der Algorithmus HYB auch diesen Faktor [WA97].

Die Ergebnisse aus [WA97] belegen außerdem, daß eine Ersetzungsstrategie, die nur auf die geschätzte Download-Zeit des Objektes (LAT) aufbaut, nicht optimal ist. Die Trefferrate liegt deutlich unter der, die z.B. SIZE bei gleicher Versuchsumgebung ergibt [WA97].

Der ebenfalls in [WA97] geprüfte Algorithmus HYB ergibt sowohl bei Trefferrate als auch der Download-Zeit, auf verschiedenen Datenbasen die besten Ergebnisse. Wahrscheinlich ist der Ansatz, mehrere Faktoren in die Ersetzungsentscheidung miteinzubeziehen, sinnvoll und kann die durchschnittlichen Werte verbessern und stabilisieren.

Außerdem darf der psychologische Effekt der großen Variation der Download-Zeit auf den Benutzer nicht ignoriert werden. Bei Strategien, die die Zeit berücksichtigen, variieren die Download-Zeiten weniger, und dem Benutzer fällt gleichmäßig längeres kurzes Warten weniger auf, als langes Warten bei einzelnen Seiten [WA97].

Alle hier vorgestellten Ersetzungsstrategien haben gemeinsam, daß sie über Objektattribute, wie Zeit, Zugriffshäufigkeit, Größe, etc. die Verweildauer im Cache berechnen. Der Inhalt wird nicht bewertet. Daß es hier auch andere Ansätze gibt, zeigt der Abschnitt 5.3. Dort wird ein besonders für Datenbanken interessanter Ansatz des semantischen Cachings beschrieben.

5 Mittel- und längerfristige Überlegungen

5.1 HTTP-NG, IIOP

Mit Einführung der 1.1 Version des Hypertext Transfer Protocols arbeitet das W3 Consortium schon an einer Weiterentwicklung. HTTP-NG[14] soll die Veränderung des WWW hin zu dynamischen Seiten und größerer Interaktivität möglich machen. Der ursprüngliche Ansatz, daß ein Client von verschiedenen Servern über URLs HTML-Seiten anfordert und diese anzeigt, ist seit Bestehen des WWW wesentlich ausgeweitet worden. Nicht nur, daß eine einfache Web-Seite mittlerweile aus vielen verschiedenen Objekten, wie Text, Grafiken, Applets und sonstigen Komponenten besteht, insbesondere ist eine Entwicklung hin zu dynamisch generierten Inhalten zu erkennen.

Das Common Gateway Interface (CGI) ermöglicht dem Server beliebige Inhalt *on demand* zu erzeugen. Jeder Aufruf führt auf Serverseite zur Abarbeitung eines Programms, das dynamisch den Inhalt der WWW Seite erzeugt. So wird derzeit noch häufig eine propritäre Anbindung von Datenbanken ans WWW implementiert. Die relativ beschränkten Möglichkeiten und die Zunahme der Last auf Serverseite hat einen weiteren Entwicklungsstand gefördert. Durch Erweiterung der Browser um die Möglichkeit Java-Applets Client-seitig ausführen zu können, kann so — immer noch proprietär — die Funktionalität erweitert werden.

Dieser bietet aber weder standardisierte Kommunikationsprotokolle, noch wird so eine Caching-Struktur unterstützt. Dies soll HTTP-NG zusammen mit der gewünschten Funktionalität unterstützen (vgl. Abbildung 5).

[14]next Generation

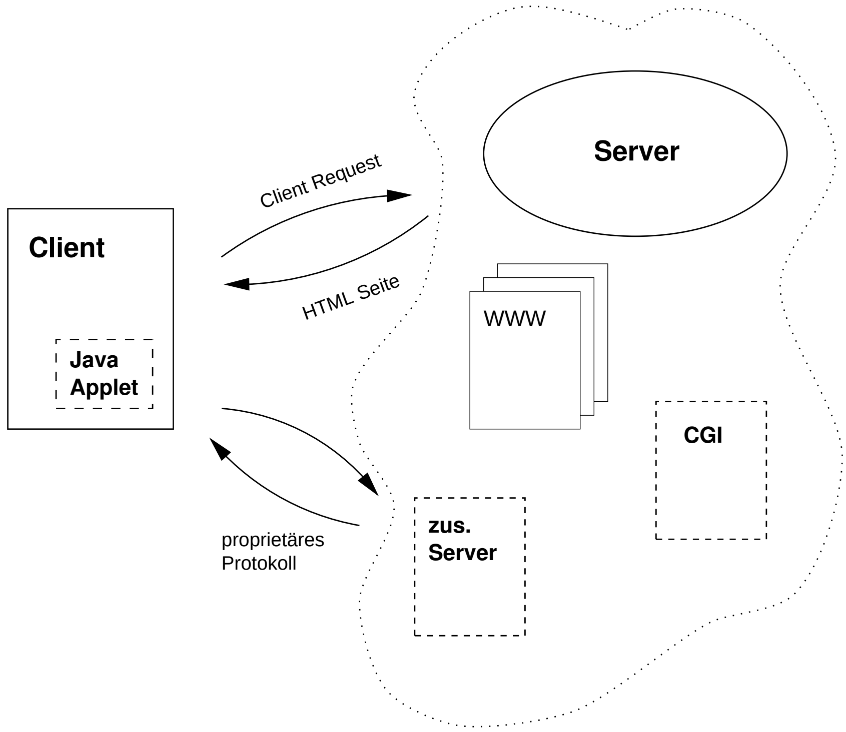


Abbildung 5: Derzeit übliche Konfiguartion eines HTTP-Server

Eine andere Entwicklungsrichtung die Alternativen zum derzeitigen HTTP-Standard bietet ist das Internet Inter-ORB[15]-Protocol (IIOP) (vgl. Abbildung 6). Der WWW Server befindet sich auf einem Rechner, auf dem außerdem ein CORBA[16] -Server läuft. CORBA als langjähriger offener Standard bietet gute Voraussetzungen, mit denen IIOP die derzeitige Mischung aus HTTP und proprietären Lösungen ersetzen und vereinheitlichen kann [Mer97].

5.2 Replikationen, Preloading

Ein weiterer vielversprechender Trend [Ber95] ist die Weiterentwicklung der Spiegelung häufig benutzter Server in die Nähe der Benutzer. Schon heute werden viele populäre Web Server (z.B. Alta Vista, Netscape) weltweit repliziert. Dies ist eine Möglichkeit, erfolgreich mit der wachsenden Anzahl an Zugriffen auf Serverseite umzugehen. Nur muß der Anwender zur Zeit wissen und explizit angeben, welche Replikation er benutzen soll. Dies kann nicht die Lösung für die zunehmende Anzahl der Benutzer sein, die die Technik des Internets nur anwenden, nicht aber verstehen wollen. Hier müßte ein Mechanismus gefunden werden, der automatisch die „nächstgelegene“, verfügbare Kopie liefert. Möglichkeiten, die auf NNTP[17] oder Multicast-Verbindungen aufbauen, sind hier vorstellbar [Ber95].

Statt neuer Technologie geht ein anderer Vorschlag davon aus, daß es besser ist, innerhalb

[15] Object Request Broker
[16] Common Object Request Broker Architecture
[17] Network News Transfer Protocol, beschrieben in RFC 977

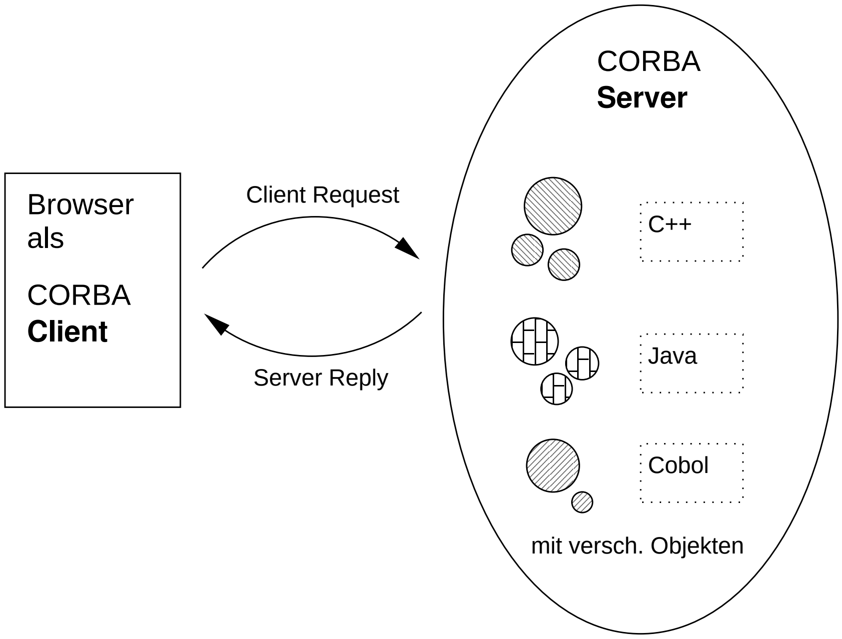


Abbildung 6: Server und Client beim Internet Inter-ORB Protocol (IIOP)

des bestehenden und auszubauenden Cache-Verbundes Replikation zu unterstützen [DP96]. Wenn ein Dokument eines Caches der Ebene n von *vielen* Caches der Ebene n-1 angefordert wird, sendet er das populäre Objekt an alle n-1-Caches. Die Abbildung 7 verdeutlicht diesen Mechanismus: Das xy-Objekt wird von vielen (hier zwei von fünf) Caches angefordert, der n-Level-Cache beantwortet die Anfrage normal mit einem OK und den Daten von xy. Doch zusätzlich sendet er das „heiße“ Objekt auch an die restlichen (drei) Caches. Dieses Vorgehen ermöglicht es, sozusagen innerhalb der Web Caches Replikationen häufig frequentierter Server nahe bei den Benutzern zu erzeugen, ohne die oben angesprochenen Nachteile und noch zu klärenden Fragen.

Bei Preloading oder Prefetching, einer möglichen Erweiterung des Kohärenz-Mechanismus, prüft der Cache regelmäßig, ohne direkte Anfrage eines Clients, selbständig die Aktualität „interessanter“ Objekte im Speicher. In wie weit dies sinnvoll ist und welche Algorithmen gute Ergebnisse liefern, muß noch geprüft werden [DP96].

5.3 Semantisches Caching von Datenbankzugriffen im WWW

Wie in Abschnitt 4.2 schon angedeutet, sind die dort beschriebenen Ersetzungsstrategien für das Caching von Ausgaben relationaler Datenbanken nicht optimal. Beim assoziativen Zugriff auf eine Datenbank werden Datenobjekte nicht direkt spezifiziert sondern dynamisch über ihren Datenwert ausgewählt und in Tupeln gruppiert. Deshalb ist der auf Objekte ausgelegte Ansatz heutiger Caches nicht geeignet.

Eine andere Strukturierung für Caches stellt [Dar96] vor. Es werden nicht Objekte, sondern semantische Regionen im Speicher abgelegt. Jede semantische Region besitzt eine Formel von Bedingungen, die die Tupel in ihr beschreibt. Bei weiteren Anfragen an das Datenbanksystem

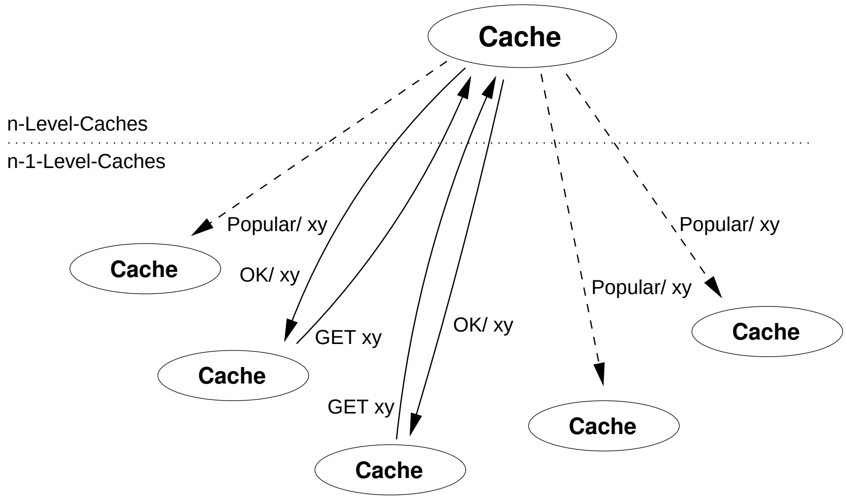


Abbildung 7: Replikation durch Web Caches

wird die Anfrage in zwei disjunkte Teile zerlegt. Ein Teil wird mit den Daten im Cache beantwortet, und der zweite betrifft fehlende Tupel und wird an die Datenbank selbst gestellt.

Semantische Regionen lassen sich in kleinere Teilgruppen zerlegen oder zu größeren verschmelzen. Ersetzungsentscheidungen werden so immer über die gesamte semantische Region getroffen [Dar96].

Unterstützt wird diese Technik derzeit noch von keinem Web-Cache, und selbst im reinen Datenbankumfeld wird sie nur in geringem Umfang eingesetzt. Dies liegt hauptsächlich am noch erforderlichen Forschungseinsatz. Viele Fragen sind noch nicht oder nur teilweise geklärt.

6 Zusammenfassung und Ausblick

Die zunehmende Popularität und das exponentielle Wachstum des World Wide Web bieten viele Herausforderungen. Eine gute Lösung vieler dieser Herausforderungen ist Caching. Es reduziert die Last an den WWW Servern, verbessert die Antwort-Zeit und schont die Bandbreite des Netzwerks. Die Technik des Cachings ist geeignet, alle Archive und Mirrors im Internet zu ersetzen. Jedes Objekt könnte über einen eindeutigen Namen weltweit gecachet und erreicht werden.

Besonders durch Vernetzung und Organisation von Web Caches läßt sich deren Effizienz und Aktualität verbessern. Hier ist aber noch viel theoretische und experimentelle Arbeit zu leisten: Wie können Kopien möglichst aktuell gehalten werden? Welche Organisationsform für Cache-Verbunde ist die günstigste? Wie können die Kosten minimiert werden? Welche Möglichkeiten bietet Preloading? Wie läßt sich die Integration von Datenbanken ins WWW und ins Caching verbessern?

Literatur

[1] Abr95]
MARC ABRAMS, CHARLES R. STANDRIDGE, u.a.: Caching Proxies: Limitations and Potentials. 7. Okt. 1995.
http://www.w3.org/pub/WWW/Journal/1/abrams.155/paper/155.html

[2] Ber93]
TIM BERNERS-LEE: Hypertext Transfer Protocol (HTTP). Internet Draft. CERN, 5. Nov. 1993.

[3] Ber95]
TIM BERNERS-LEE: Propagation, Replication and Caching on the Web. 1995.
http://www.w3.org/pub/WWW/Propagation/Activity.html

[4] Dar96]
SHAUL DAR, MICHAEL J. FRANKLIN, u.a.: Semantic Data Caching and Replacement. Mumbai (Bombay), India, 1996.
http://sunsite.informatik.rwth-aachen.de/dblp/db/conf/vldb/DarFJST96.html

[5] DP96]
ADAM DINGLE, TOMAS PARTL: Web Cache Coherence. Paris, 1996. 5. International World Wide web Conference.
http://www5conf.inria.fr/

[6] Fie97]
R. FIELDING, u. a.: Hypertext Transfer Protocol - - HTTP/1.1. January 1997.
http://andrew2.andrew.cmu.edu.edu/rfc/rfc2068.html

[7] Fis97]
HANS-JÖRG FISCHER: Datenbanksysteme und World Wide Web: Objekt-Caching im Internet. Seminar am Institut für Programmstrukturen und Datenorganisation, Karlsruhe, Juni 1997.

[8] Mar96]
EVANGELOS P. MARKATOS: Main Memory Caching of Web Documents. Heraklio, Crete, Greece, 1996. 5. International World Wide Web Conference.
http://www5conf.inria.fr/

[9] Mer97]
BERNHARD MERKLE: (R)evolution. IIOP - Alternative zu HTTP. aus: iX 7/1997 S. 136 ff.

[10] MLB95]
RADHIKA MALPANI, JACOB LORCH, DAVID BERGER: Making World Wide Web Caching Servers Cooperate. 1995. 4. International World Wide Web Conference.
http://www.w3.org/Conferences/WWW4/Papers/59/

[11] Nab97]
MASAAKI NABESHIMA: The Japan Cache Project: An Experiment on Domain Cache. Tokyo, Japan, 1997. 6. International World Wide Web Conference.
http://www6.nttlabs.com/

[12] Nea96]
DONALD NEAL: The Harvest Object Cache in New Zealand. 1996.
http://www.w3.org/pub/WWW/Journal/3/s3.neal.html

[13] Net97]
Netscape Communication Corporation: Netscape Proxy Server. 1997.
http://www.netscape.com/comprod/proxy_server.html

[14] DP96]
HENRIK F. NIELSEN, u.a.: Network Performance Effects of HTTP/1.1, CSS1, and PNG. 14.02.1997.
http://www.w3.org/pub/WWW/TR/NOTE-pipelinig-970213

[15] Pra95]
HELMUT PRALLE: Projektübersicht: Konzeption einer Cache-Server-Infrastruktur auf dem Wissenschaftsnetz. Auszug aus dem Projektantrag. 1995/1996
http://www-cache.dfn.de/cache/Project.html

[16] San96]
LISA SANGER: Caching on the Internet. 1996.
http://www.seamless.com/eric/cache.html

[17] Sch96]
OLIVER SCHADE: Gemeinsam sind wir schneller. Hierarchischer Cache-Verbund mit Harvest. In iX 8/1996.

[18] WA97]
ROLAND P. WOOSTER, MARC ABRAMS: Proxy Caching That Estimates Page Load Delays. 1997. 6. International World Wide Web Conference.
http://atlanta.cs.nchu.edu.tw/www/PAPER250.html-admin.htm

[19] Wil96]
STEPHEN WILLIAMS, u.a.: Removal Algorithms in Network Caches for World Wide Web Documents. Stanford, Aug. 1996.
http://ei.cs.vt.edu/~succeed/96sigcomm/