Seminararbeit

Cachingverfahren und Proxies

Jan Kästle

Seminar

Content Delivery Networks

Universität Mannheim

Lehrstuhl Praktische Informatik IV: Prof. Dr. Wolfgang Effelsberg

Betreuer: Christian Liebig

30. November 2004

Inhaltsverzeichnis

1 Einleitung - Die Rolle von Cachingverfahren in CDN 1

2 Verschiedene Webinhalte und Möglichkeiten sie zu cachen 2

2.1 Statische Inhalte 2

2.2 Dynamische Inhalte 2

2.3 Streaming Media 3

2.4 Transaktionen 4

3 Grundlegende Cachingverfahren 4

3.1 Caching auf unterschiedlichen Ebenen 4

3.2 Ersetzungsstrategien 5

3.3 Konsistenz und Proaktives Caching 7

3.4 Prefetching 9

4 Architekturen von Caches 9

4.1 Verteiltes Caching 10

4.2 Hierarchisches Caching 10

4.3 Cache Location in Content Delivery Networks 11

5 Zusammenfassung 12

1 Einleitung - Die Rolle von Cachingverfahren in CDN

An das Internet werden durch den zunehmend kommerziellen Gebrauch und die Bereitstellung von Media-Inhalten immer höhere Anforderungen gestellt. Performance-Schwierigkeiten, Probleme mit der *best-effort* Charakteristik des Internets und das Verlangen der Anwender nach garantierten Bandbreiten (*Quality of Service*) haben die Forschung und Entwicklung neuer Netzwerkarchitekturen und Technologien vorangetrieben. Eine viel versprechende Entwicklung in diese Richtung sind Content Delivery Netzwerke (CDNs). In CDNs werden Kopien der anzubietenden Inhalte geographisch, in direkter Nähe der Anwender, verteilt, um einen schnellen und zuverlässigen Zugriff von überall her zu gewährleisten. Die Inhalte werden dafür an die Ränder der Netzwerke gebracht. Dies führt zu einer besseren Skalierbarkeit. Im Idealfall können alle Anfragen eines Anwenders von einem Server in seinem Heim-Netzwerk bedient werden. So brauchen die angefragten Inhalte nicht das ganze Netzwerk zu durchqueren, sondern nur den Teil zwischen Anwender und dem Rand des Netzwerkes. [1]
Ein zentraler Bestandteil eines CDN ist eine Caching-Architektur, um die Inhalte effektiv bereitstellen zu können. Man kann ein CDN auch als ein Netzwerk von Caches bezeichnen. Im Gegensatz zum herkömmlichen Caching (*content-blind caching*) werden beim CDN-Caching die Eigenschaften der Inhalte ausgenutzt (*content-aware caching*), um sie effizienter cachen zu können. Wichtig sind auch Mechanismen, die automatisch bestimmen, ob sich der Inhalt auf dem Original-Server geändert hat und Updates durchzuführen sind. Herkömmliche Verfahren gewährleisten dies nicht. Neue Verfahren müssen Konsistenz garantieren können. Ersetzungsstrategien ermöglichen es, den begrenzten Speicher effizienter auszunutzen. Caching-Architekturen erweitern die Möglichkeiten individuell agierender Caches und lassen diese kooperieren, um komplexe Inhalte wie Streaming Media zur Verfügung stellen zu können.
Zuerst werden nun verschiedene Typen von Inhalten vorgestellt und erläutert, ob und wie gut sie sich zum Caching eignen und welche Methoden es für diese verschiedenen Inhalte gibt. Dann werden einige grundlegende Cachingverfahren vorgestellt. Dazu gehören Ersetzungsstrategien und Konsistenzhaltung der Inhalte auf dem Cache. Eng verbunden mit Konsistenzfragen ist das proaktive Cachen, das Inhalte aktiv auf die Caches schiebt (*Push*-Verfahren statt dem traditionellen *Pull*-Verfahren). Darauf werden verschiedene Caching-Architekturen vorgestellt, inwiefern sie Anwendung für Content Delivery Netzwerke finden und wie sie es möglich machen die Performance zu verbessern. Am Ende wird schließlich auf die Frage eingegangen, wo Replikas im Netz optimal platziert werden.

2 Verschiedene Webinhalte und Möglichkeiten sie zu cachen

Im Folgenden werden nun verschiedene Arten von Webseiten und -objekten vorgestellt, die vom Cachen profitieren könnten. Dabei wird darauf eingegangen wie gut sich die jeweiligen Inhalte cachen lassen und welche Schwierigkeiten sich dabei stellen.

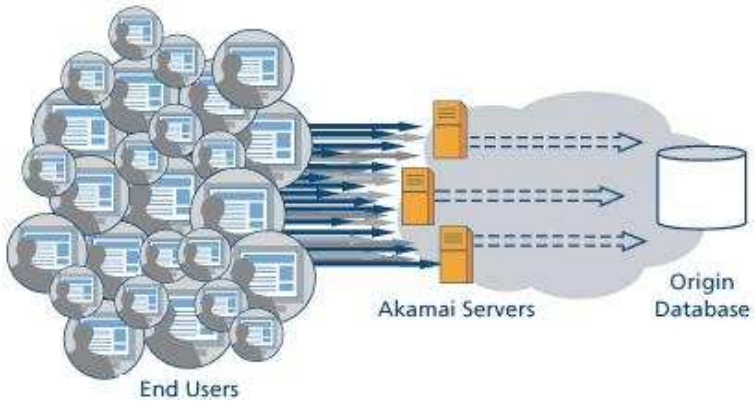
2.1 Statische Inhalte

Webobjekte, die sich nur selten ändern, fallen in diese Kategorie. Beispiele hierfür sind statische HTML-Seiten oder Bilder. Diese Inhalte lassen sich gut cachen, und ihre Aktualität kann man relativ leicht mit traditionellen Methoden sicherstellen, wie es in Kapitel 3.3 gezeigt wird.

2.2 Dynamische Inhalte

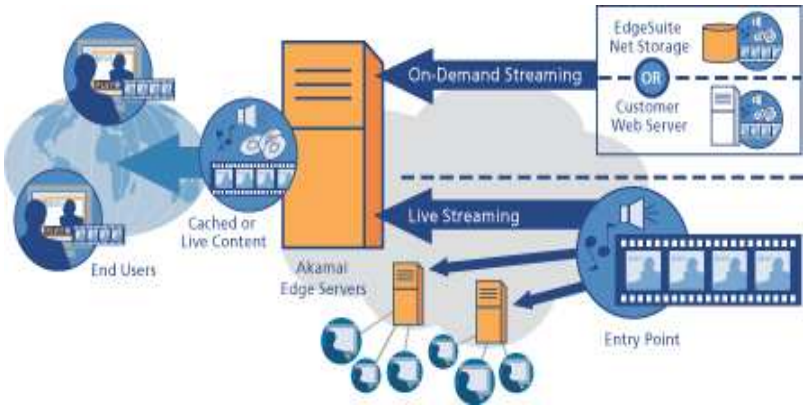
Als dynamische Inhalte bezeichnet man Inhalte, die der Webserver bei jeder Anfrage dynamisch generiert. Dies geschieht z.B. unter Verwendung einer Session ID in der URL oder mit Cookies. Der Proxy erkennt diese Inhalte anhand der Dateiendung. Dynamische Inhalte stellen bereits einen großen Anteil der Webinhalte dar, und es ist angesichts der zunehmenden Bedeutung von Webservices und anderen Webtechnologien zu erwarten, dass sich dieser Anteil weiterhin erhöhen wird. Deshalb ist es wichtig, auch diese Inhalte cachen zu können. Da viele dynamisch generierte Seiten einen großen statischen Anteil haben, ist eine Idee, die Seiten in einen dynamischen und statischen Teil zu zerlegen. Dies kann mithilfe von Templates des Servers geschehen. Somit kann der Proxy die statischen Teile cachen. Bei einer Anfrage wird dann nur der dynamische Teil vom Server abgefragt und mit dem statischen Teil vereint, bevor das ganze Dokument an den Anwender gesendet wird. [2]
Im aktiven Caching stellt der Server mit den Webseiten Applets bereit. Der Cache soll dann bei einem Cache Hit dieses Applet ausführen, um die Seite für den Anwender aufzubereiten, ohne direkt auf den Server zuzugreifen. Diese Methode kann den Anteil von cachebaren Inhalten erhöhen; der Nachteil besteht aber in der höheren CPU Belastung des Caches. [2]
Noch weiter geht der Ansatz, ganze Applikationen zu replizieren. Dies kann auf Anfrage oder automatisch geschehen. Probleme können sich dabei durch die höhere Rechenintensität auf dem Cache ergeben. Um Datenkonsistenz und Integrität nicht zu gefährden, sind nur bestimmte Anwendungen dafür geeignet.

Der CDN Hersteller Akamai unterstützt zum Beispiel das Ausführen von Java Server Pages, Servlets oder Java Beans auf den Caches. Die Schicht auf den Caches besteht aus Präsentations- und Business-Komponenten, die sich nur selten ändern, dazu gehören u.a. Produktkataloge oder Shopping Carts. [3]



2.3 Streaming Media

Da Multimedia Anwendungen im Internet immer beliebter sind, steigt auch die Bedeutung, Streaming Media effizient zu cachen. Es handelt sich dabei um große Audio oder Video Objekte. Diese könnte der Proxy theoretisch einfach als große Dateien behandeln. Wegen Speicherbeschränkungen ist dies aber nicht praktikabel. Ein Ansatz ist es, jeweils nur die ersten Frames bereitzustellen. Sobald eine Anfrage eingeht wird der Proxy dann die nächsten Frames vom Server abrufen und für den Client bereithalten. Dieses Verfahren gleicht die Variabilität der Netzwerkressourcen aus. [4]
Eine Methode, die die Caching-Architektur besser ausnutzt, zeigen die Autoren von [1] auf. Media Dateien werden in Segmente aufgeteilt. Benachbarte Caches teilen Segmente unter sich auf und tauschen sie bei Bedarf aus. Die Kooperation zwischen den Caches kann in einer hierarchischen oder verteilten Struktur stattfinden, hierauf wird in Kapitel 4 noch näher eingegangen. Beim Live Streaming kann die Baumstruktur des CDN ausgenutzt werden, um den Stream effizient zu verteilen. Das CDN von Akamai verwendet genau diese Architektur für Streaming Media:



2.4 Transaktionen

Um Transaktionen zu cachen, muss der Proxy mit Transaktionsfähigkeiten ausgestattet sein. Eine besondere Schwierigkeit beim Cachen von Transaktionen ist die Datenkonsistenz und -integrität. In existierenden Web Caching Systemen werden Transaktionen, die nur Leseoperationen beinhalten, auf Ebene des Caches ausgeführt. Alle anderen Transaktionen verbleiben weiterhin beim Server. [2]

3 Grundlegende Cachingverfahren

3.1 Caching auf unterschiedlichen Ebenen

Caching findet auf den verschiedensten Ebenen statt. Im Browser werden unter Verwendung des lokalen Dateisystems Seiten gespeichert, um die Wartezeit bei wiederholtem Zugriff zu verkürzen. Hiervon profitiert nur der lokale Anwender. Proxies von ISPs und großen Organisationen cachen anwendernah Webseiten und andere Objekte. Haben die Anwender ein ähnliches Zugriffsmuster, können Anfragen lokal bedient werden. Beim sogenannten Reverse Proxy Caching stellt ein Anbieter Caches nahe seines Webserver auf. Dadurch erreicht er eine bessere Skalier- und Verfügbarkeit. Die Verbesserungen sind dann aber nur auf diesen Anbieter beschränkt.
Content Delivery Netzwerke fallen in den Bereich des anwendernahen Caching.

3.2 Ersetzungsstrategien

Da ein Cache keinen unbegrenzten Speicherplatz zur Verfügung hat, sind effiziente Ersetzungsstrategien erforderlich, um zu bestimmen, welche Objekte ersetzt werden können, wenn neu hinzukommende Objekte nicht mehr in den Speicher passen. Die Ersetzungsalgorithmen lassen sich in drei Kategorien einteilen. Zum einen hat man Algorithmen übernommen, die in der Hauptspeicherverwaltung eingesetzt werden. *Least Recently Used* (LRU) ersetzt das Objekt, auf das am längsten nicht zugegriffen wurde. Dabei wird eine Liste der Objekte, geordnet nach dem letztem Zugriff, bereitgehalten. *Least Frequently Used* (LFU) wählt das Objekt aus, das am seltensten benutzt wurde. Dazu zählt man für jedes Objekt die Anzahl der Zugriffe. In dieser Form hat der Algorithmus einige Nachteile. So werden alte Referenzen nie vergessen (das kann man lösen indem regelmäßig die Zähler vermindert werden), und neue Objekte haben einen niedrigen Zähler und sind deshalb Kandidaten zum Ersetzen.
Zum anderen werden Objekte anhand bestimmter Merkmale ausgewählt, z.B. werden große Objekte eher ersetzt als kleine oder man zieht die Zeit heran, die benötigt wird, um das Objekt wieder vom Server zu beziehen.
Die dritte Klasse von Algorithmen versucht aus mehreren der oben genannten Merkmale zu bestimmen, welches Objekt ersetzt werden kann.
Performancemessungen ermöglichen es, Ersetzungsalgorithmen zu bewerten. Das hilft auch, neue effiziente Verfahren zu entwickeln. Dazu werden Simulationen, die typische Webobjekte und Netzwerkumgebungen modellieren, oder Trace Files - das sind echte Logdateien von Webservern und Proxies - eingesetzt. Eine der wichtigsten Performancemaße ist die Trefferquote. Sie wird folgendermaßen berechnet:

$$Trefferquote = \frac{Anzahl\ der\ Treffer}{Gesamtzahl\ der\ Anfragen\ an\ den\ Cache}$$

Auch wichtig, aber schwerer zu messen ist die vom Anwender wahrgenommene Wartezeit.
Tendenziell können Ersetzungsalgorithmen, die die Größe der Objekte mit einbeziehen, eine bessere Performance erzielen. [5]
[6] beschreibt eine Abwandlung des LRU-Algorithmus, der die Größe miteinbezieht. Dieser Algorithmus soll hier etwas ausführlicher dargestellt werden. Dazu sind zuerst einige Definitionen notwendig. Angenommen wir haben N Objekte und Objekt i hat die Größe S_i. Ein Zähler wird bei jedem Zugriff auf ein Objekt hochgezählt. Die Menge der Objekte, die zum Zeitpunkt k im Cache ist als $C(k)$

bezeichnet. ik bezeichnet das Objekt. Falls ik zum Zeitpunkt $(k-1)$ nicht im Cache ist muss man entscheiden, welche Objekte ersetzt werden. R bezeichne den benötigten Speicherplatz. Die Entscheidungsvariable y_i ist 1, falls Objekt i ersetzt werden soll, 0 andernfalls. $\triangle T_{ik}$ bezeichnet die Anzahl der Zugriffe seit dem letzten Zugriff auf i. $\frac{1}{\triangle T_{ik}}$ stellt die sogenannte *dynamische Häufigkeit* von Objekt i zum Zeitpunkt k dar. Die Summe der dynamischen Häufigkeiten der zu löschenden Objekte soll nun so klein wie möglich sein. Daraus ergibt sich folgendes Modell:

$$\text{Minimiere} \sum_{i \in C(k)} y_i / \triangle T_{ik}$$
$$\text{so dass} \sum_{i \in C(k)} S_i y_i \geq R$$
$$\text{wobei } y_i \in \{0, 1\}$$

Dieses Problem ist als das Rucksackproblem bekannt; dieses ist NP-vollständig. Es gibt dafür jedoch einen effizienten heuristischen Algorithmus. In diesem Fall würden die Objekte nach dem Produkt von Größe und dynamischer Häufigkeit $S_i * \triangle T_{ik}$ sortiert (Verhältnis von Kosten zu Größe). Es wird dann solange das Objekt mit der höchsten Häufigkeit entfernt, bis die Grenze R erreicht ist. Dieses Verfahren nennen die Autoren *Size Adjusted LRU*. Da der Vergleich der Produkte aber zu rechenintensiv ist, haben die Autoren eine Abwandlung entwickelt, die effizienter ist: das *Pyramidal Selection Scheme*. Dabei teilt man die Objekte je nach Größe in Klassen ein. Alle Objekte der Gruppe i haben eine größe zwischen 2^{i-1} und $2^i - 1$. Also gibt es $N = \lceil \log(M+1) \rceil$ verschieden Gruppen von Objekten, wobei M die Größe des Cache bezeichnet. Eine getrennte LRU Liste für jede Gruppe verwaltet die in sie fallenden Objekte. Wenn ein Objekt aus dem Cache gelöscht werden muss, werden nur die $S_i * \triangle T_{ik}$ Werte von den seltensten genutzten Objekten in jeder Gruppe verglichen. Da die Anzahl der Listen logarithmisch ist im Verhältnis von maximaler zu minimaler Größe jedes Objekts, folgt, dass der Algorithmus im schlechtesten Fall um einen logarithmischen Faktor schlechter ist, als die Kosten einer einzelnen LRU Liste. Resultierend aus diesem Mechanisums wird im schlechtesten Fall ein Objekt ausgesucht, dessen Produkt $S_i * \triangle T_{ik}$ maximal um den Faktor $\frac{1}{2}$ vom optimalen Wert abweicht. [6]

Da die Zugriffszeiten nicht linear mit der Größe steigen, sondern auch von der Netzwerkumgebung abhängen, kann hierfür ein weiterer Faktor miteinbezogen werden. Weiterhin könnte man Objekte, die bereits nicht mehr aktuell sind oder nur noch eine kurze Gültigkeit haben, bei der Ersetzung vorziehen.

Eng im Zusammenhang mit der Frage, welche Objekte ersetzt werden sollen, steht die Frage, welche

Objekte man überhaupt in den Speicher aufnimmt (Admission Control). Ein Verfahren überprüft zum Beispiel wie hoch der erwartete Beitrag des neuen Objekts zur Trefferquote ist, im Vergleich zu den Objekten die zum Ersetzen ausgesucht wurden. Der Cache nimmt das neue Objekt dann nur auf, wenn dieser Beitrag mindestens genauso hoch ist wie derjenige der zu ersetzenden Objekte. [2]

3.3 Konsistenz und Proaktives Caching

Wenn sich die Objekte auf dem Server nie ändern würden, wären die im Cache gespeicherten Seiten immer aktuell und müssten nie neu angefordert werden. Da es aber gerade auch bei dynamischen Inhalten zu häufigen Änderungen kommt, sind intelligente Strategien erforderlich, um Konsistenz zu gewährleisten. Dabei gibt es grundsätzlich zwei Möglichkeiten: die erste basiert auf Aktionen der Clients (Caches); die zweite bezieht die Server mit ein. Die client-basierten Verfahren nutzen die im HTTP Header angegebenen `max-age` und `expires` Felder. Der Cache kann daraus time-to-live (TTL) Informationen erstellen. Diese wird dann verwendet, um zu bestimmen, ob das Objekt noch gültig ist. Dabei gibt es zwei Ansätze: im proaktiven Ansatz wird in periodischen Intervallen validiert und gegebenenfalls neu angefordert, im reaktiven Ansatz wird das Objekt nur auf Anfrage eines Anwenders validiert. Die Vor- und Nachteile liegen auf der Hand. Der erste Ansatz verkürzt die Wartezeit des Users, der zweite Ansatz bietet absolute Konsistenz. Auch kann zu häufiges validieren die Netzwerkauslastung unnötig erhöhen. Wenn es keine ausreichende TTL Informationen gibt, kann die TTL auf einen bestimmten Prozentsatz der Zeit gesetzt werden, seit der das Objekt das letzte Mal geändert wurde. Hier wendet man die Intuition an, dass sich ein Objekt, das sich lange nicht geändert hat, wahrscheinlich auch nicht bald ändern wird. Absolute Konsistenz ist dann aber nicht gewährleistet. [2]
HTTP 1.1 bietet explizite Unterstützung für das Caching; so gibt es Anweisungen an den Proxy, wie `must-revalidate` oder `no-cache`, die beim Cachen helfen sollen.
Ein server-basierter Ansatz ist das Konzept der Callbacks. Wenn ein Client ein Objekt vom Server anfordert, erhält er ein *Callback Promise*. Wenn sich das Objekt ändert, informiert der Server den Client. Der Client invalidiert das entsprechende Objekt und muss es bei der nächsten Anfrage neu anfordern. Für dieses Verfahren ist es erforderlich, dass der Content Server für alle Objekte eine Liste bereithält, welche Clients bei einer Änderung zu informieren sind. Die Vorteile dieser Technik sind absolute Konsistenz, da der Server genau weiß, wann die Objekte geändert wurden, und ein

reduziertes Netzwerkaufkommen, da Nachrichten, die im Zusammenhang mit der Konsistenz stehen, nur bei einer tatsächlichen Änderung versendet werden. Dieser Ansatz skaliert aber nicht, weil der Aufwand für den Server zu groß ist, für jedes Objekt eine Liste zu verwalten. Zudem kann es bei Updates zu plötzlich hohem Netzwerktraffic kommen, wenn viele Caches gleichzeitig versuchen, die neue Version abzurufen. [2]

Ein weiteres Verfahren, das absolute Konsistenz bietet, ist das Konzept der *Leases*. Ein Lease ist ein Versprechen des Servers, dem Proxy eine Invalidierungs-Nachricht zu schicken, wenn sich das Objekt in einer im Lease festgelegten Zeit ändert. Vorteile dieses Verfahren sind, dass der Server nur eine relativ kurze Liste von Proxies verwalten muss, die explizit Interesse an diesem Objekt angegeben haben, und dass Leases eine Art Time-Out Mechanismus zur Verfügung stellen. Denn, wenn ein Proxy aufgrund von Netzwerkproblemen nicht erreicht werden kann, bekommt der Server keine Bestätigung seiner Invalidierungs-Nachricht, wenn er das Objekt ändern will. Er wartet also bis das Lease abgelaufen ist, dann kann er das Objekt ändern, ohne die Konsistenz zu gefährden.

Die eben genannten Methoden ignorieren die Eigenschaften von Content Delivery Netzwerken. Wenn Objekte aktualisiert werden, geschieht dies hier immer auf Unicast Basis. Die Struktur des Content Delivery Netzwerks lässt sich durch das folgende Verfahren besser ausnutzen.

Im Push-Verfahren (auch pre-population genannt) wird der Inhalt bei einer Änderung oder wenn ein neues Objekt hinzukommt, automatisch an die Caches verteilt. Dadurch ist es nicht nötig, Objekte auf Gültigkeit zu prüfen; die Caches können sich darauf verlassen, bei einem Update die aktuelle Version vom Server zu bekommen. Diese Methode hat einen hohen Speicherbedarf und führt zu hohem Netzwerkaufkommen, wenn Objekte öfter geändert werden als auf sie zugegriffen wird. [7]

In [8] stellen die Autoren eine Mischung aus dem Push-Verfahren und dem Callback-Verfahren vor. Der Original-Server bestimmt abhängig von Zugriffsstatistiken, welche Methode verwendet wird. Wenn ein Objekt eine geringe Zugriffsfrequenz hat, verwendet er das Callback Verfahren; bei hoher Zugriffsfrequenz kommt das Push Verfahren zur Anwendung. Im Zusammenhang mit dem Push-Verfahren stellt sich die Frage, wie die Inhalte auf Caches verteilt werden. Die eine Möglichkeit ist es, in N Unicast Verbindungen die Objekte zu übertragen. Diese Technik nutzt die Eigenschaft des Content Delivery Netzwerkes nicht aus. Die bessere Möglichkeit ist es, auf Anwendungsebene einen Multicast Baum aufzubauen und die Objekte über den Baum zu übertragen, dies ist deutlich effizienter als über Unicast Verbindungen, jedoch nicht so effizient wie natives Multicast. [9]

3.4 Prefetching

Kommt die Push-Methode nicht zum Einsatz, wie im traditionellen Caching, ist die Verwendung von Caches nur auf Seiten, auf die bereits zugegriffen wurde, beschränkt. Greift ein Anwender vor allem auf neue, d.h. nicht im Speicher des Cache befindliche Seiten zu, profitiert er nicht vom Caching. Abhilfe schafft das Prefetching. Die Schwierigkeit beim Prefetching ist, vorherzusagen, auf welche Seiten in Zukunft zugegriffen wird. Eine komplizierte Vorhersagungsstrategie aufgrund ausführlicher Information über den Anwender und Wissen über mögliche Seiten, auf die zugegriffen werden könnte, kommt aus Effizienzgründen nicht in Frage und ist auch nicht unbedingt notwendig. In der Literatur werden drei verschiedene Strategien unterschieden. Die struktur-basierten Ansätze ziehen Rückschlüsse aus den abgerufenen Webseiten, zum Beispiel aus Links in den Dokumenten. Ein einfacher struktur-basierter Ansatz wäre es, alle Links in einem HTML-Dokument aufzulösen und bereitzuhalten. In Weiterentwicklungen dieser Ansätze können der Typ des Objekts und Wartezeitstatistiken des Webservers miteinbezogen werden. History-Basierte Ansätze versuchen aus vergangenen Anfragen zukünftige Anfragen vorherzusagen. Als Beispiel wäre hier der Top10-Algorithmus zu nennen. Er verwendet eine Liste der in der Vergangenheit beliebtesten Dokumente, um zukünftige Webzugriffe vorherzusagen. Fortgeschrittene Methoden fallen meist in die Kategorie der markov-basierten Methoden. Diese versuchen anhand von Markov-Modellen zukünftige Zugriffe vorherzusagen. Prefetching kann auf Ebene des Browsers stattfinden, zum Beispiel wenn der Anwender gerade untätig ist. [2]

4 Architekturen von Caches

Die Vorteile des Caching lassen sich bei unabhängig voneinander agierenden Caches nur begrenzt ausnutzen, denn Caches sind in ihren Ressourcen beschränkt. Kooperierende Caches nutzen ein ganzes Netzwerk von Caches, um Anfragen der Anwender zu erfüllen. Durch eine größere Gruppe von Anwendern lässt sich auch der Effekt von Caches steigern und die Last besser verteilen. Netzwerke von Caches verbessern auch die Fehlertoleranz und die Zuverlässigkeit. Sie sind zentraler Bestandteil von Content Delivery Netzwerken.

4.1 Verteiltes Caching

Im verteilten Caching arbeiten alle Caches auf derselben Ebene, d.h. gleichberechtigt. Die Caches befinden sich an den Rändern der Netzwerke. Da es keine übergeordneten Caches gibt, müssen andere Wege gefunden werden, die Inhalte zwischen den einzelnen Caches aufzuteilen. Ein Cache leitet bei einem Cache-Miss die Anfrage mittels des *Inter Cache Protocol* (ICP), das auch in CDNs zum Einsatz kommt, an seine Nachbarn weiter. Dadurch kommt es zu einem erheblichen Kommunikations-Overhead. Dies kann sehr langsam sein, wenn auf alle Antworten gewartet wird. Außerdem skaliert dieser Ansatz nicht. [10]
Die Herausforderung ist es, diejenigen Nachbarn effizient zu bestimmen, die das gesuchte Objekt haben. Dafür gibt es verschiedene Methoden. Im Hash-basierten Caching bildet eine Hash-Funktion die URL auf einen Proxy ab. Bei einem Cache Miss wendet der Proxy die Funktion auf die URL des Webobjekts an. Der Rückgabewert der Funktion ist ein Proxy, der dieses Objekt bereithält. Der Vorteil dieser Methode ist der geringe Kommunikations-Overhead. Außerdem müssen auch keine Kopien vorhanden sein. Alle Proxys müssen aber dieselbe Hash-Funktion verwenden, was in großen Umgebungen zu Problemen führen kann, wenn die Funktion geändert werden muss. [2]
In einem anderen Ansatz können die Caches eine Zusammenfassung (*digest*) der Inhalte kooperierender Caches speichern und so auch das Polling vermeiden. Für die Verteilung dieser Zusammenfassung ist dann auch eine Verteilungsstruktur notwendig. Bei einem Cache-Miss kann der Proxy durch diese Zusammenfassung bestimmen, welcher Nachbar das fehlende Objekt hat.
Vorteile des Verteilten Caching ist der geringere Speicherbedarf im Vergleich zum hierarchischen Caching, da nur jeweils eine Kopie gespeichert werden muss. Beim hierarchischen Caching werden im Gegensatz dazu mehrere Kopien desselben Objekts auf verschiedenen Ebenen gespeichert. Verteilte Caches sind robuster bei Fehlern, zudem kann die Last besser verteilt werden und Engpässe lassen sich leichter vermeiden.

4.2 Hierarchisches Caching

Im hierarchischen Caching sind Caches in Form eines Baumes angeordnet. Kann ein Cache eine Anfrage nicht erfüllen, leitet er sie wiederum an den übergeordneten Knoten weiter. Falls dieser die Anfrage auch nicht erfüllen kann, leitet er sie an seinen übergeordneten Knoten weiter. Wenn die Anfrage entweder bei einem Cache oder spätestens beim Original Web Server beantwortet werden

kann, geht die Antwort den gleichen Weg in umgekehrter Reihenfolge nach unten. Dabei verbleibt in jedem Cache eine Kopie des angefragten Objekts. Diese Struktur ist relativ einfach, da ein Cache vorher nicht ermitteln muss an welchen Server die Anfrage weitergeleitet werden muss. Zudem ist der Overhead für die Kommunikation klein. Diese Organisationsform ist sehr beliebt bei großen ISPs, hat aber einige Nachteile: [10]

- Jede Ebene von Caches vergrößert das Delay.
- Auf verschiedenen Ebenen werden identische Kopien der Objekte gehalten.
- Caches, die nahe der Wurzel liegen, können leicht zu Engpässen werden.

4.3 Cache Location in Content Delivery Networks

In der Literatur herrscht Einigkeit darüber, dass Caches in Content Delivery Netzwerken an den Rändern der Netzwerke, nahe den Usern, platziert werden sollen. [2], [11]
Im Idealfall würde jeder Anwender die gewünschten Inhalte auf einem Cache innerhalb seines bzw. des Netzwerkes, seines ISPs finden.
Für eine genau Bestimmung der Standorte wurden verschiedene Verfahren entwickelt. Gemeinsames Ziel der Verfahren ist es, Replikas so zu platzieren, dass die Wartezeit der User, die Anzahl der Replikas und die Netzwerkauslastung minimal sind. Es ist zwischen den statischen und den dynamischen Verfahren zu unterscheiden. Die Eingabe eines statischen Verfahren ist eine gegebene Netzwerktopologie, eine Menge von CDN Servern sowie Daten über das Netzwerkaufkommen. Als Zielgröße werden entweder die Kosten der CDN-Infrastruktur für eine vorgegebene User Performance minimiert oder die vom Anwender wahrgenommene Wartezeit wird unter einer gegebenen Infrastruktur minimiert. Beschränkungen sind Systemressourcen, wie Speicherplatz oder die Kapazität eines Cache an parallelen Verbindungen. Die Autoren von [12] formulieren das Cache-Location Problem für CDNs als *Mixed Integer Linear Problem* (MILP). Ziel ist es, den Delay der Clients zu minimieren. Gelöst werden kann das Problem dann mit einem MILP Solver, jedoch nur für Topologien in einer Größenordung von einigen hundert Knoten. Für größere Topologien müssen Heuristiken verwendet werden.
Andere Ansätze für das Platzieren von Replikas verwenden Greedy Algorithmen.
Die dynamischen Verfahren versuchen die sich verändernde Struktur des Internets mit einzubeziehen.

Im in [7] als adaptives Caching bezeichneten Verfahren gibt es mehrere verteilte Caches, die sich, je nach Bedarf, dynamisch einer Cache Gruppe anschließen oder sie verlassen.

5 Zusammenfassung

Diese Arbeit gibt einen vertiefender Einblick in grundlegende Cachingverfahren. Sie zeigt auf, welche Probleme beim Cachen der verschiedenen Inhalte auftreten und wie diese gelöst werden können, so dass alle Inhalte von den Vorteilen des Cachen profitieren. Verschiedene Verfahren, die die Daten auf dem Cache konsistent halten, wurden dargestellt. Es wurde deutlich, dass ein wesentliche Bestandteil von CDNs das proaktive Cachen ist, bei dem Inhalte aktiv an die Caches verteilt werden, statt nur auf User Anfragen zu reagieren. Verschiedene Ersetzungsstrategien, die es ermöglichen trotz Speicherbeschränkungen gute Ergebnisse zu erzielen, wurden erörtert. Zuletzt wurde auf die verschiedenen Architekturen von Caching Systemen eingegangen.
Obwohl bewährte Cachingverfahren teilweise für CDNs übernommen werden konnten, stellen CDNs neue Herausforderungen an das Cachen und erfordern intelligente Methoden, wenn es zum Beispiel um das Cachen von dynamischen Inhalten oder Streaming Media geht. Die Forschung in diesem Bereich ist noch lange nicht abgeschlossen.

Literatur

[1] N. Bartolini, E. Casalicchio and S. Tucci. A walk through Content Delivery Networks. In Lecture Notes in Computer Science, Volume 2965 / 2004, S. 1-25, 2004.

[2] D. Zeng, F. Wang. Effi cient Web Content Delivery Using Proxy Caching Techniques. In IEEE Transactions on Systems, Man and Cybernetics, Vol. 34, No. 3, S. 270-280, 2004.

[3] Edge Computing, URL: www.akamai.com, 17.11.2004.

[4] S. Sen, J. Rexford und D. Towsley. Proxy prefi x caching for multimedia streams. In Proc. IEEE INFOCOM, New York, 1999.

[5] M. Abrams, C. Standridge, G. Abdulla, S. Williams and E. Fox, Caching Proxies: Limitations and Potentials, Proc. Fourth Int'l World Wide Web Conf., Boston, 1995.

[6] C. Aggarwal, J. Wolf und P. Yu. Caching on the World Wide Web. IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No.1, 1999.

[7] G. Barish, K. Obraczka. World Wide Web Caching: Trend and Techniques. In IEEE Communications Magazine, 2000.

[8] Z. Fei, S. Bhattacharjee, E. Zegura, and M. H. Ammar. A novel server selection technique for improving the response time of a replicated service. In Proceedings of Infocom 98, 1998.

[9] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In Proceedings of ACM Sigmetrics, June 2000.

[10] P. Rodriguez, C. Spanner and E. Biersack. Analysis of Web Caching Architectures: Hierarchical and Distributed Caching. IEEE/ACM Transactions on Networking, Vol. 9, No. 4, 2001.

[11] Z. Su, J. Katto, Y. Yasuda. Replication Algorithms to Retrieve Scalable Streaming Media over Content Delivery Networks. Proc. of the 5th ACM SIGMM international workshop on Multimedia information retrieval, Berkeley, California, USA, 2003.

[12] A. Wierzbicki. Internet Cache Location and Design of Content Delivery Networks, Networking 2002 Workshops, E. Gregori et al.(Eds.), Springer-Verlag Berlin Heidelberg, S. 69-82, 2002.