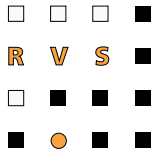
Planung und Einsatz lokaler Cache-Server

Studie

Oktober 2001, C. Grimm



Lehrgebiet Rechnernetze und Verteilte Systeme (RVS)
Prof. Dr.-Ing. Helmut Pralle

Universität Hannover

Inhalt

1 Einleitung 2
1.1 Ziel der Studie 2

2 Grundlagen 4
2.1 Architektur des World Wide Web 4
2.1.1 Semantische Komponenten 4
2.1.2 Software-Komponenten 5
2.1.3 Das Hypertext Transfer Protocol 6
2.2 Caching im World Wide Web 10
2.2.1 Nutzen von Caching 10
2.2.2 Aufbau von Cache-Servern 11
2.2.3 Performance von Cache-Servern 12
2.2.4 Konsistenz zwischen Cache-Server und WWW-Server 14
2.2.5 Umgang mit Cookies 18

3 Einsatz lokaler Cache-Server 21
3.1 Konfiguration der WWW-Browser 21
3.1.1 Selbständige Konfiguration durch die Nutzer 21
3.1.2 Nutzung einer zentralen Konfigurationsdatei 21
3.1.3 Transparente Konfiguration 22
3.2 Strategien für die Nutzung 22
3.2.1 Freiwillige Nutzung 22
3.2.2 Erzwungene Nutzung 23
3.2.3 Transparente Umleitung 24
3.3 Dimensionierung lokaler Cache-Server 26
3.4 Kalkulation der notwendigen Einsparungen 29
3.5 Verfügbare Produkte 33

4 Zusammenfassung 34

Literaturverzeichnis 36

Das Vorhaben wurde aus Mitteln des Bundesministeriums für Bildung, Wissenschaft, Forschung und Technologie (BMBF) durch den Verein zur Förderung eines Deutschen Forschungsnetzes e. V. (DFN-Verein) finanziert.

Kapitel 1

Einleitung

Cache-Server speichern abgerufene Inhalte aus dem World Wide Web lokal in der Nähe der Nutzer und stellen die Inhalte bei erneutem Abruf unmittelbar zur Verfügung. Die Vorteile des Caching im World Wide Web kommen gleichzeitig verschiedenen an den Kommunikationsabläufen beteiligten Gruppen zugute:

- Nutzer: Objekte aus dem Cache-Server können in der Regel mit deutlich höherer Datenrate übertragen werden als vom originalen WWW-Server. Dadurch werden Wartezeiten reduziert, so dass die Nutzbarkeit des World Wide Web steigt. Weiterhin verringern sich durch kürzere Sitzungsdauern anfallende zeitabhängige Verbindungsentgelte.

- Netzbetreiber: der WWW-Verkehr auf den Leitungen im Kernnetz und an den Übergangspunkten in andere Netze wird verringert.

- Betreiber von WWW-Servern: die Anzahl eingehender Requests, die von den Servern verarbeitet werden müssen, sinkt. Entsprechend reduziert sich auch das Datenvolumen des ausgehenden WWW-Verkehrs. Die Last auf den WWW-Servern sinkt.

1.1 Ziele der Studie

Seit der Einführung der volumen-basierten Tarifierung können Einrichtungen im Wissenschaftsnetz lokale Cache-Server mit dem Ziel einsetzen, durch Reduzierung des eingehenden Datenverkehrs in eine günstigere Kategorie eingeordnet zu werden. Der Aufwand für die Beschaffung und den Betrieb eines Cache-Servers kann den finanziellen Einsparungen gegenübergestellt werden. Der Entscheidung über den Einsatz eines lokalen Cache-Servers sollte daher eine Kalkulation vorausgehen, mit der die Dimensionierung und damit die Anschaffungskosten des Cache-Server berechnet werden können.

In dem ersten Teil der Studie wird zunächst eine Reihe von Grundlagen erläutert, die

das Caching im World Wide Web berühren. Diese Grundlagen bilden die Voraussetzung für die Beurteilung der Anforderungen und Leistungsgrenzen von Cache-Servern. Hierzu zählen:

- Unterstützung von Caching in HTTP,
- Aufbau und Performance von Cache-Servern,
- Prüfung der Aktualität der gespeicherten Objekte, Konsistenz.

In einem zweiten Teil der Studie werden praktische Aspekte bei dem Einsatz lokaler Cache-Server untersucht. Hierbei stehen zunächst verschiedene Strategien im Vordergrund, wie Cache-Server mit dem Ziel einer hohen Nutzung im lokalen Netz betrieben werden können. Anschließend wird beispielhaft die Dimensionierung eines lokalen Cache-Servers vorgestellt. Die dargestellten Berechnungen sind so angelegt, dass jede Einrichtung mit ihren Nutzungsdaten die Dimensionierung nachvollziehen kann. Die Studie schließt mit einer Betrachtung der zentralen Fragestellung, welche Einsparungen mit einem lokalen Cache-Server erreicht werden müssen, um bei einem vorgegebenen Datenvolumen in eine günstigere Kategorie eingeordnet werden zu können.

Kapitel 2

Grundlagen

In diesem Kapitel werden verschiedene Grundlagen des Caching im World Wide Web erläutert. Das Verständnis dieser Grundlagen bildet eine Voraussetzung für die Beurteilung der Leistungsfähigkeit und der Leistungsgrenzen von Cache-Servern. Ein häufiges Argument gegen den Einsatz von Cache-Servern ist die mangelnde Konsistenz der Daten gegenüber den originalen WWW-Servern. Daher liegt ein Schwerpunkt der Erläuterungen auf den Verfahren, nach denen Cache-Server die Aktualität der gespeicherten Objekte beurteilen können.

2.1 Architektur des World Wide Web

2.1.1 Semantische Komponenten

Das World Wide Web wurde als ein eigenständiges System zur Übertragung und Darstellung von hypertext-basierten Informationen über das Internet konzipiert. Die grundlegenden Anforderungen lassen sich in folgende semantische Komponenten unterteilen:

- Adressierung der Ressourcen: URI, URL, URN.
- Übertragung der Daten: HTTP.
- Repräsentation der Informationen: HTML, XHTML, SMIL.

Da die Eigenschaften von HTTP entscheidend für die Performance des World Wide Web sind und darüber hinaus wesentliche Bedeutung für das Caching besitzen, werden die wichtigsten Merkmale von HTTP in Kapitel 2.1.3 näher erläutert. Die übrigen semantischen Komponenten haben keine direkten Einfluss auf das Caching.

2.1.2 Software-Komponenten

Wie fast alle Dienste im Internet basiert auch das World Wide Web auf der Client-Server-Architektur. In Abbildung 2.1 sind die Komponenten dieser Architektur, erweitert durch einen Proxy- oder Cache-Server, dargestellt. Die Kommunikation zwischen den Komponenten erfolgt über ein festgelegtes Protokoll, im Fall des World Wide Web über HTTP. Die Bedeutung der einzelnen Komponenten wird im Folgenden kurz erläutert.

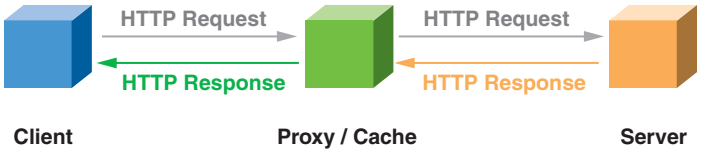


Abbildung 2.1 Client-Server-Architektur

WWW-Server

WWW-Server stellen die Informationen im World Wide Web zur Verfügung. Die prinzipielle Funktionsweise eines WWW-Servers zur Bearbeitung von HTTP-Requests läßt sich in vier Schritte unterteilen:

- Empfang des HTTP-Requests,
- Auswerten der URL nach dem gewünschten Objekt,
- Lesen des Objekts aus dem Dateisystem,
- Übertragen des Objekts in einer HTTP-Response.

WWW-Klienten

Die Darstellung der Informationen auf den WWW-Klienten erfolgt durch die so genannten WWW-Browser. Den Browsern fällt zunächst die zentrale Aufgabe zu, den HTML-Code zu interpretieren, referenzierte Inhalte von WWW-Servern abzurufen und alle Komponenten einer Seite vollständig und korrekt auf dem Bildschirm anzuzeigen. Weiterhin ermöglichen sie den Nutzern die Interaktion über Maus oder Tastatur.

Die Integration von FTP, Gopher, WAIS sowie Email und News macht aus WWW-Browsern heute universelle Programme, mit denen verschiedenste Informationsdienste im Internet genutzt werden können.

WWW-Proxies und WWW-Caches

Ein Proxy (engl. Stellvertreter) ist eine Instanz, die Zugriffe auf Objekte stellvertretend für einen oder mehrere Prozesse ausführt [Sha86]. Der Einsatz von Proxies kann aus verschiedenen Gründen notwendig sein. So verfügen Proxies häufig über besondere Zugriffsrechte, die den eigentlichen Prozessen nicht eingeräumt werden sollen. Nur durch den Übergang über den Proxy wird den Prozessen der Zugriff auf bestimmte Objekte ermöglicht. Ebenso lassen sich Proxies zur Anpassung von Nachrichten oder Funktionsaufrufen einsetzen.

Bezogen auf Client-Server-Architekturen stellt ein Proxy ein eigenständiges System dar, das zwischen Klienten und Server geschaltet ist. Der Proxy vermittelt die Nachrichten, die zwischen Klienten und Servern ausgetauscht werden. Ein typischer Anwendungsbereich von Proxies in verteilten Systemen sind die so genannten *Application Level Gateways*, die in Verbindung mit Firewalls eingesetzt werden.

Aus technischer Sicht ist ein Cache ein Proxy, der um einen Datenspeicher erweitert wurde. In diesem Speicher werden Objekte, die vom Server ausgeliefert werden, abgelegt und können bei erneutem Abruf unmittelbar geliefert werden. Die Eigenschaften von Caches im World Wide Web werden in einem eigenständigen Kapitel näher dargelegt.

2.1.3 Das Hypertext Transfer Protocol

Mit dem Hypertext Transfer Protokoll wurde ein eigenständiges Protokoll zur Übertragung von Daten im World Wide Web eingeführt. Die Entwicklung von HTTP vollzog sich in mehreren Schritten, wobei Verbesserungen im Protokollablauf, im Verbindungs-Management und in ergänzenden Header-Informationen vorgenommen wurden. In den folgenden Abschnitten werden die Eigenschaften von HTTP/1.0 und HTTP/1.1 betrachtet, die für das Caching und die Performance von HTTP von Bedeutung sind.

2.1.3.1 Grundlegende Eigenschaften von HTTP

Für die gesicherte Übertragung der Objekte setzt HTTP ein verbindungsorientiertes Transportprotokoll voraus. Auch wenn es nicht explizit gefordert wird, setzen Implementierungen von HTTP in der Regel auf einem TCP/IP-Protokollstack auf.

HTTP ist nicht statusbehaftet. Das bedeutet, dass jede Folge von HTTP-Request und HTTP-Response unabhängig von vorangegangenen Übertragungen ist. Auf keiner

Instanz in der WWW-Architektur werden Statusinformationen bezüglich des HTTP-Protokolls gespeichert. Um trotzdem der Forderung nach Statusinformationen nachzukommen, wurden die so genannten *Cookies* eingeführt [KrMo00]. Cookies sind nicht Bestandteil einer Spezifikation von HTTP, sondern lediglich eine proprietäre Erweiterung. Da der Einsatz von Cookies mittlerweile verbreitet ist, und Cookies den Nutzen von Caches beeinträchtigen können, werden sie in einem eigenständigen Kapitel 2.2.5 beschrieben.

2.1.3.2 Aufbau von Nachrichten in HTTP

In Abbildung 2.2 werden die Elemente, aus denen HTTP-Request und -Response bestehen, anhand eines Beispiels dargestellt.

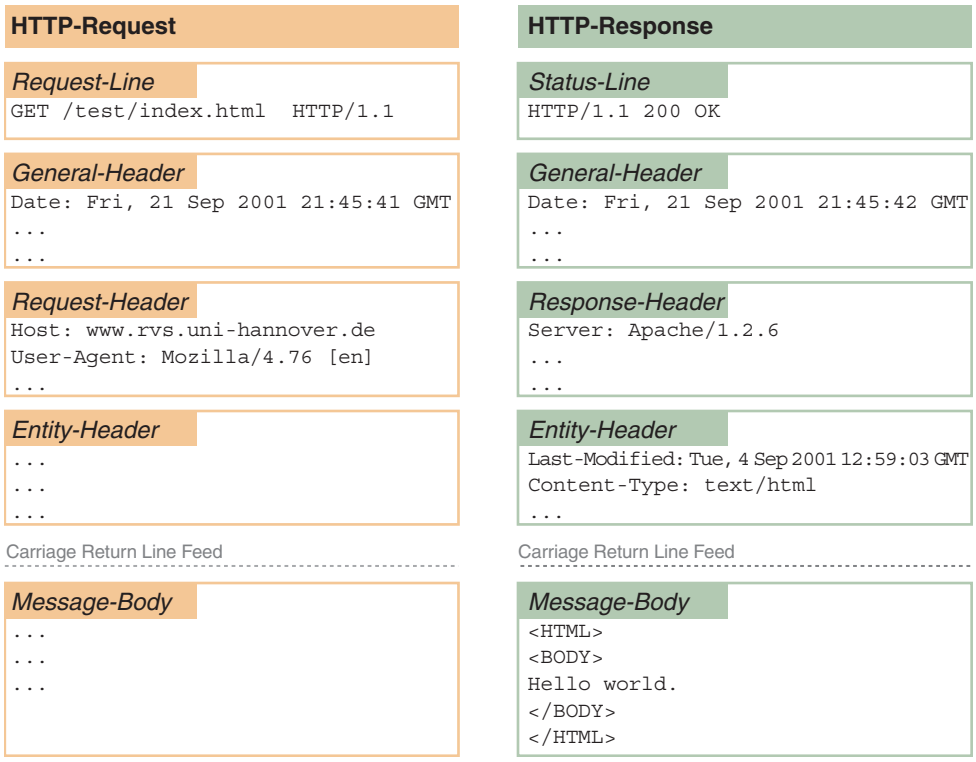


Abbildung 2.2 Aufbau von HTTP-Request und HTTP-Response

Aufbau HTTP-Request

Ein HTTP-Request wird von der *Request-Line* eingeleitet. Diese Zeile enthält die Methode des Aufrufs, den URL sowie die Versionsnummer des HTTP-Protokolls, das im WWW-Browser implementiert ist. Folgende Methoden für HTTP-Requests sind in HTTP/1.1 spezifiziert:

- GET: ruft das in dem URL angegebene Objekt vom WWW-Server ab.

- HEAD: ruft nur die Header-Informationen des in dem URL angegebenen Objekts ab.

- POST: sendet Daten an den WWW-Server. In der Regel wird diese Methode bei der Nutzung von Formularen verwendet. Die Daten werden im Body des HTTP-Requests übertragen. In dem URL ist das CGI-Programm auf dem WWW-Server angegeben, das die Daten verarbeitet.

- PUT: das in dem URL angegebene Objekt wird auf dem WWW-Server angelegt oder überschrieben. Die Daten des Objekts werden im Body des HTTP-Requests übertragen.

- DELETE: das in dem URL angegebene Objekt wird auf dem WWW-Server gelöscht.

- TRACE: ermöglicht den Test der Verbindung zum WWW-Server. Der WWW-Server kopiert alle Daten aus dem Header des HTTP-Requests in den Body der HTTP-Response.

- OPTIONS: mit dieser Methode können Informationen über den WWW-Server abgerufen werden. So läßt sich z. B. feststellen, ob die Methode TRACE von einem WWW-Server unterstützt wird.

- CONNECT: diese Methode ist zur Tunnelung von Ende-zu-Ende verschlüsselter Kommunikation über Caches notwendig [Res00]. So wird ein Cache durch die CONNECT-Methode angewiesen, eine Verbindung zu dem spezifizierten WWW-Server aufzubauen und die Daten transparent über die Verbindung durchzureichen.

Es ist zu beachten, das nur die Methoden GET und HEAD für Implementierungen nach HTTP/1.1 zwingend vorgeschrieben sind. Auf weitere Details zum Aufbau von HTTP-Requests wird hier nicht näher eingegangen. Die für das Caching wichtigen Header-Informationen werden in den folgenden Kapiteln betrachtet. Für eine detaillierte Beschreibung der Header sowie einen Vergleich zwischen HTTP/1.0 und HTTP/1.1 kann neben der Spezifikation in RFC 2616 [FGM+99] auch auf [KMK99] verwiesen werden.

Aufbau HTTP-Response

Die *Status-Line* enthält die Versionsnummer des HTTP-Protokolls, das der WWW-Server zur Kommunikation mit dem WWW-Klienten verwendet, einen Statuscode, der das Ergebnis der Bearbeitung des HTTP-Requests anzeigt, sowie eine Meldung im Klartext, die den Statuscode erläutern soll. Die Versionsnummer, die der WWW-Server zurücksendet, darf nicht höher sein als die, die der WWW-Klient in der Re-

quest-Line angegeben hat. Durch dieses Verfahren einigen sich WWW-Klient und WWW-Server auf die höchste gemeinsame Protokollversion. Der Klartext der Statusmeldungen wird durch RFC 2616 lediglich vorgeschlagen, Implementierungen dürfen diese Vorschläge ausdrücklich überschreiben.

Die Statuscodes werden in fünf Klassen unterteilt:

- 1xx: Informational. Reserviert für experimentelle Protokollimplementierungen, wird in der Praxis bisher nicht verwendet.

- 2xx: Success. Signalisiert die erfolgreiche Bearbeitung eines HTTP-Requests. Das geforderte Objekt ist im Body der HTTP-Response enthalten,

- 3xx: Redirection. Das geforderte Objekt kann nicht oder braucht nicht unter dem angegebenen URL ausgeliefert werden. Gegebenenfalls wird dem WWW-Klient ein alternativer URL mit der HTTP-Response übergeben.

- 4xx: Client Error. Der WWW-Server hat einen Fehler im HTTP-Request erkannt. Dieser Fehler tritt z. B. dann auf, wenn das im URL angegebene Objekt nicht auf dem WWW-Server existiert.

- 5xx: Server Error. Der WWW-Server kann wegen Überlastung oder interner Fehler (z. B. defekter Festplatte) den HTTP-Request nicht erfolgreich bearbeiten. Ein weiterer Grund für diese Meldung kann ein fehlgeschlagener Verbindungsaufbau zwischen WWW-Cache und WWW-Server sein.

Auch hier wird auf eine weiterführende Erläuterung der Header-Informationen verzichtet, indem auf die oben angegebenen Quellen verwiesen wird.

2.2 Caching im World Wide Web

Aufgrund der hohen Wartezeiten, die häufig bei der Übertragung von Daten im World Wide Web auftreten, liegt es nahe, das Prinzip des Caching anzuwenden. Zwei grundlegende Möglichkeiten für den Einsatz von Caching bieten sich an:

- Browser-Cache: bereits im WWW-Browser können Objekte im Hauptspeicher oder auf der Festplatte zwischengespeichert werden. Dieser Cache ist dann von Vorteil, wenn durch häufiges Vor- und Zurückblättern gleiche Seiten innerhalb kurzer Zeit betrachtet werden. Der Zugriff auf die Daten im Browser-Cache ist nur dem jeweiligen Nutzer gestattet.

- Cache-Server: auf eigenständigen Cache-Servern im lokalen Netz lassen sich die Inhalte, die von mehreren Nutzern abgerufen werden, zentral speichern. Die Vorteile eines Cache-Servers kommen dann zum Tragen, wenn unterschiedliche Nutzer dieselben Inhalte abrufen.

Im weiteren Verlauf der Studie werden ausschließlich Cache-Server betrachtet.

2.2.1 Nutzen von Caching

Der ursprüngliche Grund für den Einsatz von Cache-Servern war die Verringerung der Wartezeiten bei der Übertragung von Objekten im World Wide Web. Wichtiger für die Entwicklung der Caching-Technologie im World Wide Web war jedoch das Interesse der Netzbetreiber. Ziel der Netzbetreiber war es, durch Einsatz von Cache-Servern den hohen Anteil an WWW-Verkehr im Kernnetz und an den Übergangspunkten in andere Netze zu reduzieren.

Anhand Abbildung 2.3 wird die Berechnung der Maßzahlen zur Bewertung von Cache-Servern dargelegt. In der Darstellung werden die Datenflüsse an den Schnittstellen zwischen WWW-Servern und dem Cache-Server sowie zwischen dem Cache-Server und den WWW-Klienten betrachtet. R_{in} bezeichnet die Menge der HTTP-Responses, die der Cache-Server von WWW-Servern empfängt. Entsprechend bezeichnet R_{out} die Menge der HTTP-Responses, die der Cache-Server an die WWW-Klienten überträgt. Da die betrachteten Cache-Server nicht selbständig Inhalte aus dem WWW abrufen, gilt $R_{in} \subseteq R_{out}$. Weiterhin kann ein kausales System vorausgesetzt werden, in dem jeder HTTP-Response ein HTTP-Request vorausgeht. Unter der Annahme, dass HTTP-Responses nicht verloren gehen können, kann an beiden betrachteten Schnittstellen die Anzahl an HTTP-Responses den HTTP-Requests gleichgesetzt werden.

HTTP-Responses von WWW-Servern an den Cache-Server treten nur dann auf, wenn der Cache-Server eine Anfrage nicht aus seinem Speicher beantworten kann. Die Anzahl N_{Miss} dieser so genannten Cache-Misses entspricht somit den einge-

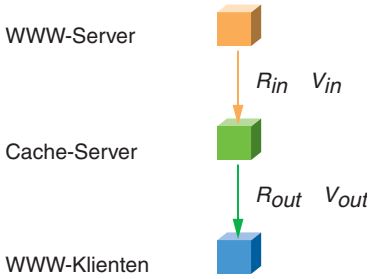


Abbildung 2.3 Einsparung Cache-Server

henden HTTP-Responses $|R_{in}|$. Weiterhin ergibt sich die Anzahl an Cache-Hits N_{Hit}, d. h. das Objekt befindet sich im Cache-Server, aus der Differenz von ausgehenden zu eingehenden HTTP-Responses $|R_{out}| - |R_{in}|$ am Cache-Server. Die Objekt-Trefferrate H_O (engl. Object Hit Ratio, Request Hit Ratio) ist definiert als das Verhältnis von Cache-Hits zu insgesamt bearbeiteten HTTP-Requests:

$$H_O = \frac{N_{Hit}}{N_{Hit} + N_{Miss}} = \frac{|R_{out}| - |R_{in}|}{|R_{out}| - |R_{in}| + |R_{in}|} = \frac{|R_{out}| - |R_{in}|}{|R_{out}|} = 1 - \frac{|R_{in}|}{|R_{out}|} \quad (2.1)$$

Die Volumen-Trefferrate H_V (engl. Volume Hit Ratio, Byte Hit Ratio) ergibt sich entsprechend, wobei die Volumen bereits als skalare Größen vorausgesetzt werden:

$$H_V = \frac{V_{out} - V_{in}}{V_{out}} = 1 - \frac{V_{in}}{V_{out}} \quad (2.2)$$

Aus der Volumen-Trefferrate ergibt sich sofort, um welchen Betrag der Cache-Server den eingehenden WWW-Verkehr verringert. Die Reduzierung der Wartezeiten kann dagegen nicht absolut angegeben werden, da für jeden Cache-Hit die Information fehlt, wie lange die Übertragung von dem originalen WWW-Server gedauert hätte. Jedoch lässt ein Vergleich der Datenraten, mit denen Cache-Hits und Cache-Misses übertragen werden, unter Berücksichtigung der Objekt-Trefferrate eine Abschätzung zu.

In Kapitel 2.2.4 wird eine differenziertere Betrachtung von Cache-Hits und Cache-Misses vorgenommen.

2.2.2 Aufbau von Cache-Servern

Bei der technischen Umsetzung müssen hardware-basierte und software-basierte Cache-Server unterschieden werden. In hardware-basierten Cache-Servern sind sämtliche Komponenten einschließlich Hardware und Betriebssystem auf den Einsatz als Cache-Server optimiert [TML99]. Diese Geräte können in der Regel nur als Cache-Server betrieben werden.

Ein software-basierter Cache-Server besteht aus einem herkömmlichen UNIX-basierten System, auf dem eine eigenständige Caching-Software installiert ist. Software-basierte Cache-Server weisen in der Regel geringere Leistungswerte als hardware-basierte auf. Demgegenüber sind die Anschaffungskosten von software-basierten Caches-Servern geringer, außerdem können sie für andere Zwecke, so z. B. als Datenbank- oder WWW-Server, verwendet werden.

2.2.3 Performance von Cache-Servern

In der Literatur finden sich zahlreiche Untersuchungen über die Performance von WWW-Servern (u. a. [AAY97] [BDR97] [BaCr98]) und Cache-Servern (u. a. [MEW00] [RoSo99] [FCD+99]). Es ist zu beachten, dass sich Untersuchungen über WWW-Server nur bedingt auf Cache-Server übertragen lassen. Ein WWW-Server stellt eine eindeutige Datenquelle im Netz dar. In dieser Datenquelle werden Objekte von den Festplatten gelesen, eventuell im Hauptspeicher für weitere schnelle Zugriffe abgelegt und an die WWW-Klienten übertragen. Ein Cache-Server vereint sowohl Datenquelle als auch Datensenke. Das Lastverhalten eines Cache-Servers wird wesentlich durch häufige Lese- und Schreiboperationen auf den Festplatten bestimmt.

Zwei Maßzahlen können zur Beurteilung der Performance eines Cache-Servers herangezogen werden:

- Durchsatz, gemessen in HTTP-Responses / Sekunde,
- Antwortzeiten für Cache-Hits und Cache-Misses.

In den Antwortzeiten für Cache-Misses sind die Verzögerungszeiten enthalten, die durch die Verbindungen zu den WWW-Servern entstehen. Sie werden dennoch angegeben, um die Antwortzeiten der Cache-Hits besser beurteilen zu können. Häufig werden als weitere Maße Objekt- und Volumen-Trefferrate genannt. Da die Trefferraten jedoch von dem Speichervolumen eines Cache-Servers sowie von der Anzahl und der Zusammensetzung der Nutzer abhängen, dienen sie lediglich zur Beurteilung des Nutzens und nicht der Performance.

Für Aussagen über potentielle *Bottlenecks* von Cache-Servern können Erfahrungsberichte aus dem praktischen Einsatz herangezogen werden [Wes01a]. Bei der Hardware-Ausstattung beeinflussen I/O-Komponenten die Performance maßgeblich. Auch wenn bereits einfache Festplattensysteme bis zu einer theoretischen Grenze von 100 MByte/s arbeiten, verringern die häufigen Lese- und Schreiboperationen für kleine Dateien diese Rate für einen Cache-Server auf maximal 10 MByte/s. Andere Komponenten, wie Prozessor, Hauptspeicher oder Netzinterface, haben dagegen keinen signifikanten Einfluss oder können ohne großen Aufwand den Anforderungen entsprechend ausgelegt werden.

Die Unterschiede zwischen hardware- und software-basierten Cache-Servern werden bei einer Betrachtung der Betriebssysteme deutlich. Software-basierte Cache-Server verwenden das auf die jeweilige Hardware angepasste UNIX-Betriebssystem. Da UNIX-Betriebssysteme für einen Mehrbenutzerbetrieb ausgelegt sind, liegen die zentralen Aufgaben in einer sicheren Zugriffssteuerung auf die Hardware-Ressourcen oder das Dateisystem sowie in einem zuverlässigen Prozess-Management [Bac86]. Diese Zielsetzung geht jedoch zu Lasten der Performance [Mog95b] [BDM98]. Als deutlicher Engpass hat sich dabei das *Standard UNIX File System* (UFS), bzw. betriebssystem-abhängige Varianten davon, erwiesen. So kann allein durch den Einsatz eines alternativen, für Caching optimierten Dateisystems der Durchsatz eines software-basierten Cache-Servers um den Faktor 2–4 erhöht werden [SGHS01].

Neben dem Dateisystem hat auch der TCP/IP-Stack des Betriebssystems entscheidenden Einfluss auf die Performance. Der TCP/IP-Stack ist Teil des Betriebssystems und wird folglich in Software implementiert. Wesentliche Eigenschaften des TCP/IP-Stacks, wie Timeouts, Fenstergrößen oder Sende- und Empfangspuffer, lassen sich durch so genannte Kernelparameter verändern. Dabei kann jedoch nur ein Kompromiss aus hoher Performance und zuverlässiger Übertragung erreicht werden, der sich in der Regel aus dem Einsatz in lokalen oder Weitverkehrsnetzen ergibt [Coc97].

Fast alle Hersteller von hardware-basierten Cache-Servern verwenden eigene Microkernel-Architekturen [BaOb00]. Ein Betriebssystem lässt sich wesentlich vereinfachen, indem auf kooperative Mechanismen, die die Zugriffe verschiedener Prozesse auf die Ressourcen des Systems regeln, verzichtet wird. Dem Cache-Prozess als zentraler Instanz können alleinige oder zumindest priorisierte Zugriffsrechte auf die Hardware-Ressourcen und das Dateisystem eingeräumt werden [TML99].

Zur Messung der Performance von Cache-Servern hat sich die Software *Polygraph* etabliert, die sowohl WWW-Klienten als auch WWW-Server nachbildet [Pol01]. Durch Angabe von u. a. Objekt-Trefferraten, Objektgrößen oder Zwischenankunftszeiten erzeugen die WWW-Klienten einen charakteristischen Strom von HTTP-Requests. Die WWW-Server senden entsprechende HTTP-Responses zurück, wobei sie durch einstellbare Verzögerungen oder Paketverluste die Datenraten beeinflussen. Die resultierende Workload bildet den typischen WWW-Verkehr in Weitverkehrsnetzen nach.

In regelmäßig stattfindenden *Cache-Offs* stellen Hersteller von Cache-Servern ihre Geräte für Benchmarks zur Verfügung, um unter identischen Voraussetzungen vergleichbare Resultate zu erzielen. Der letzte Cache-Off fand im Oktober 2000 statt [RoWe00]. Leistungsfähige Cache-Server erreichten dabei einen Durchsatz von über 2.000 HTTP-Responses/s. Bei den meisten Cache-Servern lagen die mittleren Anwortzeiten für einen Hit unter 200 ms, für einen Miss unter 3 s.

Allgemein lassen diese Benchmarks erkennen, dass hardware-basierte Cache-Server deutlich leistungsfähiger sind als software-basierte. Hardware-basierte Cache-Server werden jedoch nur in geringen Stückzahlen gefertigt und sind entsprechend teuer. Deshalb zeigt eine Normierung der Ergebnisse auf eine einheitlichen Preis, dass auch der Einsatz software-basierter Cache-Server unter entsprechenden Voraussetzungen gerechtfertigt ist.

2.2.4 Konsistenz zwischen Cache-Server und WWW-Server

Seit HTTP/1.0 stehen Verfahren zur Verfügung, mit denen Cache-Server die Aktualität der gespeicherten Objekte überprüfen können. So enthält eine HTTP-Response in dem Entity-Header `Last-Modified: "date"` das Datum der letzten Änderung eines Objekts. Unter Verwendung dieses Datums, das einer Versionsnummer des Objekts entspricht, kann der Cache-Server in einem *bedingten* HTTP-Request die Überprüfung durch den WWW-Server veranlassen. Hierfür wird im Header des bedingten HTTP-Requests die Zeile `If-Modified-Since: "date"` (IMS) eingefügt. Ist das Objekt im Cache-Server noch aktuell, sendet der WWW-Server eine HTTP-Response mit der Status-Line `HTTP/1.0 304 Not Modified` zurück. Der Cache-Server kann nach Erhalt dieser Antwort das Objekt aus dem Speicher an den WWW-Klienten übertragen. Ist das Objekt auf dem Cache-Server dagegen veraltet, antwortet der WWW-Server mit der Übertragung des aktuellen Objekts. Die Übertragung des Objekts wird, wie nach dem Erhalt eines herkömmlichen HTTP-Requests, mit der Status-Line `HTTP/1.0 200 OK` eingeleitet.

Die Einhaltung einer *starken* Konsistenz erfordert, dass bei jedem Cache-Hit zunächst die Aktualität des Objekts geprüft wird. Eine Überprüfung kann jedoch nur durch einen HTTP-Request nach dem oben dargestellten Verfahren an den originalen WWW-Server erfolgen, wodurch signifikante Verzögerungen entstehen. Deshalb steht eine starke Konsistenz im Widerspruch zu der Forderung, dass Cache-Server die Antwortzeiten im World Wide Web reduzieren sollen [LiCa97]. Um die Überprüfung für jeden Cache-Hit zu vermeiden, werden auf Cache-Servern heuristische Verfahren zur Abschätzung der Aktualität verwendet. Mit diesem Verfahren lässt sich jedoch nur eine *schwache* Konsistenz erreichen, bei der keine Garantie für die Aktualität der Inhalte gegeben werden kann [GwSe96].

Durch Einführung weiterer Header in HTTP/1.0 sollte die schwache Konsistenz auf Cache-Servern zuverlässiger gestaltet werden. So gibt der Entity-Header `Expires: "date"` in einer HTTP-Response ein festes Datum an, an dem ein Cache-Server spätestens die Aktualität des Objekts auf dem WWW-Server überprüfen muss. Der Nutzen dieses Headers ist jedoch begrenzt, da nur selten a priori bekannt ist, wann ein Objekt geändert werden wird. Außerdem werden Cache-Server das Objekt nicht abrufen, auch wenn es wider Erwarten doch vor dem mit `Expires:` angegebenen Datum auf dem WWW-Server geändert wird. Es zeigte sich, dass dieser Header hauptsächlich verwendet wird, um durch Angabe eines bereits vergange-

nen Datums den Cache-Server zu einer ständigen Aktualisierung des Objekts zu zwingen [WiMi99]. Für einen ähnlichen Zweck wurde in HTTP/1.0 ein weiterer Entity-Header `Pragma: no-cache` eingeführt, mit dem das Speichern von Objekten auf einem Cache-Server sogar völlig unterbunden werden kann.

Mit HTTP/1.1 wurde der Entity-Header `Age: "value"` eingeführt, durch den ein Cache-Server Klienten signalisiert, vor wie viel Sekunden die Aktualität des übertragenen Objekts auf dem originalen WWW-Server überprüft wurde. Als weitere Ergänzung steht in HTTP/1.1 der General-Header `Cache-Control: "value"` für HTTP-Requests und HTTP-Responses zur Verfügung. Durch die Angabe verschiedener Direktiven in diesem Header lässt sich die Speicherung von Objekten in Caches genauer steuern. Unter anderem stehen folgende Direktiven zur Verfügung:

- `private`: das Speichern des Objekts in der HTTP-Response ist nur im Browser-Cache, nicht aber auf Cache-Servern gestattet.

- `no-store`: das Speichern des Objekts in der HTTP-Response ist sowohl im Browser-Cache als auch auf Cache-Servern untersagt.

- `no-cache`: das Speichern des Objekts in der HTTP-Response ist sowohl im Browser-Cache als auch auf Cache-Servern gestattet. Vor der Auslieferung aus dem Cache muss jedoch stets die Aktualität des Objekts überprüft werden.

- `max-age=n`: gibt in einer HTTP-Response an, wie viele Sekunden `n` das Objekt von dem Cache-Server ohne vorhergehende Aktualisierung ausgeliefert werden darf. Die Wirkung entspricht dem Entity-Header `Expires:`. In einem HTTP-Request signalisiert `max-age=n` dem Cache-Server, dass der WWW-Klient nur solche Objekte akzeptiert, die vor weniger als `n` Sekunden aktualisiert wurden.

In Ergänzung zu Kapitel 2.2.1 lässt sich die Definition eines Cache-Hits ausweiten. So liegt ein *unbestätigter* Cache-Hit vor, wenn der Cache-Server anhand der Heuristik das Objekt als aktuell beurteilt. Wird das Objekt dagegen als veraltet beurteilt, wird es mit einem IMS-Request an den WWW-Server überprüft. Sendet der WWW-Server ein `Not-Modified` zurück, handelt es sich um einen *bestätigten* Cache-Hit und das Objekt kann aus dem Cache-Server übertragen werden. Wird ein aktuelles Objekt von dem WWW-Server übertragen, handelt es sich um einen bestätigten Cache-Miss. In diesem Fall wird das Objekt vom WWW-Server an den WWW-Klienten weitergeleitet, wobei das veraltete Objekt im Cache-Server mit dem aktuellen

überschrieben wird. Eine Übersicht der erweiterten Definitionen von Cache-Hit und Cache-Miss ist in Tabelle 2.1 zusammengefasst.

	Objekt ist im Cache	Objekt ist aktuell	HTTP-Request an WWW-Server	HTTP-Response von WWW-Server
unbestätigter Hit	ja	ja	–	–
bestätigter Hit	ja	nein	IMS	`304 Not Modified`
bestätigter Miss	ja	nein	IMS	`200 OK`
einfacher Miss	nein	–	GET	`200 OK`

Tabelle 2.1 Erweiterte Definition von Cache-Hit und Cache-Miss

Den höchsten Nutzen erbringen unbestätigte Cache-Hits, da die Objekte ohne vorhergehende Überprüfung auf dem WWW-Server sofort ausgeliefert werden können. Das Datenaufkommen, das bei bestätigten Cache-Hits zwischen Cache-Server und WWW-Server anfällt, ist vernachlässigbar gering. Die Zeit, die bei der Überprüfung des Objekts anfällt, verzögert jedoch die Auslieferung eines bestätigten Cache-Hits gegenüber einem unbestätigten Cache-Hit deutlich. Bei bestätigten Cache-Misses und einfachen Cache-Misses werden weder Datenvolumen noch Übertragungszeit eingespart.

Unbestätigte Cache-Hits verursachen keinen Datenverkehr zwischen Cache-Servern und WWW-Servern. Hierdurch ergibt sich der hohe Nutzen von unbestätigten Cache-Hits, jedoch auf Kosten einer schwachen Konsistenz. Bei bestätigten Cache-Hits, bestätigten Cache-Misses und einfachen Cache-Misses wird dagegen die Konsistenz im Rahmen der möglichen Genauigkeit gewahrt. Daraus folgt offensichtlich, dass eine Erhöhung der Trefferraten durch unbestätigte Cache-Hits zu Lasten einer gesicherten Konsistenz geht. Nach welchen Verfahren die Aktualität von Objekten abgeschätzt werden kann, um auch bei unbestätigten Cache-Hits eine möglichst sichere Konsistenz zu gewährleisten, wird im folgenden Kapitel untersucht.

2.2.4.1 Heuristik zur Beurteilung der Aktualität

Die Heuristik zur Beurteilung der Aktualität eines Objekts auf einem Cache-Server ist nicht Bestandteil einer HTTP-Spezifikation. Im Folgenden wird beispielhaft der Algorithmus erläutert, der in der Cache-Software *Squid* implementiert ist [Wes01b].

Zur Berechnung der Aktualität werden für jedes Objekt zunächst folgende Variablen definiert:

- `NOW`: aktuelles Datum auf dem Cache-Server bei Eintreffen eines HTTP-Requests.

- `EXPIRES`: das im Entity-Header `Expires:` der HTTP-Response angegebene Datum.

- `OBJ_DATE`: das im General-Header `Date:` der HTTP-Response angegebene Datum des WWW-Servers.

- `OBJ_LASTMOD`: das im Entity-Header `Last-Modified:` der HTTP-Response angegebene Datum der letzten Änderung des Objekts.

- `CLIENT_MAX_AGE`: die im General-Header `Cache-Control: max-age=n` angegebene maximal akzeptierbare Verweildauer des Objekts im Cache-Server.

Sämtliche Zeitinformationen werden mit einer Auflösung von 1 Sekunde angegeben. Die Verwendung der Header `Date:` und `Last-Modified:` wird von den HTTP-Spezifikationen nicht zwingend gefordert. Fehlen diese Informationen in einer HTTP-Response, werden unter Annahme des *worst case* folgende Ersetzungen in Squid vorgenommen:

- ein fehlender General-Header `Date:` wird durch das aktuelle Datum auf dem Cache-Server ersetzt, an dem das Ende des Headers der HTTP-Response gelesen wird.

- ein fehlender Entity-Header `Last-Modified:` wird durch den General-Header `Date:` aus derselben HTTP-Response ersetzt (`OBJ_LASTMOD=OBJ_DATE`).

Außerdem wird für den Algorithmus vorausgesetzt, dass die zeitlich kausale Reihenfolge $\texttt{OBJ\_LASTMOD} \leq \texttt{OBJ\_DATE} \leq \texttt{NOW}$ eingehalten wird. Bei einer Verletzung dieser Forderung werden die Ersetzungen `OBJ_DATE=NOW` sowie `OBJ_LASTMOD=OBJ_DATE` vorgenommen.

Aus den genannten Variablen werden in einem nächsten Schritt folgende Größen abgeleitet:

- `LM_AGE=OBJ_DATE-OBJ_LASTMOD`: das Alter des Objekts auf dem WWW-Server zum Zeitpunkt der letzten Aktualisierung.

- `OBJ_AGE=NOW-OBJ_DATE`: die bisherige Verweildauer des Objekts auf dem Cache-Server seit der letzten Aktualisierung.

- `LM_FACTOR=OBJ_AGE/LM_AGE`: Verhältnis von Alter des Objekts auf dem WWW-Server zur Verweildauer des Objekts im Cache-Server.

In Abbildung 2.4 sind die zuvor definierten Variablen dargestellt. Weiterhin sind drei mögliche Zeitpunkte eingefügt, wie sie im Entity-Header `Expires:` angegeben werden können. So liegt `EXPIRES_1` vor `OBJ_DATE` und damit auch stets vor

NOW. Dieses Objekt muss daher bei jedem Zugriff auf Aktualität geprüft werden. Der Zeitraum zwischen OBJ_DATE und EXPIRES_2 ist zum Zeitpunkt NOW überschritten, so dass auch dieses Objekt aktualisiert werden muss. Das Objekt, dessen Verweildauer erst zum Zeitpunkt EXPIRES_3 abläuft, kann hingegen unmittelbar ausgeliefert werden.

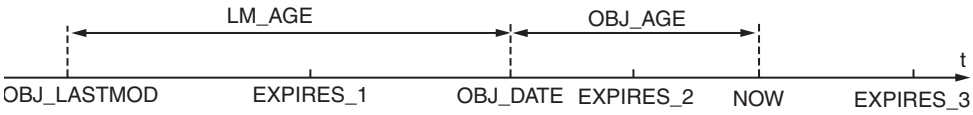


Abbildung 2.4 Variablen zur Berechnung der Aktualität

Zur Steuerung des Algorithmus lassen sich drei Parameter, die so genannten *Refresh Pattern*, konfigurieren:

- CONF_MIN: untere Grenze für die Verweildauer der Objekte im Cache-Server. Der Standardwert ist 0.

- CONF_MAX: obere Grenze für die Verweildauer der Objekte im Cache-Server. Der Standardwert beträgt 259.200 Sekunden bzw. 3 Tage.

- CONF_PERCENT: obere Grenze für das Verhältnis LM_FACTOR. Der Standardwert beträgt 0,2, d. h. die zulässige Verweildauer der Objekte im Cache-Server beträgt 1/5 des Alters zum Zeitpunkt der Auslieferung von den WWW-Servern.

Der vollständige Algorithmus ergibt sich aus Abbildung 2.5. Zu beachten ist die Reihenfolge der Bearbeitung. Falls vorhanden, haben die Angaben der Header Expires: oder Cache-Control: Priorität. Erst danach werden die Refresh Pattern verwendet, wobei zuerst die Überschreitung der maximalen Verweildauer, dann das Verhältnis LM_FACTOR und schließlich die minimale Verweildauer im Cache-Server geprüft wird.

Untersuchungen zeigen, dass der ausschlaggebende Parameter für die Häufigkeit der Aktualisierungen die maximale Verweildauer CONF_MAX ist. Eine geringe Verweildauer führt zu häufigen Aktualisierungen und damit zu einem Anstieg der bedingten Cache-Hits oder bestätigten Cache-Misses. Allerdings ist anzunehmen, dass eine geringe Verweildauer eine bessere Konsistenz der Objekte sichert. Wieweit jedoch Veränderungen der Refresh Pattern tatsächlich die Konsistenz der Objekte beeinflussen, kann im Rahmen dieser Studie nicht beantwortet werden. Hierfür müsste eine ständige Überprüfung der Objekte auf den WWW-Servern durchgeführt werden, was jedoch aufgrund des hohen Aufwandes kaum durchführbar ist. Entsprechende Untersuchungen sind auch in der Literatur nicht bekannt.

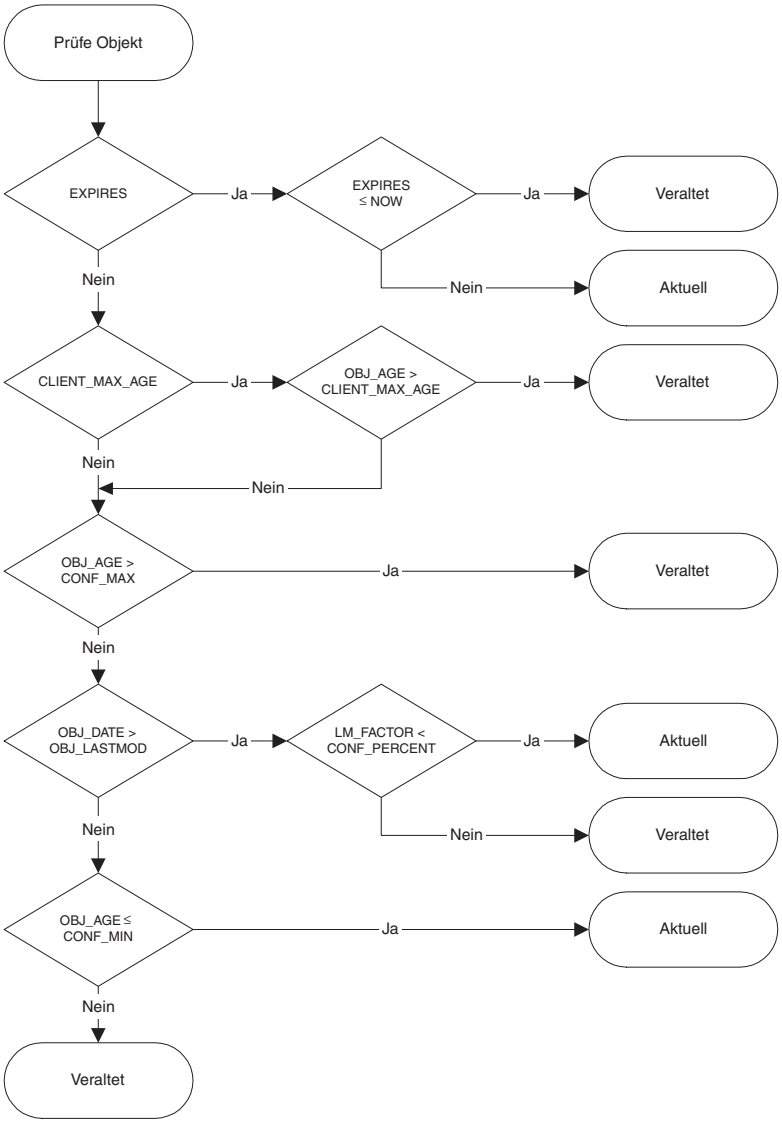


Abbildung 2.5 Algorithmus zur Überprüfung der Aktualität

2.2.5 Umgang mit Cookies

HTTP wurde als statusfreies Protokoll konzipiert, in dem der WWW-Server HTTP-Requests unabhängig voneinander bearbeitet. Eine Reihe von Anwendungen ist jedoch darauf angewiesen, dass Statusinformationen über mehrere Verbindungen weitergegeben und gegebenenfalls über längere Zeiträume gespeichert werden können. Mit Hilfe von CGI-Programmen lassen sich lediglich einfache Anwendungen, wie z. B. das Ausfüllen einer Reihe von Formularen, realisieren. Erst seit der Einführung der so genannten *Cookies* können vollständige Statusinformationen zwischen aufeinanderfolgenden HTTP-Requests übertragen und sogar über verschiedenen Sitzungen gespeichert werden [KrMo00].

Cookies werden ausschließlich auf WWW-Servern generiert. Für die Übertragung an einen WWW-Klienten enthält die HTTP-Response den Header `Set-Cookie: "value"`. Der Wert des Cookies besteht aus einem beliebigen ASCII-Text von maximal 4 KByte Länge. Der WWW-Klient legt den Inhalt des Cookies zusammen mit dem URL des Objekts für spätere Zugriffe auf der Festplatte ab. Bei dem nächsten Zugriff auf das Objekt erweitert der WWW-Klient unter Verwendung des gespeicherten Cookies den Header des HTTP-Requests um das Feld `Cookie: "value"`. Der WWW-Server kann das geforderte Objekt in Abhängigkeit des Cookies generieren.

Ein Cache-Server darf Objekte mit Cookies nicht ohne zusätzliche Maßnahmen speichern und an andere Klienten ausliefern. In der Cache-Software Squid wird vor der Speicherung der Eintrag `Set-Cookie: "value"` aus dem HTTP-Header eines Objekts gelöscht. HTTP-Requests nach dem Objekt werden mit einem Header `If-modified-since:` an den WWW-Server gesendet. Cookies in HTTP-Requests werden nicht verändert.

Die Tatsache, dass Cookies unbestätigte Cache-Hits verhindern, scheint zunächst den Nutzen von Cache-Servern zu mindern. Verschiedene Untersuchungen zeigen jedoch, dass der Einsatz von Cookies derzeit auf weniger als 5% der Objekte beschränkt ist [Wes01a] [WiMi99].

Kapitel 3

Einsatz lokaler Cache-Server

3.1 Konfiguration der WWW-Browser

In Abhängigkeit der eingesetzten Strategie zur Nutzung von Cache-Servern (s. Kapitel 3.2) müssen die WWW-Browser entsprechend konfiguriert werden. Eine korrekte Konfiguration der WWW-Browser ist eine notwendige Voraussetzung für die Nutzung der Cache-Server und damit auch für den Zugang zum WWW. Um den Aufwand zur Konfiguration der WWW-Browser aus Sicht der Nutzer möglichst gering zu halten, haben sich mittlerweile verschiedene Verfahren etabliert, mit denen Konfigurationsdateien zentral vorgehalten und von den WWW-Browsern selbstständig abgerufen werden können. Die unterschiedlichen Ansätze für die Konfiguration der WWW-Browser werden in den folgenden Kapiteln erläutert.

3.1.1 Selbständige Konfiguration durch die Nutzer

Die Nutzer tragen selbstständig die Parameter der Cache-Server in die Konfiguration ihrer WWW-Browser ein. Die Angaben bestehen in der Regel aus dem Dienst, der über den Cache-Server geleitet werden soll (z. B. HTTP oder FTP), dem symbolischen Namen des Cache-Servers sowie der Portnummer, auf der der Cache-Server die Requests entgegen nimmt. Das Vorgehen bei der Konfiguration unterscheidet sich zwischen verschiedenen WWW-Browsern erheblich. In der Regel werden die Parameter in den Voreinstellungen des WWW-Browsers oder über Umgebungsvariablen angegeben.

3.1.2 Nutzung einer zentralen Konfigurationsdatei

Mit dem *Proxy Auto-Config File Format* (PAC) wurde bereits in der Version 2.0 des WWW-Browsers von Netscape die Verwendung einer zentralen Konfigurationdatei vorgeschlagen. Die PAC-Datei wird auf einem WWW-Server im lokalen Netz vorgehalten, die Nutzer müssen lediglich den URL der Datei in ihrem WWW-Browser eintragen. Der Vorteil dieses Verfahrens liegt darin, dass Veränderungen an den Pa-

rametern der Cache-Server lediglich eine Anpassung der PAC-Datei erfordern und somit den Nutzern verborgen bleiben können.

Die Syntax einer PAC-Datei ist an Javascript angelehnt. In der Datei werden die oben aufgeführten Parameter der Cache-Server eingetragen. Zusätzlich lassen sich detaillierte Angaben vornehmen, so dass z. B. durch Anwenden von Regular Expression auf die URLs die HTTP-Requests zwischen mehreren Cache-Servern verteilt werden können.

3.1.3 Transparente Konfiguration

Mit dem *Web Proxy Autodiscovery Protocol* (WPAD) wurde von den Firmen Microsoft, Inktomi, Sun Microsystems und RealNetworks der Versuch unternommen, ein transparentes Verfahren für das Auffinden der PAC-Datei zu etablieren. Unter Verwendung bestehender Protokolle, wie dem *Dynamic Host Configuration Protocol* (DHCP), dem *Service Location Protocol* (SLP) oder spezieller Einträge im *Domain Name Service* (DNS), sollten WWW-Browser den URL der PAC-Datei ermitteln und die PAC-Datei automatisch abrufen können. Bisher wurde WPAD nur in dem Internet Explorer implementiert. Die zukünftige Entwicklung von WPAD scheint fraglich, da die Spezifikation seit einem Draft von Oktober 2000 nicht fortgeführt wurde. Der entsprechende Draft wurde inzwischen von der IETF zurückgezogen.

3.2 Strategien für die Nutzung

Ein Cache-Server erbringt erst dann einen Nutzen, wenn mehrere Nutzer dieselben Inhalte aus dem World Wide Web abrufen. Die Wahrscheinlichkeit dafür, dass Objekte mehrfach abgerufen werden, steigt mit zunehmender Anzahl Nutzer. Daher ist eine zentrale Fragestellung bei dem Einsatz eines Cache-Servers, durch welche Maßnahmen möglichst viele Nutzer über den Cache-Server geleitet werden können. Hierbei lassen sich drei Ansätze unterscheiden, die in den folgenden Kapiteln betrachtet werden.

3.2.1 Freiwillige Nutzung

Die Entscheidung zur Nutzung des Cache-Servers wird den Nutzern überlassen. Wie in Abbildung 3.1 dargestellt, können die WWW-Browser sowohl direkt, als auch über den Cache-Server auf die Inhalte im WWW zugreifen.

Ein hohe freiwillige Nutzung wird sich nur dann einstellen, wenn der Cache-Server durch Verringerung der Wartezeiten einen erkennbaren Vorteil für die Nutzer erbringt. Die Schwierigkeit bei der freiwilligen Nutzung liegt darin, dass unzufriedene Nutzer den direkten Weg zu den WWW-Servern vorziehen und kaum wieder auf den Cache-Server zurückgreifen werden. Ein langfristiger Nutzen des Cache-Servers bei freiwilliger Nutzung ist fraglich.

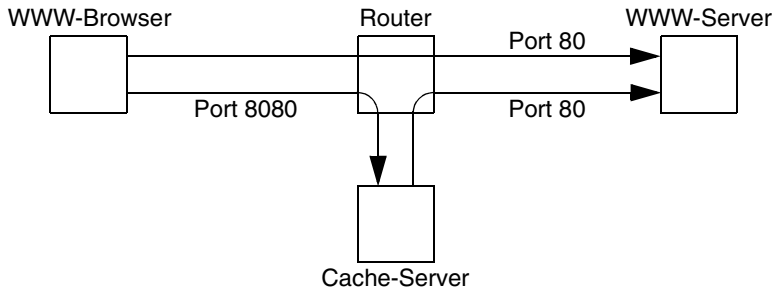


Abbildung 3.1 Freiwillige Nutzung eines Cache-Servers

3.2.2 Erzwungene Nutzung

Auf einem Router kann ausgehender WWW-Verkehr anhand der Ziel-Portnummer 80 für HTTP identifiziert werden. Unter Verwendung entsprechender Kontrollmechanismen auf dem Router ist es möglich, lediglich dem Cache-Server direkten Zugriff auf WWW-Server zu gestatten. Ausgehender WWW-Verkehr von anderen Systemen im Netz wird durch den Router blockiert (Abbildung 3.2). Durch diese Maßnahme werden die Nutzer gezwungen, ihre WWW-Browser zur Nutzung des Cache-Servers nach den in Kapitel 3.1 genannten Verfahren zu konfigurieren.

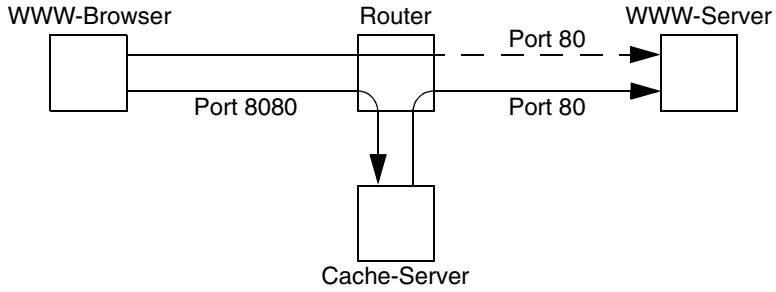


Abbildung 3.2 Erzwungene Nutzung eines Cache-Servers

Der Verfügbarkeit und Leistungsfähigkeit des Cache-Servers kommt bei der erzwungenen Nutzung eine höhere Bedeutung zu als bei der freiwilligen Nutzung, da ein Ausfall oder Überlastung des Cache-Servers den Nutzern den Zugang zum WWW unmöglich macht.

Voraussetzung für die Einführung der erzwungenen Nutzung eines Cache-Servers ist, dass alle Nutzer über die notwendige Konfiguration ihrer WWW-Browser informiert werden.

3.2.3 Transparente Umleitung

Ein Router oder ein so genannter Layer-4-Switch identifiziert im lokalen Netz ausgehenden WWW-Verkehr und leitet ihn automatisch an einen Cache-Server weiter. Dieses Verfahren ist aus Sicht der Nutzer transparent, da sie keine Konfiguration an ihrem WWW-Browser vornehmen müssen. Daraus folgt, dass wie bei der erzwungenen Nutzung keine Möglichkeit besteht, den Cache-Server zugunsten eines direkten Zugriffs auf das WWW zu umgehen. Die korrekte Bezeichnung für transparentes Caching lautet nach der Taxonomie gemäß RFC 3040 *Interception Caching*, ein transparenter Cache wird als *Interception Proxy* bezeichnet [CMT01].

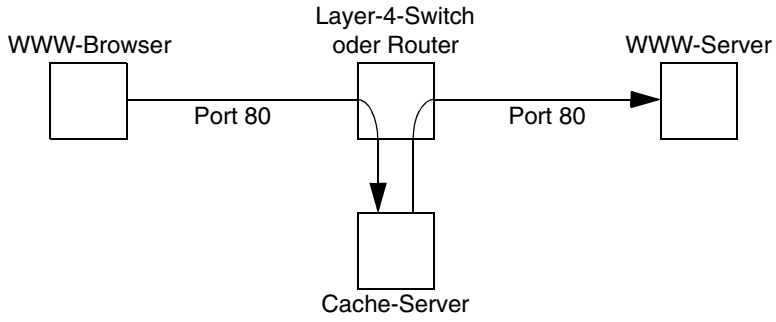


Abbildung 3.3 Transparente Umleitung von WWW-Verkehr

Umleitung durch Router mit WCCP

Für die transparente Umleitung des WWW-Verkehrs von einem Router an einen Cache-Server steht das *Web Cache Coordination Protocol* (WCCP) von Cisco [CFTW01] zur Verfügung. WCCP ist ausschließlich in Routern von Cisco verfügbar, für die Implementierung in Cache-Servern werden Lizenzgebühren von Cisco gefordert. Im Folgenden wird die Version 2.0 von WCCP beschrieben, die in IOS ab der Version 12.0 enthalten ist.

Mit WCCP wird die Verfügbarkeit von Cache-Servern kontrolliert und die eigentliche Umleitung des WWW-Verkehrs auf mehrere Cache-Server gesteuert. Über einen kontinuierlichen Austausch der WCCP-Meldungen `HERE_I_AM` und `I_SEE_YOU` prüfen Router und Cache-Server gegenseitige Verfügbarkeit. Durch die Meldung `DECLINE_PACKET` kann ein Cache-Server unter anderem temporäre Überlastung signalisieren, wonach der Router die Weiterleitung neuer HTTP-Requests auf diesen Cache-Server unterlässt. Der in WCCP beschriebene Algorithmus erkennt Ausfälle von Cache-Servern innerhalb von 30 Sekunden.

Stehen mehrere Cache-Server zur Verfügung, kann über WCCP eine Lastverteilung vorgenommen werden. Hierfür tauschen Cache-Server und Router die Meldungen `ASSIGN_BUCKETS` aus, mit denen eine Partitionierung der einzelnen Cache-Server

vorgenommen wird. Der Router verwendet als Hashwerte für die Partitionierung die IP-Adressen der WWW-Server.

Die Umleitung der Pakete vom Router zum Cache-Server erfolgt durch IP-Tunneling nach der *Generic Routing Encapsulation* (GRE) [RFC 2784]. Mit dieser Methode fasst der Router jedes vom einem WWW-Browser empfangene IP-Paket in ein neues IP-Paket ein und leitet es an den Cache-Server weiter. Auf dem Cache-Server werden der GRE-Header entfernt und die ursprünglichen IP-Pakete zusammengefügt. Für den Verkehr vom Cache-Server an den Router sind keine zusätzlichen Maßnahmen erforderlich. Die Umleitung über GRE bietet den Vorteil, dass Router und Cache-Server in verschiedenen IP-Subnetzen installiert sein können. Als alternatives Verfahren zu GRE bietet WCCP auch ein so genanntes *Layer 2 Rewrite* an, in dem lediglich MAC-Adressen umgeschrieben werden. Bei diesem Verfahren entfällt der Overhead durch die GRE-Header, jedoch müssen Router und Cache-Server in demselben Netzsegment betrieben werden.

Umleitung durch Layer-4-Switch

Steht kein Router von Cisco für den Einsatz von WCCP zur Verfügung, kann die transparente Umleitung des WWW-Verkehrs mit einem Layer-4-Switch erfolgen. Ein Layer-4-Switch stellt eine eigenständige Komponente im Netz dar, die durch Auswertung der Portnummern in TCP- oder UDP-Paketen die Daten an definierte Systeme umlenken kann. Für die Umleitung stehen mit IP-Tunneling und Layer 2 Rewrite prinzipiell dieselben Verfahren wie bei WCCP zur Verfügung.

Durch den Einsatz von Layer-4-Switches werden keine eigenständigen Protokolle eingeführt, mit denen die Verfügbarkeit der Cache-Server kontrolliert werden kann. Alternativ werden Eigenschaften herkömmlicher Protokolle ausgewertet. Hierbei bieten sich die Antwortzeiten von ARP-Requests, ICMP-Echo, TCP-Connect oder HTTP-Responses an.

Die Methoden zur Lastverteilung unterscheiden sich bei verschiedenen Implementierungen von Layer-4-Switches. Neben einem einfachem Round-Robin kann der Layer-4-Switch die Anzahl der aktiven TCP-Verbindungen (least connections), die Antwortzeiten oder die Datenrate (z. B. durch Auszählen von Paketen pro Zeiteinheit) der angeschlossenen Cache-Server als Grundlage für die Verteilung der HTTP-Requests verwenden.

Um gewährleisten zu können, dass HTTP-Requests mit denselben URLs stets an einen eindeutigen Cache-Server geleitet werden, sind jedoch andere Methoden zur Lastverteilung erforderlich. Eine einfache Methode besteht in einem Hashing auf Basis der Adressen von WWW-Servern (vgl. WCCP). Hierbei besteht jedoch die Gefahr, dass die Requests nicht gleichmäßig zwischen den Cache-Servern verteilt werden. Eine gleichmäßige Verteilung bietet ein Hashing auf der Basis vollständi-

ger URLs. Der Nachteil dieses Verfahrens ist jedoch, dass der Switch nicht nur das TCP-Paket, sondern den vollständige HTTP-Request auswerten muss. Entsprechende Geräte werden häufig als Layer-7-Switch bezeichnet.

Da Layer-4-Switches lediglich die Portnummern aus den TCP- bzw. UDP-Paketen auswerten, können sie in der Regel einfacher und dadurch performanter ausgelegt werden als komplexe Router mit WCCP. Demgegenüber stellt ein Layer-4-Switch eine zusätzliche Komponente im Netz dar, während der Router in jedem Fall vorhanden ist. Ob Layer-4-Switches oder WCCP für eine transparente Umleitung verwendet werden, hängt folglich auch von den bereits vorhandenen Komponenten im Netz ab.

3.3 Dimensionierung von Cache-Servern

Die notwendige Dimensionierung von Cache-Servern lässt sich nur unter einer Reihe von Annahmen bestimmen. Im Folgenden werden einige Hilfen zur Abschätzung einer geeigneten Dimensionierung vorgenommen, wobei auf mehrere Erfahrungswerte aus der Praxis zurückgegriffen wird.

Für die Dimensionierung eines Cache-Servers müssen folgende Größen bestimmt werden:

- Kapazität des Netzinterface
- Volumen der Festplatten
- Hauptspeicher
- Durchsatz in Requests/Sekunde

Die ersten drei Größen kennzeichnen die Hardware-Ausstattung und bilden somit feste und prüfbare Größen, wogegen der Durchsatz durch die Implementierung des Cache-Servers bestimmt wird und häufig schwer verifizierbar ist. In den folgenden Abschnitten werden Rezepte für Berechnungen eines Cache-Servers, zum Teil am Beispiel der Universität Hannover, durchgeführt. Die Darstellungen sind so angelegt, dass jede Einrichtung die Berechnungen mit eigenen Werten nachvollziehen kann.

Die Grundlage der Berechnungen bilden verschieden Werte, die sich aus messbaren Größen sowie Erfahrungswerten zusammensetzen:

- eingehendes Datenvolumen an Wochentagen: 250 GByte (z. B. aus Customer Network Management).

- Anteil WWW-Verkehr: 60% (Erfahrungswert).
- Anteil nicht-cachebares Volumen: 20% (Erfahrungswert, hierzu zählen z. B. URLs, die die Zeichenkette `cgi` oder `?` enthalten).
- Request- und Volumen Trefferrate: 30% (Erfahrungswert, s. Kapitel 3.4).
- mittlere Objektgröße im WWW: 10 KByte (aus div. Untersuchungen)
- Peak-Faktor: 5 (Erfahrungswert, evtl. deutlich höher)

Netzinterface

Die Auswahl des geeigneten Netzinterfaces orientiert sich an der vorhandenen Infrastruktur sowie an der geplanten Strategie für den Einsatz. Übliche Größen sind Fast Ethernet oder Gigabit Ethernet, wobei je nach Verkehrsaufkommen oder notwendiger Redundanz zwei Interfaces zu empfehlen sind.

Plattenkapazität

Als empfohlene Plattenkapazität wird in [Wes01a] das Volumen angenommen, dass über einen Zeitraum von drei Tagen gespeichert werden kann. Die Größenordnung der Plattenkapazität kann nach folgenden Berechnungen abgeschätzt werden:

Datenvolumen Wochentag		*250 GByte/Tag*
Anteil WWW-Verkehr	*x 60%*	*150 GByte/Tag*
Cache-Miss	*x 70%*	*105 GByte/Tag*
Anteil „cacheable“	*x 80%*	*84 GByte/Tag*
Zeitraum 3 Tage	*x 3 Tage*	***252 GByte***

Hauptspeicher

Bei hardware-basierten Cache-Servern ist der zur Verfügung stehende Hauptspeicher bereits an die verwendete Plattenkapazität angepasst. Für die Cache-Software Squid lautet eine Empfehlung, dass ca. 1 MByte Hauptspeicher pro 32 MByte Plattenspeicher verwendet werden sollen [Wes01a]. In dem oben genannten Beispiel ergibt sich hierfür ein Hauptspeicher von ca. 8 GByte!

Mittlere Ankunftsrate, Durchsatz

Der Durchsatz, der von einem Cache-Server bedient werden muss, kann mit folgender Abschätzung angegeben werden:

Datenvolumen Wochentag		*250 GByte/Tag*
Anteil WWW-Verkehr	*x 60%*	*150 GByte/Tag*
in KByte pro Sekunde	*x 12,1363*	*1.820 KByte/Sekunde*
mittlere Objektgröße	*/ 10 KByte/Objekt*	*182 Objekte/Sekunde*
entspricht		*182 Requests/Sekunde*
Peak-Faktor	*x 5*	***910 Requests/Sekunde***

Der ermittelte Wert stellt die Ankunftsrate von HTTP-Requests dar. Der Durchsatz bzw. die Höhe der gleichzeitig zu bewältigenden HTTP-Verbindungen setzt sich aus der Ankunftsrate neuer HTTP-Requests sowie der Bedienzeit zusammen. Bei typischen mittleren Bedienzeiten von über 3 bis 5 Sekunden ergibt sich ein notwendiger Durchsatz von deutlich über 1000 HTTP-Requests. Da für jeden Cache-Miss je eine Verbindung zwischen WWW-Klient und Cache-Server sowie zwischen Cache-Server und WWW-Server aufgebaut wird, erhöht sich die Anzahl der vom Cache-Server tatsächlich verwalteten Verbindungen zusätzlich um den Anteil der Cache-Misses.

Durchsatz der Festplatten

Die Belastung der Festplatten bzw. des Dateisystems setzt sich aus häufigen Schreib- und Leseoperationen zusammen.

Schreiben auf Festplatte		
Datenvolumen Wochentag		*250 GByte/Tag*
Anteil WWW-Verkehr	*x 60%*	*150 GByte/Tag*
Cache-Miss	*x 70%*	*105 GByte/Tag*
Anteil „cacheable“	*x 80%*	*84 GByte/Tag*
in KByte pro Sekunde	*x 12,1363*	***1.019 KByte/Sekunde***
mittlere Objektgröße	*/ 10 KByte/Objekt*	***102 Objekte/Sekunde***

Lesen von Festplatte		
Datenvolumen Wochentag		*250 GByte/Tag*
Anteil WWW-Verkehr	*x 60%*	*150 GByte/Tag*
Cache-Hit	*x 30%*	*45 GByte/Tag*
in KByte pro Sekunde	*x 12,1363*	***546 KByte/Sekunde***
mittlere Objektgröße	*/ 10 KByte/Objekt*	***55 Objekte/Sekunde***
Peak-Leistung		
Peak-Faktor	*x 5*	
Schreiben		***5.095 KByte/Sekunde***
	/ 10 KByte/Objekt	***510 Objekte/Sekunde***
Lesen		***2.730 KByte/Sekunde***
	/ 10 KByte/Objekt	***273 Objekte/Sekunde***

Der berechnete Wert der resultierenden Transferleistung erscheint zunächst gering. Bei gefüllten Festplatten setzen sich die Operationen jedoch aus häufigen, über sämtliche Sektoren verteilten Lese- und Schreiboperationen zusammen.

Bei Dateisystemen, die auf dem UNIX File System (UFS) basieren, wird der Durchsatz wesentlich durch die synchron zu den Lese- und Schreiboperationen durchgeführte Aktualisierung der Verzeichniseinträge gesenkt. Die Aktualisierungen sind notwendig, um die Konsistenz des Dateisystems auch bei gleichzeitigem Zugriff durch mehrere Nutzer bzw. Prozesse aufrecht zu halten. Zu den typischen Hinweisen für eine Verbesserung der Performance gehören daher Mount-Optionen oder Parameter bei der Erstellung des Dateisystems, die auf eine strenge Konsistenz zugunsten schnellerer Zugriffszeiten verzichten.

3.4 Kalkulation der notwendigen Einsparungen

Seit Einführung der volumen-basierten Tarifierung im Gigabit-Wissenschaftsnetz lassen sich die notwendigen Einsparungen durch einen Cache-Server kalkulieren, um in eine günstigere Kategorie eingestuft werden zu können. Die Grundlage der Tarifierung bildet das eingehende Datenvolumen der Einrichtung, das in den monat-

lichen Rechnungsstellungen des DFN-Vereins mitgeteilt wird. Aus der monatlichen Summe und der aktuellen Liste der Nutzungsentgelte kann jede Einrichtung ermitteln, welche Einsparungen für die Einordnung in eine niedrigere Tarifierung notwendig sind.

Beispiel für Übergang zwischen G-WiN-Kategorie 11 und 12

Die folgenden Berechnungen werden am Beispiel einer Einrichtung vorgenommen, die in die Kategorie 11 mit einem maximalen eingehenden Datenvolumen von 6 TByte/Monat eingeordnet ist (Stand November 2001). Als Ergebnis der Berechnungen wird die notwendige Volumen-Trefferrate $H_V = 1 - V_{in}/V_{out}$ (Gleichung 2.2) ermittelt, die ein Cache-Server erzielen muss, um das eingehende Datenvolumen V_{WiN} stets auf einem Wert unterhalb von 6 TByte/Monat zu halten. Betrachtet wird dabei der gesamte Bereich der Kategorie 12, der bei einem Volumen von 12 TByte/Monat endet.

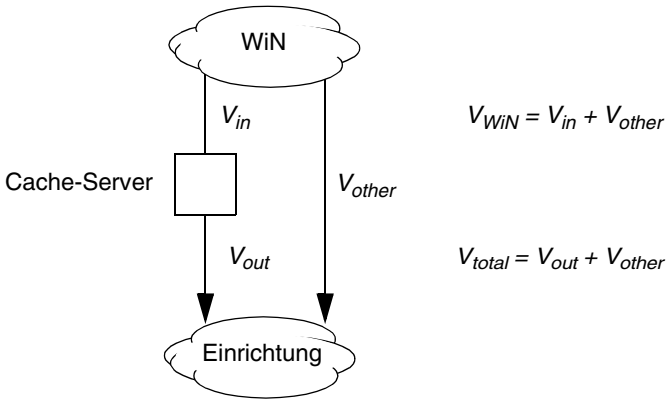


Abbildung 3.4 Berechnung der notwendigen Einsparungen

Bei den Berechnungen ist zu beachten, dass das tatsächlich genutzte Datenvolumen nur zu einem Teil aus WWW-Verkehr besteht. Um die folgenden Berechnungen für verschiedene Anteile durchführen zu können, wird das Verhältnis k_{WWW} eingeführt:

$$\begin{aligned} k_{WWW} &= \frac{V_{out}}{V_{total}} \quad \Leftrightarrow \quad V_{out} = k_{WWW} \cdot V_{total} \\ \Rightarrow V_{other} &= (1 - k_{WWW}) \cdot V_{total} = \frac{1 - k_{WWW}}{k_{WWW}} \cdot V_{out} \end{aligned} \tag{3.1}$$

Der genauen Angabe von k_{WWW} kommt im weiteren Verlauf der Betrachtungen eine zentrale Bedeutung zu. Der aktuelle Wert kann durch Auswertung von Port-Statistiken auf dem Access-Router einfach ermittelt werden. Die Schwierigkeit liegt jedoch in Vorhersagen über die zukünftige Entwicklung von k_{WWW}. So ist ein

Absinken von k_{WWW}, verursacht z. B. durch das zunehmende Aufkommen von Peer-to-Peer-Diensten, nicht auszuschließen.

Die Berechnung der notwendigen Volumen-Trefferrate H_V ergibt sich wie folgt:

$$\begin{aligned} V_{WiN} &= V_{in} + V_{other} \\ \Leftrightarrow V_{WiN} &= (1 - H_V) \cdot V_{out} + V_{other} \\ \Leftrightarrow H_V &= 1 - \frac{V_{WiN} - V_{other}}{V_{out}} \\ \Leftrightarrow H_V &= 1 - \frac{V_{WiN} - (1 - k_{WWW}) \cdot V_{total}}{k_{WWW} \cdot V_{total}} \\ \Leftrightarrow H_V &= \frac{1}{k_{WWW}} \cdot \left(1 - \frac{V_{WiN}}{V_{total}}\right) \end{aligned} \tag{3.2}$$

Diese Gleichung für die Volumen-Trefferrate H_V kann auch anschaulich aus Gleichung 2.2 abgeleitet werden. Das Diagramm in Abbildung 3.5 stellt die Ergebnisse aus Gleichung 3.2 grafisch dar. Das tatsächliche Datenvolumen V_{total} wird auf der Ordinate für einen Bereich von 6–12 TByte/Monat aufgetragen, als eingehendes Datenvolumen V_{WiN} wird die obere Grenze von Kategorie 11 mit 6 TByte/Monat angenommen. Neben dem in den vorangegangenen Berechnungen verwendeten Anteil von 60% WWW-Verkehr sind alternativ die Kurven für Anteile von 50% und 70% in das Diagramm eingetragen.

Wie der Abbildung zu entnehmen ist, kann in dem gewählten Beispiel mit einer typischen Volumen-Trefferrate von 30% ein Datenvolumen von ca. 1,4 TByte/Monat kompensiert werden, um in die Kategorie 11 unterhalb von 6 TByte/Monat zu verbleiben.

Allgemeine Darstellung

Gleichung 3.2 lässt sich auch für eine allgemeine Darstellung der notwendigen Volumen-Trefferrate eines Cache-Servers verwenden. Im Gegensatz zu dem oben dargestellten Beispiel wird in dem Diagramm aus Abbildung 3.6 auf der Ordinate das Verhältnis von eingehendem zu tatsächlich genutztem Verkehr V_{WiN}/V_{total} für einen Bereich von 0.5 bis 1 aufgetragen.

Aus diesem Diagramm lässt sich unabhängig von der aktuellen Kategorie die notwendige Trefferrate ermitteln, indem das Verhältnis von der oberen Grenze der nächst günstigeren Volumenklasse V_{WiN} zu dem aktuell ermittelten eingehenden Volumen V_{total} gebildet wird.

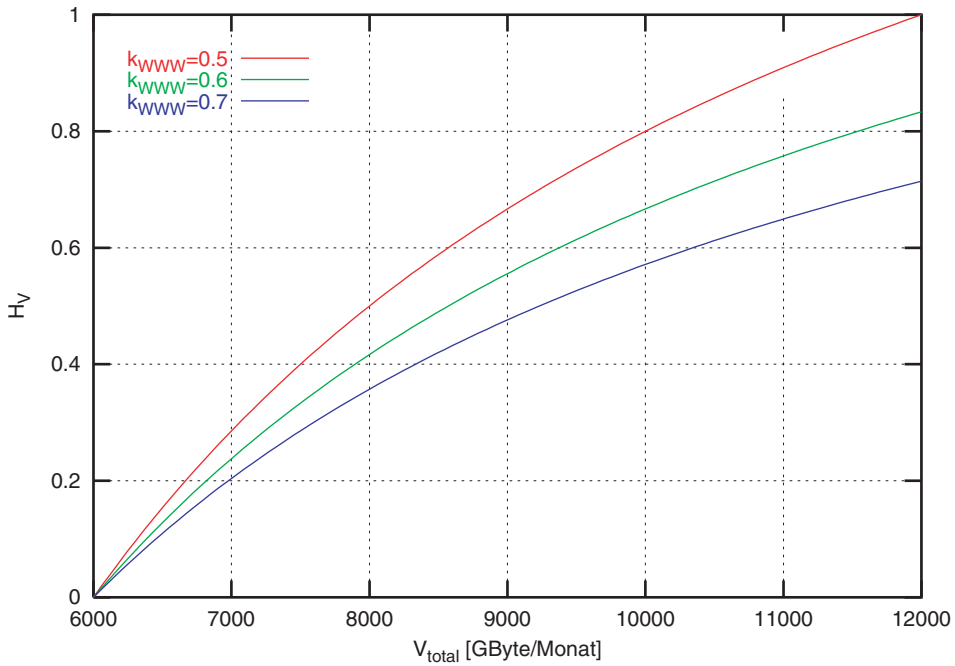


Abbildung 3.5 Notwendige Volumen-Trefferrate am Beispiel der Kategorie 11

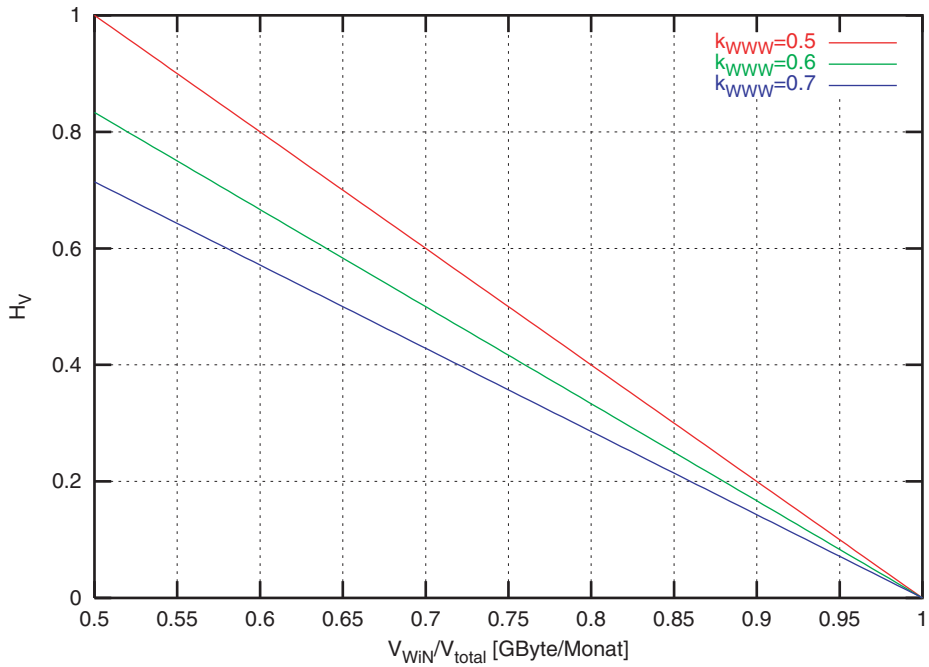


Abbildung 3.6 Allgemeine Berechnung der notwendigen Volumen-Trefferrate

3.5 Verfügbare Produkte

Auf eine Zusammenstellung verfügbarer Cache-Server wird im Rahmen dieser Studie verzeichtet, da die Spezifikationen in der Regel nur kurzzeitige Gültigkeit besitzen. Eine von Herstellern authorisierte Übersicht mit vergleichbaren, in der Praxis ermittelten Leistungsmerkmalen ist mit der bevorstehende Veröffentlichung der Ergebnisse des neuesten Cache-Offs [Pol01] Ende November 2001 zu erwarten.

Kapitel 4

Zusammenfassung

Mit dem vorliegenden Bericht wird eine Entscheidungshilfe für über Einsatz lokaler Cache-Server vorgelegt. Der primäre Nutzen lokaler Cache-Server wird dabei durch die Reduzierung von eingehendem WWW-Verkehr definiert. Andere Gründe für den Einsatz von Cache-Servern, wie z. B. die Reduzierung von Zugriffszeiten oder sicherheitsrelevante Aspekte, wurden im Rahmen dieser Studie nicht näher betrachtet.

Ein zentrales Ergebnis liegt in einer Kalkulation, mit der sich der notwendige Nutzen lokaler Cache-Server berechnen lässt. Die Kalkulation orientiert sich an der volumen-basierten Tarifierung, die für alle an das G-WiN angeschlossenen Einrichtungen gültig ist. Anhand bekannter Verkehrsdaten kann somit für jede Einrichtung die erforderliche Volumen-Trefferrate eines lokalen Cache-Servers ermittelt werden. Die Volumen-Trefferrate liefert das maßgebende Kriterium für die Entscheidung über den Einsatz eines lokalen Cache-Servers.

In einem weiteren Abschnitt werden Strategien vorgestellt, um den gesamten WWW-Verkehr über lokale Cache-Server zu führen. Die Bewertung der Ansätze berücksichtigt den zur Umsetzung notwendigen Aufwand sowohl aus Sicht der Nutzer als auch aus Sicht der Administratoren. Als besonders geeignete Lösung wird das so genannte transparente Caching angesehen. Mit WCCP oder Layer-4-Switches stehen hierfür zwei Alternativen zur Auswahl, deren jeweiligen Vorteile sich jedoch nur anhand lokaler Gegebenheiten beurteilen lassen.

Neben der Entscheidungshilfe über den Einsatz sowie der Empfehlung für ein Betriebskonzept bildet die Dimensionierung lokaler Cache-Server den dritten Schwerpunkt der Studie. Analog zu der Kalkulation der notwendigen Einsparungen werden in einfach durchzuführenden Berechnungen grundlegende Ausstattungs- und Leistungsmerkmale von Cache-Servern definiert. Aus diesen Parametern können Anforderungen an die zu beschaffenden Cache-Server abgeleitet werden. Durch Verweis auf aktuelle Benchmarks wird auf die Beschreibung konkreter Produkte im Rahmen dieser Arbeit verzichtet.

Abschließend muss darauf hingewiesen werden, dass der Anteil des WWW-Verkehrs gegenüber dem gesamten Verkehr maßgeblich in die Berechnungen eingeht. Da davon auszugehen ist, dass andere Dienste an Bedeutung gewinnen werden, sollte eine Abschätzung der zukünfigen Entwicklung des WWW-Verkehrs in die dargelegten Betrachtungen einbezogen werden.

Literaturverzeichnis

[AAY97] J. M. Almeida, V. Almeida, D. J. Yates. *Measuring the Behavior of a World-Wide Web Server.* Proc. 7th IFIP Conference on Performance Networking, S. 57–72, White Plains, April 1997.

[Bac86] M. J. Bach. *Design of the Unix Operating System.* Prentice Hall, 1986.

[BaCr98] P. Barford, M. E. Crovella. *Generating Representative Web Workloads for Network and Server Performance Evaluation.* Proc. of Performance 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of computer Systems, S. 151–160, Madison, Mai 1998.

[BaOb00] G. Barish, K. Obraczka. *World Wide Web Caching: Trends and Techniques.* IEEE Communications Magazine, Vol. 38, Nr. 5, S. 178–184, Mai 2000.

[BDR97] G. Banga, P. Druschel. *Measuring the Capacity of a Web Server.* Proc. 1st USENIX Symposium on Internet Technologies and Systems, Monterey, Dezember 1997.

[BDM98] G. Banga, P. Druschel, J. C. Mogul. *Better operating system features for faster network servers*. Proc. 1st Workshop on Internet Server Performance, Madison, Juni 1998.

[CFTW01] M. Cieslak, D. Forster, G. Tiwana, R. Wilson. *Web Cache Communication Protocol V2.0.* Internet Draft, IETF, Expires Oct 2001.

[CMT01] I. Cooper, I. Melve, G. Tomlinson. *Internet Web Replication and Caching Taxonomy.* RFC 3040, IETF, Januar 2001.

[Coc97] A. Cockroft. *How does Solaris 2.6 improve performance stats and Web performance?* Unix Insider, August 1997.

[FCD+99] A. Feldmann, R. Caceres, F. Douglis, G. Glass, M. Rabinovich. *Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments.* Proc. IEEE Infocom ’99, S. 107–116, 1999.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1.* RFC 2616, IETF, Juni 1999.

[GwSe96] J. Gwertzman, M. Seltzer. *World Wide Web Cache Consistency.* Proc. USENIX 1996 Annual Technical Conference, San Diego, Januar 1996.

[KMK99] B. Krishnamurthy, J. C. Mogul, D. M. Kristol. *Key Differences between HTTP/1.0 and HTTP/1.1.* In Proc. 8th WWW Conference, Toronto, Mai 1999.

[KrMo00] D. Kristol, L. Montulli. *HTTP State Management Mechanism.* IETF, RFC 2965, Oktober 2000.

[LiCa97] C. Liu, P. Cao. *Maintaining Strong Cache Consistency in the World-Wide Web.* Proc. 17th International Conference on Distributed Computing Systems, Baltimore, Mai 1997.

[MEW00] A. Mahanti, D. Eager, C. Williamson. *Temporal Locality and its Impact on Web Proxy Cache Performance.* Performance Evaluation Journal: Special Issue on Internet Performance Modelling, Vol. 42, Nr. 2/3, S. 187–203, September 2000.

[Mog95b] J. C. Mogul. *Operating system support for busy interent servers.* Proc. 5th Workshop on Hot Topics in Operating Systems, Orcas Islands, Mai 1995.

[Pol01] *Web Polygraph.* http://www.web-polygraph.org/, Oktober 2001.

[Res00] E. Rescorla. *HTTP Over TLS.* RFC 2818, IETF, Mai 2000.

[RoSo99] A. Rousskov, V. Soloviev. *A performance study of the Squid proxy on HTTP/1.0.* WorldWide Web Journal, Special Edition on WWW Characterization and Performance and Evaluation Vol. 2, Nr. 1/2, S. 47–67, 1999.

[RoWe00] A. Rousskov, D. Wessels. *The Third Cache-Off.* Oktober 2000. http://www.measurement-factory.com/results/public/cacheoff/N03/

[Sha86] M. Shapiro. *Structure and Encapsulation in Distributed Systems: The Proxy Principle.* Proc. 6th International Conference on Distributed Computer Systems, Cambridge, Mai 1986.

[SGHS01] E. Shriver, E. Gabber, L. Huang, C. A. Stein. *Storage management for web proxies.* Proc. 2001 USENIX Annual Technical Conference, Boston, Juni 2001.

[TML99] G. Tomlinson, D. Major, R. Lee. *High-Capacity Internet Middleware: Internet Caching System Architectural Overview.* Proc. 2nd Workshop on Internet Server Performance, Atlanta, Mai 1999.

[Wes01a] D. Wessels. *Web Caching*. O'Reilly, Juni 2001.

[Wes01b] D. Wessels. *SQUID Frequently Asked Questions.* http://www.squid-cache.org/Doc/FAQ/FAQ.html (Oktober 2001).

[WiMi99] C. E. Wills, M. Mikhailov. *Examining the Cacheability of User-Requested Web Resources*. Proc. 4th International Web Caching Workshop, San Diego, Juni 1999.