Diplomarbeit

Tracebasierte Simulation des WWW-Zugriffsverhaltens zur Bestimmung optimaler Replikationsstrategien

Andreas Lauer

Oktober 1996

Fachbereich Informatik
Universität Kaiserslautern

ausgegeben von Prof. Dr. rer. nat. Jürgen Nehmer
betreut von Dipl.-Inform. Michael Baentsch

Erklärung:

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kaiserslautern, den 14.10.96 Andreas Lauer

Zusammenfassung:

Diese Arbeit beschäftigt sich mit der Analyse von WWW-Serverstatistiken. Es werden Untersuchungen der tageszeitabhängigen Bandbreite bei den Übertragungen zu weltweit über Proxies erreichbaren WWW-Servern durchgeführt, Modelle zur Schätzung von Seitenübertragungszeiten vorgestellt und bewertet, sowie mehrere Caching-Verfahren (TTL, Random, LCP und LRU) nach verschiedenen, im WWW relevanten Bewertungskriterien miteinander verglichen. Diese sind Byte-Hit-Rate, mittlere Seitenübertragungszeit und von den WWW-Servern übertragenes Datenvolumen. Für Bandbreite und Latenz zeigen sich Vorteile in den Nacht- und frühen Morgenstunden. Die Mittagszeit bestätigt sich als ungünstiger Zeitraum, um beispielsweise eine Aktualisierung der im Cache gehaltenen WWW-Seiten durchzuführen. Im Vergleich der Caching-Verfahren nimmt das einfach zu implementierende LRU-Verfahren den ersten Platz ein, obgleich das komplexere LCP-Verfahren bei bestimmten Konstellationen für niedrigere mittlere Übertragungszeiten sorgt und das TTL-Verfahren für große Cachevolumen zum LRU-Verfahren aufschliessen kann. Weitere Ergebnisse der Arbeit sind: Die Einrichtung eines Bereichs zur Verwaltung replizierter Seiten (meistreferenzierte Seiten aus dem Vormonat des Simulationszeitraums) bewirkt eine Verbesserung aller o.g. Bewertungsmaße. Weiterhin hat sich die Beschränkung der Größe von in den Cache einzulagernden Seiten als Mittel zur Senkung der mittleren Seitenübertragungszeit erwiesen.

Inhaltsverzeichnis

1. Einleitung 1

2. Analyse der Ausgangssituation 3

2.1 Überblick 3
2.2 Klassische Cachingverfahren 3
2.3 Cachingverfahren für WWW-Seiten 7
2.4 Replikation 24
2.5 Seitenübertragungszeiten 25

3. Das Simulationssystem 31

3.1 Zielsetzung und Entwurfsentscheidungen 31
3.2 Simulationsablauf 32
3.3 Spezifikation der Eingaben 34
3.4 Funktionalität 41
3.5 Spezifikation der Ausgaben 47
3.6 Systembeschreibung 50

4. Auswertung der Meßergebnisse 67

4.1 Überblick 67
4.2 Grundlageninformation 67
4.3 Übertragungszeitenschätzung 82
4.4 Caching-Strategien 92

5. Zusammenfassung und Ausblick 113

6. Literaturverzeichnis 117

7. Angang 119

A. Meßwerte der Cacheauswertung 119

1. Einleitung

Das Internet erlebte in den letzten Jahren einen bisher nie dagewesenen Aufschwung. Immer mehr Benutzer sind an das weltumspannende Netzwerk angeschlossen. Immer mehr Informationen werden von immer mehr Servern angeboten. Immer aufwendigere Software verlangt nach mehr Übertragungsbandbreite.

Das erhöhte Datenaufkommen macht sich deutlich mit den gesunkenen Übertragungsraten bemerkbar. Größere Datenmengen können oft nur noch in Zeiten geringerer Netzwerkbelastung übertragen werden. Transfers in Spitzenbelastungszeiten werden schnell zur Geduldsprobe. Das Defizit an Übertragungskapazität wird in Zukunft noch stärker spürbar werden; Maßnahmen zur Entlastung der Netzwerke werden weiter an Bedeutung gewinnen.

Caching ist eine der Maßnahmen, die dazu beitragen können, die vorhandenen Kapazitäten effizienter zu nutzen. Das Konzept Caching ist aus der Computerwelt nicht mehr wegzudenken. Jedes heute produzierte Computersystem besitzt einen Prozessorcache, Festplatten steigern ihre Performance, indem sie Cachingtechniken verwenden. Im Internet ist das Konzept verantwortlich für deutliche Leistungssteigerungen (z.B. bei der Verwaltung von Namensräumen im DNS), indem es Bandbreitenanforderungen reduziert und für bessere Skalierbarkeit sorgt.

Auch das World-Wide Web bietet gute Ansatzpunkte, um Caching nutzbringend einzusetzen. Proxy-Server zum Beispiel eignen sich aufgrund ihrer zentralen Stellung innerhalb eines geschlossenen Subnetzes für den Einsatz von Caching – Daten können nur über den Proxy-Server in oder aus dem Netz gelangen. Daten können hier effizienter gespeichert werden. Von im Cache gehaltenen Daten können u.U. mehrere Clients profitieren, die bessere Informationsgrundlage macht es möglich, bessere Entscheidungen über zu speichernde oder nicht zu speichernde Daten zu treffen.

Es stellen nun sich eine Reihe von Fragen, die zum Teil elementarer Natur sind:

1. Gelten die Erkenntnisse des „klassischen" Caching auch für das Word-Wide Web?
   Welchen Einfluß haben variable Seitengröße oder server- und tageszeitabhängige Ladezeiten auf die verwendeten Heuristiken?
2. Welche Heuristiken sind für dieses Szenario geeignet?
   Liefern bekannten Verfahren (z.B. LRU) auch hier gute Resultate?
3. Welche Grenzen sind dem Caching gesetzt?
   Ist Caching alleine ausreichend, um den Anforderungen im WWW zu genügen?
   Wie weit kann z.B. Seitenreplikation, wie sie das CgR-Konzept [BAE96] vorsieht, die Leistung verbessern?

Punkt eins der Liste erfordert eine Überprüfung der Grundlagen, auf denen Caching im WWW beruht. Ist z.B. die Cache-Hit-Rate immer noch ein geeignetes Gütekriterium? Nach welchen Kriterien müssen Cachingverfahren im WWW bewertet werden?

Punkt zwei setzt die Möglichkeit voraus, einzelne Heuristiken unter gleichen Bedingungen gegenüberzustellen. Obwohl einige Arbeitsgruppen sich bereits mit verschieden Heuristiken beschäftigt haben, ist in der Regel ein Vergleich der einzelnen Ergebnisse problematisch, da sie auf unterschiedlichen Ausgangsdaten basierten. So könnte es sein, daß ein Cachingverfahren, das sich für einen Proxy-Server bewährt hat, bei einem Betrieb in einem Backbone nur mäßig gut ist. Ein objektiver Vergleich von verschiedenen Cachingstrategien kann daher nur in einer Simulationsumgebung stattfinden, die in der Lage ist, wiederholt dasselbe realistische Anfrageverhalten zu erzeugen und somit identische Voraussetzungen für jeden Testlauf schaffen kann. Basis hierfür bilden Tracedaten eines realen Proxy-Servers. Zu weiteren Eigenschaften dieser Umgebung gehört die leichte Variation möglichst vieler Cacheparameter genauso wie die automatisierte Durchführung kompletter Meßreihen.

Punkt drei bringt die Replikation als ergänzendes Konzept in den Blickpunkt. Zu Beginn dieser Arbeit hat sich schnell gezeigt, daß die Entwicklung entsprechender Heuristiken ohne eine genauere Analyse des Datenmaterials (in Form der Logfiles) nicht möglich ist. Die Unterstützung einer solchen Analyse durch entsprechende Werkzeuge fehlte. Die oben schon erwähnte Umgebung zur Simulation eines realistischen Anfrageverhaltens wurde deshalb um eine Analysekomponente ergänzt, die eine Datenaufbereitung und -visualisierung unterstützt. In das Simulationssystem wurde ebenfalls eine Infrastruktur integriert für das Management replizierter Seiten. Diese Funktionalität ermöglicht es, schon vorausschauend, Erkenntnisse über die Größenordnung der zu erwartenden Gewinne zu erhalten. Außerdem ermöglicht sie eine reibungslose Integration und Validierung neu entwickelter Replikationsheuristiken.

Im folgenden sollen die wesentlichen Gesichtspunkte noch einmal aufgeführt werden, unter denen das Simulationssystem entwickelt wurde:

- Tracebasiert
- Unterstützung der Logfile-Analyse
- Flexible Manipulation der Cacheparameter
- Automatisierte Durchführung von Testreihen
- Management für replizierte Seiten
- Leichte Integration neuer Strategien

Das System wurde auf den Namen CaSimiR getauft, was für <u>Ca</u>che-<u>Sim</u>ulator mit <u>i</u>ntegriertem <u>R</u>eplikationsmanagement steht.

Kapitel 2 geht nun im weiteren näher auf Grundlagen des Caching ein. Unterschiede des Caching im WWW im Vergleich zum klassischen Caching werden aufgezeigt, Replikation als Ergänzung und weitere Grundlagen, die u.a. für die Simulation benötigt werden (Schätzung von Übertragungszeiten) sind Thema dieses Abschnitts. Während sich Kapitel 3 mit dem Simulationssystem selbst beschäftigt – hier werden Anforderungen an das Softwaresystem, dessen Funktionalität und Design sowie Implementierungsaspekte genauer beleuchtet –, wendet sich Kapitel 4 den durchgeführten Messungen zu. Themen hier sind eine Beschreibung der durchgeführten Simulationen, eine Auswertung der Meßergebnisse und ihre weiterreichende Bewertung. Kapitel 5 schließt mit einer Zusammenfassung und zeigt mögliche Ansatzpunkte für die weitere Arbeit auf.

2. Analyse der Ausgangssituation

2.1 Überblick

Neue Heuristiken zu finden und alte zu verbessern erfordert zunächst eine genaue Analyse der Ausgangssituation. Ohne die Kenntnis der Funktionsprinzipien bleibt ein Erfolg dem Zufall überlassen. Entwurfsentscheidungen können ohne ausreichende Grundlagen nicht getroffen werden. Dabei entsteht die Gefahr, daß möglicherweise zu viel Zeit in das Tuning schlechter Verfahren investiert wird, deren Potential auch dann immer noch nicht an das von guten heranreicht.

Dieses Kapitel wird sich daher zunächst mit den Grundlagen des Caching beschäftigen. In Abschnitt 2.2 werden die elementaren Prinzipien vorgestellt, auf denen Caching basiert und klassische Cachingverfahren beschrieben. Darauf folgt in Abschnitt 2.3 eine Gegenüberstellung mit Varianten, die besser auf die veränderten Bedingungen im World-Wide Web (z.B. variable Ladezeiten und Seitengrößen) abgestimmt sind. Mit der Frage, wie Caching durch ergänzende Konzepte, insbesondere durch die Replikation, unterstützt werden kann und welche Vorteile sich daraus ergeben, beschäftigt sich Abschnitt 2.4.

Der letzte Abschnitt widmet sich dann schließlich einer Problematik, die sowohl für das vorgestellte Replikationskonzept als auch für die Simulation selbst von Bedeutung ist: das Schätzen von Übertragungszeiten. Drei getestete Schätzmodelle werden vorgestellt und auf die speziellen Anforderungen an die Modelle eingegangen.

2.2 Klassische Cachingverfahren

Im gleichen Maße, wie die Entwicklung der Prozessoren voranschreitet, wird auch die Weiterentwicklung der Peripherie notwendig, um die gewonnene Leistung beispielsweise nicht wieder durch Wartezyklen einzubüßen. Speichermedien (Halbleiter, permanente Speicher) und Datenübertragungssystemen (Busse, Netzwerke) gilt hier besonderes Interesse. In beiden Fällen gilt: je schneller desto teurer. Um einen Kompromiß zu wahren zwischen benötigter Kapazität des Speichers, dessen Geschwindigkeit und den Kosten des Gesamtsystems, spielt das Konzept des Caching eine wichtige Rolle.

Caching baut auf eine Speicherhierarchie auf, wie sie in Abb. 2-1 angedeutet wird. Die grundlegende Idee dabei ist, gerade benötigte Daten in einem in der nächst niedrigeren Hierarchiestufe angesiedelten und damit schnelleren Speicher zu halten. Jeder Speicher enthält also immer einen Ausschnitt der nächsten darüberliegenden Stufe. So kann zum Beispiel der Prozessor in günstigen Konstellationen mit den Daten im Cache und in den Registern arbeiten, ohne auf den Hauptspeicher zugreifen zu müssen. Die Zugriffe erfolgen schneller und der Bus wird entlastet – er kann beispielsweise für eine parallele DMA-Übertragung genutzt werden. Zusätzlich zur effizienteren Nutzung des teureren Speichermediums kommen also noch höhere Verarbeitungsgeschwindigkeit und Entlastung der Übertragungsmedien. Caching ist aber nicht nur zwischen Prozes-

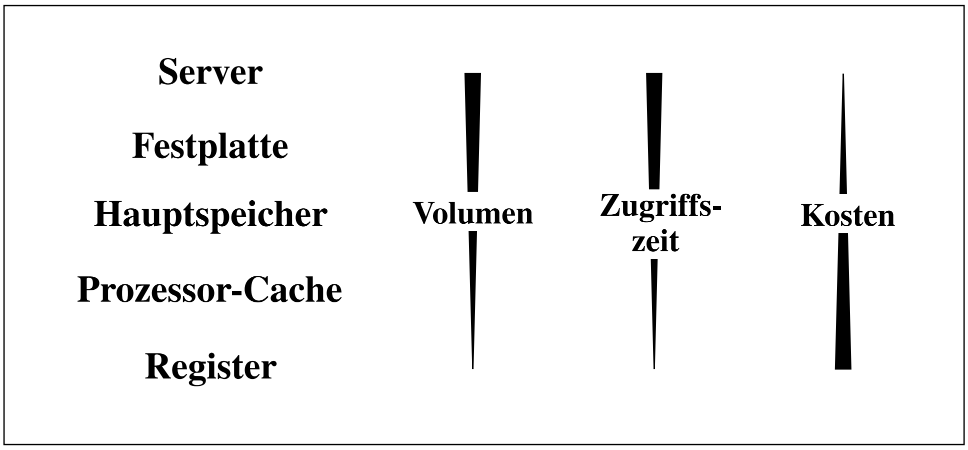


Abbildung 2-1: Speicherhierarchie

sor und Hauptspeicher denkbar, sondern prinzipiell zwischen allen Ebenen der Speicherhierarchie. Entscheidend für ein Funktionieren dieser Technik, ist eine Eigenschaft, die als *Referenzlokalität* bezeichnet wird. Dieser Begriff beschreibt das Verhalten von Prozessen, zu einem Zeitpunkt bestimmte Speicherbereiche oft zu referenzieren (diese sollten im schnelleren Speicher gehalten werden) und andere wenig oder gar nicht. Je besser dieses Referenzverhalten ausgeprägt ist, desto effektiver kann ein Cache arbeiten. Für ein Programm, das in zufälliger Art und Weise auf den Speicher zugreift, sind keine besonderen Vorteile durch Caching zu erwarten.

In Verbindung mit virtueller Speicherverwaltung wurden genauere Untersuchungen über die Referenzlokalität von Prozessen gemacht [DEN70]. Ähnlich wie beim Hauptspeicher-Caching, das den Arbeitsspeicher in kleine atomare Blöcke unterteilt (mehrere Byte groß), die bei Bedarf in den Cache geladen werden, wird der Adreßraum eines Prozesses bei der virtuellen Speicherverwaltung seitenweise (mehrer Kilobyte groß) verwaltet. Zwischenzeitlich nicht benötigte Seiten, werden auf der Festplatte gespeichert, während die Seiten, auf denen der Prozeß gerade arbeitet, im Hauptspeicher stehen. Näherungsweise läßt sich die Menge von Seiten, die zu einem bestimmten Zeitpunkt bevorzugt von einem Prozeß referenziert werden, aus dem vorhergehenden Referenzverhalten ableiten. Diese Näherung wird auch als *Working-Set* bezeichnet und beschreibt die Menge von Seiten, die für die derzeitige Arbeit benötigt wird. In die Menge werden alle Seiten aufgenommen, die bei den letzten n Zugriffen referenziert wurden. Die Größe n des Beobachtungsfensters beeinflußt, wie genau das Working-Set das tatsächliche Referenzverhalten wiedergibt. Ist n zu klein gewählt, fehlen u.U. Seiten, ist n zu groß gewählt, werden möglicherweise noch Seiten eines vorangegangenen Lokalitätsintervalls erfaßt und das Working-Set enthält zu viele Seiten. Ziel eines Caching-Verfahrens muß es nun sein, im laufenden Betrieb das aktuelle Working-Set möglichst genau zu bestimmen und im Cache zu halten. Je nachdem zwischen welchen Ebenen der Speicherhierarchie der Cache angesiedelt ist, fällt das Seitengranulat feiner oder gröber aus.
Alle in diesem Kapitel vorgestellten Verfahren gehen von den Annahmen aus:

- Seiten[1] haben immer gleiche Größe;
- die Zugriffszeiten auf alle Seiten sind ungefähr gleich und schwanken nur unwesentlich, Zeitvorteile durch Caching sind also für alle Seiten gleich;

1. Die Begriffe „Seite” und „Block” werden synonym verwendet

- Gütekriterium für eine Heuristik ist die Cache-Hit-Rate,
  sie gibt an, wie oft Seiten aus dem Cache geladen werden konnten, ohne daß auf das langsamere Speichermedium (z.B. Hauptspeicher oder Festplatte) zugegriffen werden mußte. Eine maximale Trefferquote führt zu einer minimalen durchschnittlichen Seitenladezeit.

Abb. 2-2 zeigt, welche Entscheidungen bei einem Cachezugriff getroffen werden müssen. Für die neue Seite muß zunächst entschieden werden, ob sie in den Cache aufgenommen werden soll. Hier spielen technische Gründe eine Rolle, die z.B. die Aufnahme bestimmter Adreßräume, in die Ein- und Ausgabegeräte eingeblendet sind, verbieten. Soll die Seite eingelagert werden, überprüft das Verfahren, ob die angeforderte Seite bereits im Cache gespeichert ist. War dies nicht der Fall, muß die Seite vom

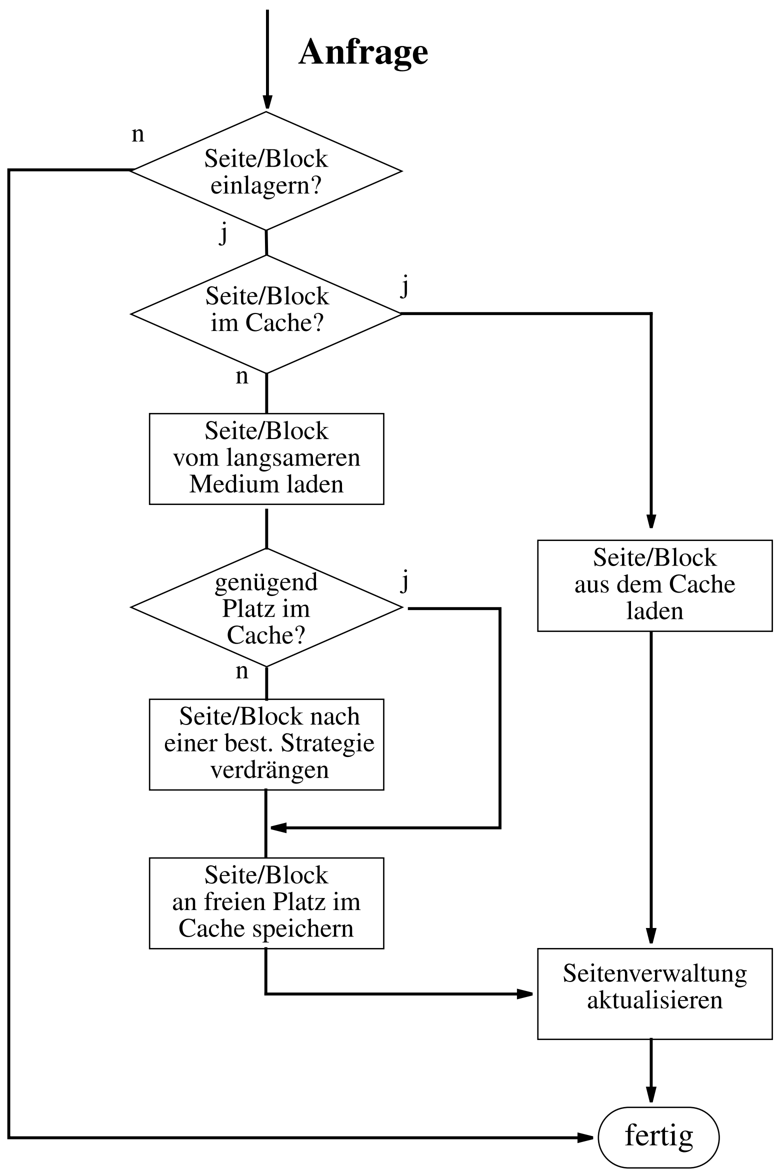


Abbildung 2-2: Flußdiagramm eines klassischen Caching-Verfahrens

langsameren Medium, z.B. dem Hauptspeicher oder der Platte, geladen werden. Caches arbeiten also, was das Einlagern der Seiten anbelangt, immer mit einer Demand-Paging-Strategie, d.h. Seiten werden erst *nach* einem Zugriff in den Cache geladen. Andere sog. Pre-Paging-Strategien, die Seiten schon *vor* einem ersten Zugriff in den Cache laden, werden erst im Hinblick auf das World-Wide Web interessant (Stichwort: Replikation).

Soll die Seite eingelagert werden, muß, falls nicht schon vorhanden, Platz für die neue Seite durch Auslagern einer anderen geschaffen werden. Die Strategie nach der diese Seite gewählt wird, bestimmt maßgeblich die Qualität – sprich, die Hit-Rate – des Caches. Im Allgemeinen trägt auch das Verfahren den Namen der verwendeten Ersetzungsstrategie.

Es folgen Cachingverfahren, die mit den o.g. Annahmen arbeiten:

2.2.1 Optimal

Das optimale Verfahren nach Belady [BEL66] ist ein theoretisches, das in der Lage ist, in die Zukunft zu schauen. Es wird die Seite verdrängt, welche erst wieder am weitesten in der Zukunft benötigt wird. Es kann auch vorkommen, daß eine Seite überhaupt nicht in den Cache aufgenommen wird (obwohl referenziert), weil sie in der Zukunft nicht mehr benötigt wird. Auf diese Weise wird immer die optimale Menge an Seiten im Cache gehalten, die eine maximale Cache-Hit-Rate gewährleistet. Diese Betrachtungen gelten nur unter den auf Seite 4 genannten Voraussetzungen: gleiche Seitengröße und gleiche Caching-Gewinne für alle Seiten!

Wenngleich das Verfahren nicht für einen praktischen Einsatz geeignet ist, bietet es doch eine gute Vergleichsbasis für die übrigen nun folgenden Heuristiken. Real einsetzbare Verfahren haben nicht die Möglichkeit, in die Zukunft zu blicken. Sie müssen zukünftige Referenzfolgen auf der Basis vergangener Referenzen schätzen. Dies führt u.a. dazu, daß jede referenzierte Seite in den Cache eingelagert wird, da davon ausgegangen werden muß, das dieselbe Seite auch in der nächsten Zukunft wieder referenziert wird.

2.2.2 Random

Bei der Random-Strategie wird eine zufällig gewählte Seite aus dem Cache verdrängt. Eine Referenzlokalität wird ignoriert, häufig referenzierte Seiten werden genauso behandelt, wie selten referenzierte.

2.2.3 FIFO

Als Verbesserung zur Random-Strategie, wird hier die Vergangenheit mit in die Verdrängungsentscheidung mit einbezogen. Kandidat für eine Auslagerung ist die Seite, die bis dahin am längsten im Cache gelagert wurde. Die Verweildauer im Cache ist allerdings die einzige Information, die für eine Verdrängung herangezogen wird. Häufige Referenzen auf bestimmte Seiten bleiben unbeachtet. Eine früh eingelagerte Seite wird verdrängt, obwohl sie möglicherweise seit diesem Zeitpunkt häufig verwendet wurde. Sie zu verdrängen würde aber bedeuten, eine Seite des oben erwähnten Working-Sets auszulagern.

2.2.4 LFU

LFU steht für Least Frequently Used. Die Verdrängung wird auf der Basis der Referenzhäufigkeit getroffen. Dazu wird jeder Cacheseite ein Zähler zugeordnet. Eine Seitenbenutzung wird durch die Inkrementierung des jeweiligen Zählers festgehalten. Niedrige Zählerstände lassen nun auf eine nachlassende Beliebtheit schließen, und qualifizieren die zugehörige Seite für die Auslagerung. Um nicht in der Vergangenheit häufig referenzierte aber in letzter Zeit wenig oder gar nicht verwendete Seiten zu übervorteilen – eine Seite mit hohem Zähler könnte sich im Cache festsetzen, obwohl sie inzwischen nicht mehr benötigt wird – erfolgt in periodischen Zeitabständen ein Rücksetzen aller Zähler auf Null. Das Verfahren besitzt die Tendenz neue Seiten, die insbesondere am Ende des Rücksetzintervalls in den Cache kommen, mit größerer Wahrscheinlichkeit wieder zu verdrängen. Bei großen Caches kann sich das Zurücksetzen der Zähler als aufwendig erweisen.

2.2.5 Second Chance

Referenzierte Seiten bekommen bei diesem Verfahren eine zweite Chance, bevor sie ausgelagert werden. Dazu sind alle Seiten in einer zirkulären Liste eingetragen. Ein Flag zeigt eine zwischenzeitliche Referenzierung an. Um einen Verdrängungskandidaten zu ermitteln, wird die Liste rundum nach der ersten Seite mit nicht gesetztem Referenz-Flag durchsucht und diese verdrängt. Trifft das Verfahren dabei auf Seiten mit gesetztem Flag, werden diese zunächst verschont und das Referenzflag gelöscht. Das Referenzverhalten wird hier nur eingeschränkt berücksichtigt: Muß sich das Verfahren zwischen mehreren Seiten mit gesetztem Referenzflag entscheiden, wird die zuerst eingelagerte Seite verdrängt. In diesem Fall degeneriert das Verfahren also zum einem FIFO-Verfahren.

2.2.6 LRU

Das LRU-Verfahren (<u>L</u>east <u>R</u>ecently <u>U</u>sed) ist am stärksten an die Betrachtungen angelehnt, die zur Referenzlokalität und dem Working-Set weiter oben gemacht wurden. Verschiebt sich das aktuelle Lokalitätsintervall, kommen neue Seiten zum Working-Set hinzu, andere fallen aus der Menge heraus – sie werden nicht mehr (oder nur noch sehr selten) referenziert. Dies macht sich das LRU-Verfahren zunutze. Alle Seiten im Cache werden nach ihrem letzten Referenzierungszeitpunkt geordnet. Die Seite, deren letzter Benutzungszeitpunkt am weitesten in der Vergangenheit liegt, wird verdrängt. Der Algorithmus nähert das optimale Verfahren am besten an und zeichnet sich darüber hinaus durch eine einfache Implementierbarkeit aus.

2.3 Cachingverfahren für WWW-Seiten

Abb. 2-3 zeigt ein mögliches Einsatzszenario für einen WWW-Cache. WWW-Server in aller Welt bieten Informationen an. Das im World-Wide Web verwendete Hypertext Transfer Protocol (HTTP) bietet Clients die Möglichkeit diese Information in seitenorientierter Form über das Netzwerk zu laden [BER96]. Jedes Objekt besitzt einen eindeutigen Bezeichner (Uniform Resource Locator URL, [W3C]) aus dem der anbietende Server hervorgeht. Der Client baut eine Verbindung zu dem entsprechenden Server auf und fordert das gewünschte Objekt explizit an. Solange das Objekt nicht fertig übertragen ist oder anderweitige Probleme bei der Übertragung auftreten (z.B. Netzwerkfehler) bleibt die Verbindung bestehen.

In dem hier betrachteten Szenario befinden sich alle Clients hinter einer sog. Firewall. Aus Sicherheitsgründen können sie nur untereinander frei miteinander kommunizieren. Der Kontakt zur Außenwelt kann lediglich über einen Proxy-Server abgewickelt werden.
Der Proxy-Server steht an zentraler Position. Anfragen der Clients hinter der Firewall werden von ihm stellvertretend an die Server in aller Welt weitergeleitet. Die entsprechenden Antworten müssen ebenfalls erst den Proxy-Server passieren, bevor sie zurück

zu den Clients gelangen. Auf diese Weise passieren viele Daten in Form der WWW-

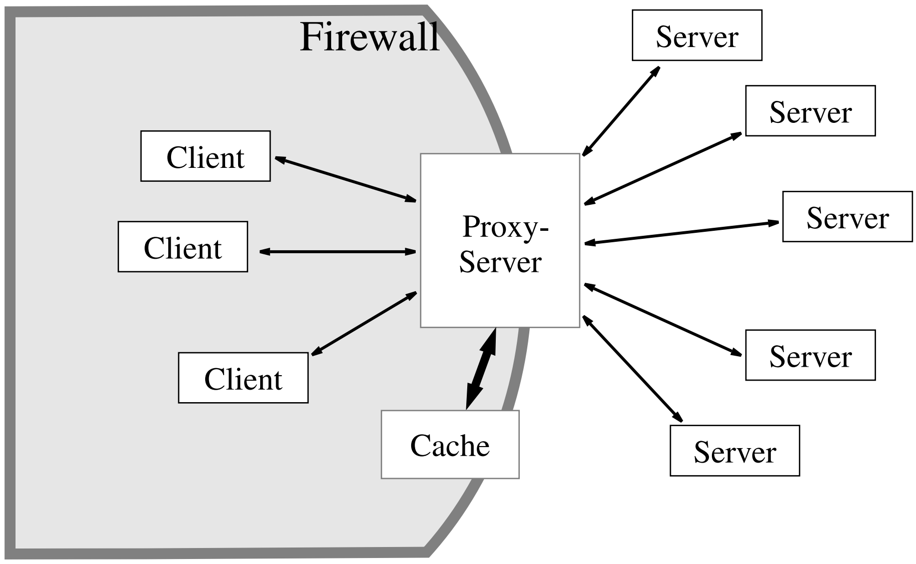


Abbildung 2-3: Proxy-Server Szenario

Seiten den Server. Mehrere Clients, die auf gleiche Seiten zugreifen, profitieren genauso von einer Zwischenspeicherung der Seiten in einem Cache wie ein einzelner, der oft gleiche Seiten, die Themen aus seinem Interessengebiet anbieten, lädt [CAT95].

Bevor auf die speziell auf das World-Wide Web abgestimmten Verfahren eingegangen wird, ist zunächst ein Blick auf die veränderten Rahmenbedingungen notwendig:

- **unterschiedliche Seitengrößen**
  die Größe einer WWW-Seite kann zwischen wenigen hundert Byte und mehreren MByte variieren. Oft reicht es nicht mehr aus, eine einzige Seite aus dem Cache zu verdrängen. Große Seiten, die in den Cache eingelagert werden sollen, können u.U. die Auslagerung von sehr vielen kleinen, häufig gebrauchten Seiten bewirken. Eine Beschränkung der maximalen Seitengröße könnte zum Beispiel hier den Cache entlasten und das Problem lindern. Zusätzliche Schwierigkeiten können durch Fragmentierung des Cachespeichers entstehen – obwohl genügend freier Cachespeicher vorhanden wäre, existiert in diesem Fall kein zusammenhängender Speicherbereich mit der notwendigen Größe.
- **Serverspezifische Ladezeiten für WWW-Seiten**
  die Server, von denen die Seiten angefordert werden, sind über die ganze Welt verteilt. Entsprechend unterschiedlich sind die Ladezeiten.
- **Starke Variation des Gewinnpotentials verschiedener Seiten**
  zusammenhängend mit den unterschiedlichen Ladezeiten ergibt sich für jede Seite ein anderer Caching-Gewinn. Von nahegelegenen[1] Servern werden die Seiten kaum spürbar schneller übertragen als aus dem Cache, während der Vorteil bei weit entfernten Servern signifikant ist. Eine Auflösung der demokratischen Behandlung gespeicherter Seiten muß in Betracht gezogen werden. Denkbar wäre ein besonde-

1. Der Begriff „nah” bezieht sich hier weniger auf eine geographische als auf eine netzwerktechnische Entfernung, wenngleich dies oft zusammenfällt.

rer Schutz für besonders gewinnbringende Seiten oder, wie es bei der Replikation der Fall ist, ein spezieller Bereich, in dem diese Seiten gesondert gehalten und verwaltet werden.

- **Seiteninvalidierung**
  Auf den WWW-Servern werden Seiten geändert, ohne daß eine explizite Invalidierungsnachricht an die Clients (und damit auch den Proxy-Server) abgesetzt wird. Die gespeicherten Seiten dürfen also nicht beliebig lange im Cache verbleiben, ohne daß ihre Gültigkeit überprüft wird. Techniken zur Erhaltung der Cache-Konsistenz, wie sie in [GWE96] beschrieben werden, kommen hier zum Einsatz.

- **Cache-Hit-Rate kein ausreichendes Gütekriterium mehr**
  Eine hohe Trefferquote im Cache ist nicht mehr alleine ausschlaggebend für die Beurteilung eines Cachingverfahrens. Andere Kriterien für die Cachebeurteilung treten nun in den Vordergrund:

  – Byte-Hit-Rate
    eine entsprechend der Seitengröße gewichtete Cache-Hit-Rate, die besser das Verhältnis zwischen aus dem Cache und von den Servern übertragenen Daten widerspiegelt.

  – Durchschnittliche Seitenladezeit
    sie ist vor allem für den Client von Vorteil, dessen Wartezeit entsprechend geringer ausfällt.

  – Netzwerkbelastung
    gemessen in „Anzahl der Zugriffe auf die Server" oder „Datenaufkommen zwischen Proxy und den Servern". Dieser Gesichtspunkt ist besonders für Netzwerkbetreiber interessant. Er ermöglicht die effiziente Nutzung vorhandener Übertragungskapazitäten.

  – Aktualität der gelieferten Seiten
    angegeben mit der Quote, mit der veraltete Seiten aus dem Cache geliefert wurden (stale-rate).

- **Tageszeitabhängige Übertragungszeiten**
  Die günstigeren Übertragunszeiten in den Nachtstunden sollten, z.B. in einer aktiven Ergänzung des typischerweise passiven Caching für eine Aktualisierung der gespeicherten Seiten genutzt werden.

- **Gleichzeitige Anfragen**
  Das Cache-System muß nun in der Lage sein, gleichzeitige Anfragen zu bedienen. Insbesondere Kollisionen bei Auslagerungen von Seiten sind zu beachten.

Der nächste Abschnitt erläutert an einigen Beispielen, welche problematischen Situationen beim Caching von WWW-Seiten zu beachten sind. Abschnitt 2.3.2 wendet sich dann der Konsistenzerhaltung des Caches zu. Das Kapitel schließt mit der Vorstellung zweier Verfahren, in die die zuvor gewonnenen Erkennisse eingeflochten wurden.

2.3.1 „Optimal"

Einige kurze Beispiele sollen zunächst mit der WWW-Caching-Problematik vertraut machen und aufzeigen, welche Faktoren in ein optimales Verfahren einfließen müssen. Im Anschluß daran wird ein formaler Algorithmus vorgestellt, der die optimale Seitenmenge im Cache hält, um eine möglichst geringe durchschnittliche Seitenladezeit zu erreichen. Ein Nachweis für dessen Optimalität steht noch aus – deshalb steht der Begriff „Optimal” auch in Anführungszeichen.

Vorab noch ein paar Erläuterungen zu der verwendeten Notation (siehe Abb. 2-4): Auf der x-Achse wird die Zeit, auf der y-Achse die Seitenbezeichnung abgetragen – in der Regel sind alle Seiten der Reihe nach durchnumeriert. Die Rechtecke in den Diagrammen geben für jede Seite an, wann sie referenziert wird (linker Rand des Rechtecks), wie lange ein Laden der Seite aus dem Cache dauern würde (bis zu rechten Rand des schraffierten Teils) und den Zeitpunkt, an dem ein Laden – direkt vom Server – abgeschlossen sein würde (rechtes Ende). Der weiße Bereich gibt also den möglichen durch Caching erreichbaren Zeitgewinn an. Ein dunkler Hintergrund hinter einem Rechteck deutet an, daß sich diese Seite in diesem Zeitraum im Cache befindet. Fehlt der Hintergrund muß die entsprechende Seite vom Server geladen und u.U. danach in den Cache eingelagert werden. Die einzelnen abgetragenen Zeitpunkte werden unterschieden nach Referenzzeitpunkten – zusammengefaßt in der Menge „TRef” – und Zeitpunkten, an denen einzelne Seiten komplett vom Server geladen sein könnten. Letztere bilden die Menge „TEnd”. Befindet sich zu einem Zeitpunkt $t \in$ TEnd die zugehörige Seite nicht im Cache, ist „t” ein sog. Entscheidungszeitpunkt, an dem über eine Einlagerung entschieden werden muß.

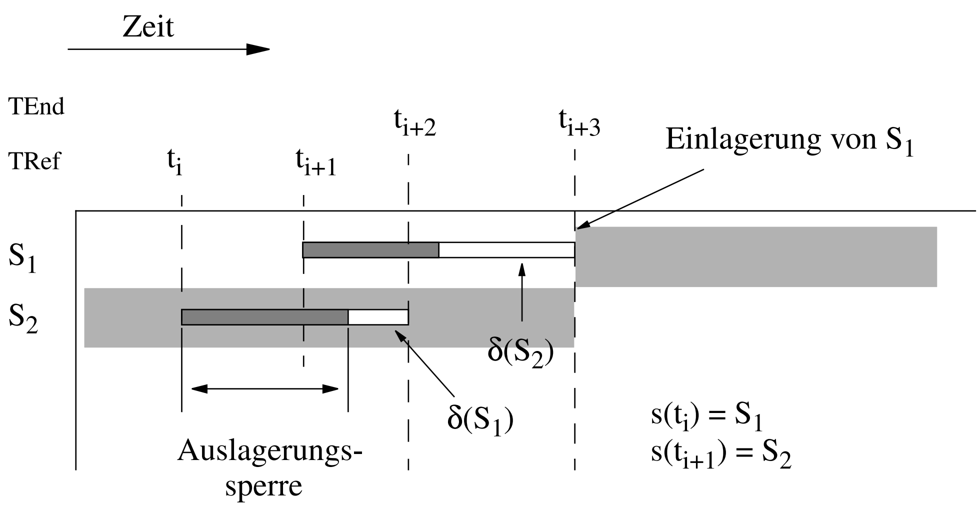


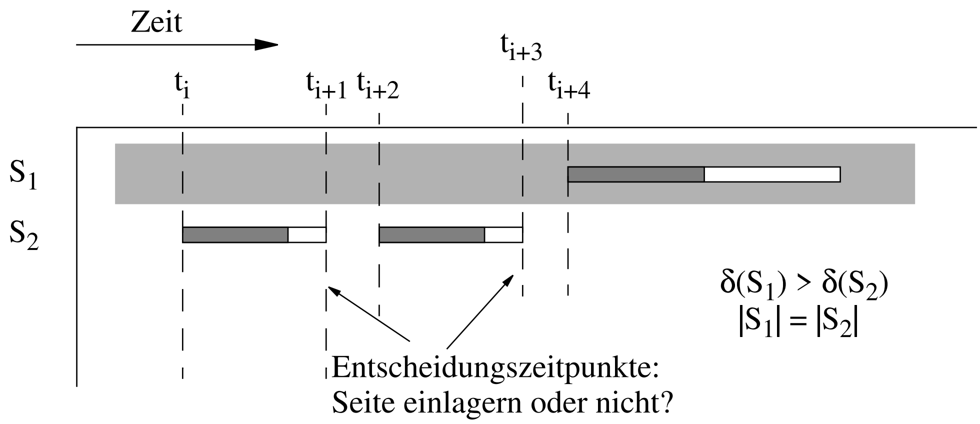
Abbildung 2-4: Seitenreferenz-Diagramm

Mit $s(t_i)$ wird die zum Zeitpunkt t_i referenzierte Seite bezeichnet. $|s|$ steht für die Größe der Seite[1] und $\delta(s)$ für den möglichen Gewinn, der durch Caching der Seite „s” erzielt werden kann. Der Einfachheit halber wird angenommen, daß der Gewinn zu jedem Zeitpunkt konstant ist, obwohl in der Realität eine Abhängigkeit von der Tageszeit zu beobachten ist. An den prinzipiellen Beobachtungen ändert dies jedoch nichts, macht aber die verwendeten Notationen übersichtlicher und besser verständlich.

Beispiel 2-1 zeigt eine erste typische Situation: Seite S_1 ist in den Cache eingelagert, zum Zeitpunkt t_i wird die Seite S_2 zum erstenmal referenziert. Nachdem die Seite vom Server geladen wurde (t_{i+1}), muß nun entschieden werden, ob sie in den Cache genom-

1. Obwohl Seitengröße und Ladezeit oft zusammenhängen, ist es doch möglich, daß kleine Seiten von weit entfernten Servern eine lange Ladezeit haben. Solche Seiten können im Diagramm fälschlicherweise den Eindruck erwecken, große Seiten zu sein.

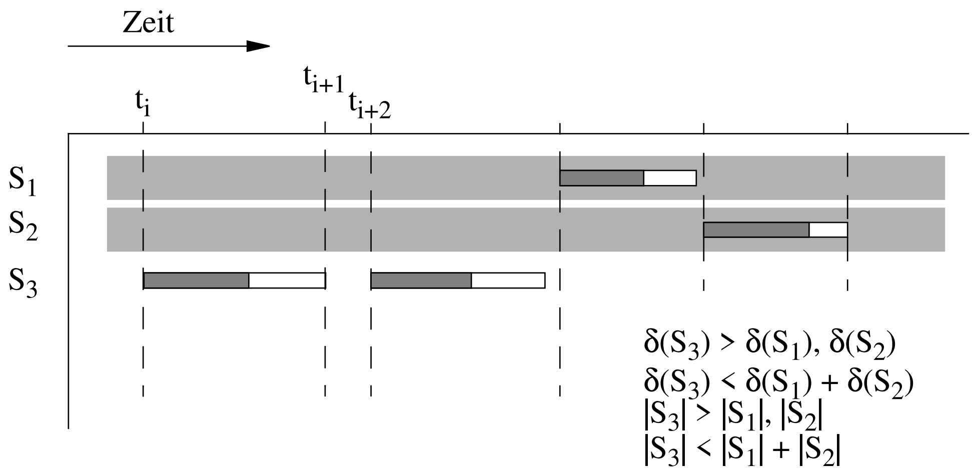
men wird oder nicht. Im ersten Fall müßte dafür Seite S_1 ausgelagert werden – wir nehmen an, daß beide Seiten nicht zusammen im Cache gehalten werden können. Um eine Entscheidung treffen zu können, müssen die möglichen Auswirkungen betrachtet werden. Tritt Seite S_2 an die Stelle von S_1, ergibt sich zum Zeitpunkt t_{i+2} ein Cache-Hit und damit verbunden ein Zeitvorteil $\delta(S_2)$ gegenüber einem normalen Laden. Da Seite S_1 nun ausgelagert ist, ergibt sich bei t_{i+4} ein Cache-Miss, also kein zusätzlicher Gewinn mehr. Bleibt S_1 im Cache-Speicher, führt nach dem ersten auch der zweite Zugriff auf Seite S_2 zu einem Cache-Miss. Dennoch fällt der Gewinn mit $\delta(S_1)$ größer aus, S_1 kann nämlich nun aus dem Cache geladen werden. Die Entscheidung, die ein optimaler Caching-Algorithmus zum Zeitpunkt t_{i+1} treffen muß, lautet also: Seite S_2 verwerfen. Man beachte, daß das klassische optimale Verfahren von Belady Seite S_1 ausgelagert hätte.



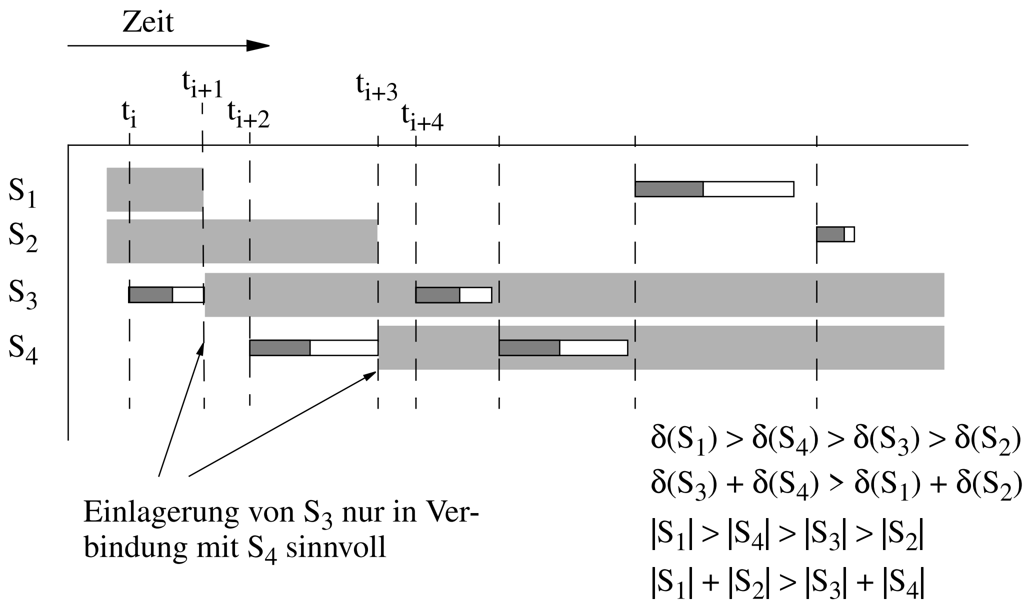
Beispiel 2-1: Einlagerung von S_2 scheitert am zu geringen Gewinn

Der Faktor „Seitengröße” kommt in Beispiel 2-2 in den Blickpunkt: Der Cache ist voll gefüllt mit den Seiten S_1 und S_2. Seite S_3 wird zum Zeitpunkt t_i referenziert. Die zweite Referenz auf S_3 würde einen Gewinn $\delta(S_3)$ ergeben – mehr als von Seite 1 oder Seite 2 zu erwarten ist. Seite 3 muß also eingelagert werden? Die Antwort auf diese Frage ist „Nein”. Grund dafür ist die Größe von S_3. Der Cache ist bereits gefüllt, die Verdrängung von S_1 *oder* S_2 bringt nicht genug Platz im Cache. Folglich müßten beide Seiten S_1 *und* S_2 verdrängt werden. Der Gewinn $\delta(S_3)$ würde also auf Kosten der in der Summe höheren Gewinne $\delta(S_1)$ und $\delta(S_2)$ gehen. Insgesamt gesehen wäre eine Einlagerung von S_3 somit die falsche Entscheidung.
Eine weitere Konstellation, bei der die Seitengröße eine wichtige Rolle spielt, ist in Beispiel 2-3 illustriert. Zum Zeitpunkt t_{i+1} muß über eine Einlagerung von S_3 entschieden werden. Seite S_1 bringt alleine zu wenig Gewinn, um eine Auslagerung von S_2 und S_3 zu rechtfertigen. Um S_1 zu verdrängen ist $\delta(S_3)$ zu gering. Um S_2 zu verdrängen ist S_3 zu groß. Erst zusammen mit S_4 wird eine Einlagerung sinnvoll. Das Beispiel zeigt, daß nachfolgende Seitenreferenzen entscheidende Auswirkungen auf eine Einlagerung haben können. Letztendlich müssen alle Ersetzungsmöglichkeiten in Erwägung gezogen werden, bevor eine endgültige Entscheidung getroffen wird.

Verschiebt sich der Referenzzeitpunkt für S_1 hinter den ersten von S_3, ergibt sich folgende Situation (Beispiel 2-4): S_3 ist zu groß, um S_2 alleine verdrängen zu können. Zusätzlich S_1 aus dem Cache zu entfernen, würde einen zu großen Gewinnverlust bedeuten. Seite S_2 wird also verworfen und ein weiterer Cache-Miss für S_3 in Kauf genommen. An dem Beispiel wird deutlich, daß eine gewinnbringende Seitenkombina-



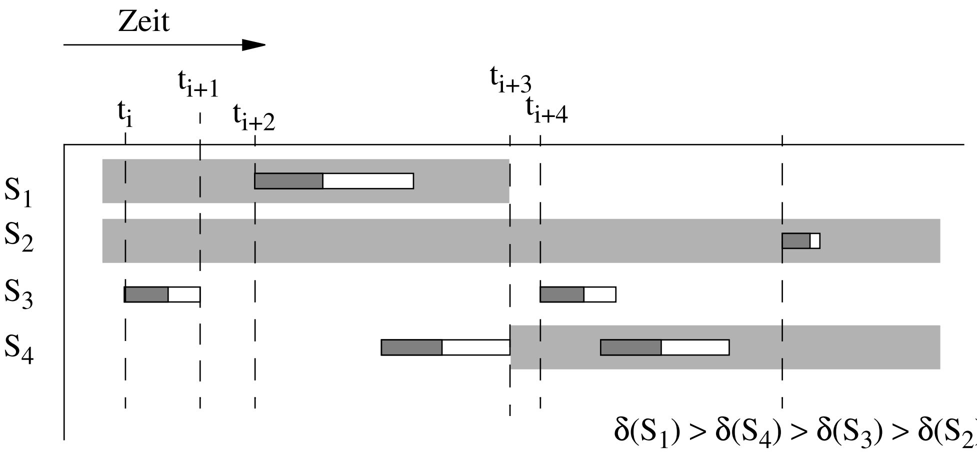
Beispiel 2-2: Mehrere Seiten müssen verdrängt werden



Beispiel 2-3: Verdrängung mit Hilfe weiterer Seiten

tion (S_3, S_4) nicht automatisch eine Menge von nominell weniger gewinnbringenden Seiten (S_1, S_2) aus dem Cache verdrängen kann. Referenz- und Verdrängungszeitpunkte der Seiten spielen eine wichtige Rolle.

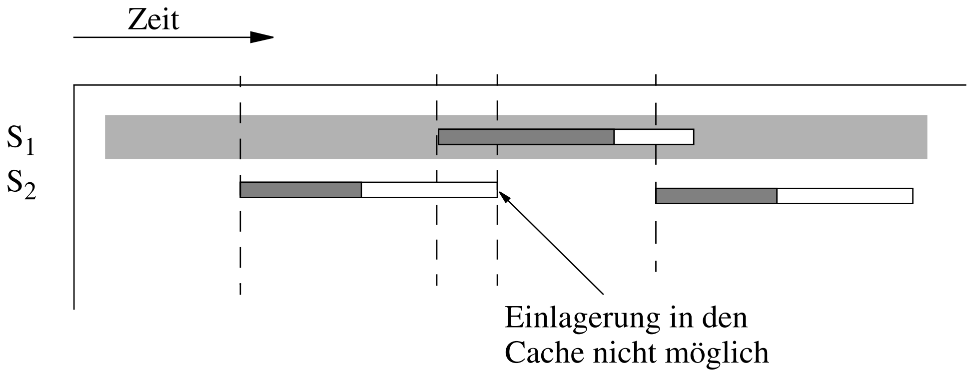
Ein weiterer Punkt, der hier angesprochen werden soll, ist die Möglichkeit, daß mehrere Seiten gleichzeitig oder überlappend aus dem Cache geladen werden können und Seiten, während sie aus dem Cache geladen werden, für eine Auslagerung nicht in Frage kommen. Die Graphik von Beispiel 2-5 zeigt Seite S_1 im Cache gespeichert. S_2 wird noch vor Seite S_1 referenziert. Zu diesem Zeitpunkt ist sie nicht im Cache gespeichert. Ein Cache-Miss und das Laden der Seite vom Server ist die Folge. Während des Ladens von S_2 wird auch S_1 referenziert. Diese Seite ist im Cache abgelegt, sie wird ohne Inanspruchnahme des entsprechenden Servers geladen. Der Ladevorgang ist noch nicht beendet, wenn Seite S_2 vollständig vorliegt und das Caching-Verfahren über eine Einlagerung entscheiden muß. Einziger Verdrängungskandidat ist Seite 1. Sie ist aber durch den parallel ablaufenden Ladevorgang gesperrt. S_2 muß also verworfen, der dar-



$$\delta(S_1) > \delta(S_4) > \delta(S_3) > \delta(S_2)$$
$$\delta(S_3) + \delta(S_4) > \delta(S_1) + \delta(S_2)$$
$$|S_1| > |S_4| > |S_3| > |S_2|$$
$$|S_1| + |S_2| > |S_3| + |S_4|$$

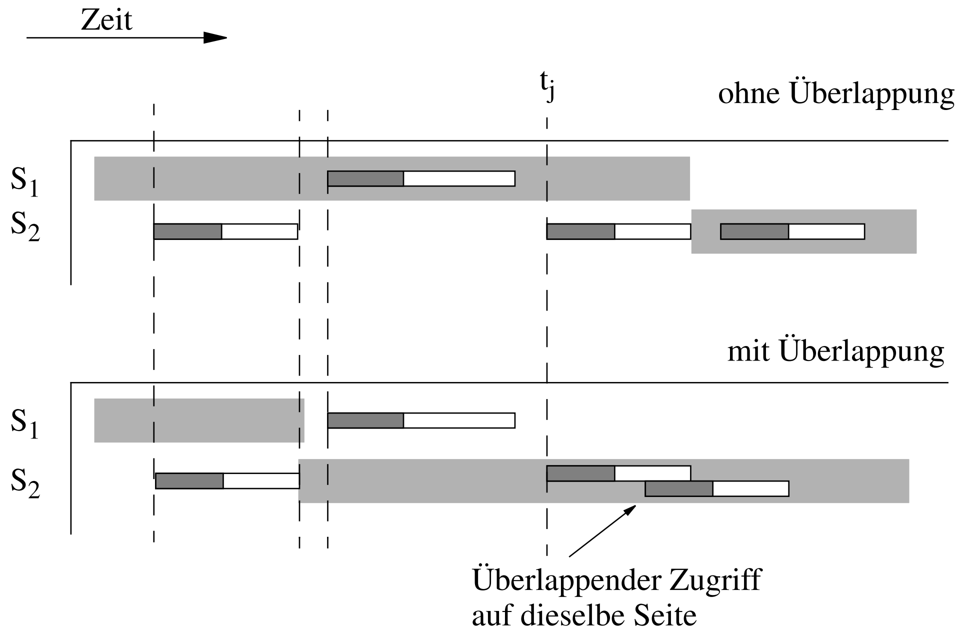
Beispiel 2-4: Veränderter Referenzierungszeitpunkt verhindert Einlagerung von S_3

auffolgende Cache-Miss hingenommen und auf den höheren Gewinn $\delta(S_2)$ verzichtet werden. Das optimale Caching-Verfahren muß u.U. schon vor solch einer Situation entscheiden, ob z.B. die Einlagerung der Seite S_2 unterlassen und somit auf diese Weise von vonherein der Konflikt verhindert wird.



Beispiel 2-5: Auslagerungssperre während des Ladens aus dem Cache

Das letzte Beispiel beschäftigt sich mit der überlappenden Referenzierung ein und derselben Seite. Seite S_2 ist in Beispiel 2-6 von solch einer Überlappung betroffen. In der oberen Seitenfolge liegt genug Zeit zwischen der zweiten und dritten Referenz auf S_2. S_2 wird beim ersten Mal nicht im Cache gespeichert, nach der zweiten Referenz kann die Seite S_1 aus dem Cache verdrängen (S_1 wird zu diesem Zeitpunkt nicht mehr benötigt) und der dritte Zugriff kann schließlich von der Einlagerung profitieren. Bei der unteren Seitenfolge überlappen sich die Ladezeiten bei der zweiten und dritten Anfrage. Befindet sich S_2 zum Zeitpunkt t_j noch nicht im Cache, kann sie auch nicht mehr rechtzeitig zur dritten Referenz vom Server geladen werden. Beide Male müßte über das Netzwerk auf den WWW-Server zugegriffen werden. Ist S_2 zum Zeitpunkt t_j bereits gespeichert, kann S_2 zweimal aus dem Cache geladen werden. Hier zeigt sich



Beispiel 2-6: Überlappende Cache-Anfragen

nochmals, daß eine Unterscheidung von Referenzzeitpunkt und Einlagerungszeitpunkt unumgänglich ist.

Es folgt nun der „optimale” Verdrängungsalgorithmus für WWW-Seiten:

Die Idee

Für jeden einzelnen Zeitpunkt $t_i \in$ TRef $\cup$ TEnd wird der optimale Cachezustand C_i auf der Basis des letzten Zustands C_{i-1} berechnet. Der Zustand gibt an, welche Seiten zu diesem Zeitpunkt im Cache gespeichert sind – er wird im weiteren wie eine Menge von Seiten behandelt. Die Berechnung der Zustände wird von der rekursiven Funktion „NextC” durchgeführt, der dazu zukünftige Referenzen und Ladezeiten für alle Seiten zur Verfügung stehen. Änderungen des Cachezustandes können nur an Entscheidungszeitpunkten $t \in$ TEnd auftreten, wenn eine neue bisher nicht im Cache befindliche Seite „s” geladen wurde. Die Frage, ob diese Seite eingelagert werden soll oder nicht, wird folgendermaßen beantwortet: „NextC” sucht eine Teilmenge C' der im Cache gespeicherten Seiten und berechnet für den Fall einer Ersetzung dieser Menge durch „s” den möglichen maximalen Gewinn im weiteren Verlauf. C' muß so gewählt sein, daß insgesamt der höchste Gewinn von allen möglichen Teilmengen erzielt wird. Das so gewonnene Ergebnis wird mit dem maximalen Gewinn verglichen, der mit dem alten Cachezustand erreichbar ist. Je nachdem welches Ergebnis das bessere war, wird eine Ersetzung und damit Änderung des Cachezustands durchgeführt oder nicht.

Anhand der so entstandenen Zustandsfolge $C_0, C_1, \ldots, C_n$ kann eine Entscheidungsfunktion „E” nun explizit angeben, ob die zu einem Zeitpunkt t geladene Seite neu in den Cache aufgenommen werden muß oder nicht. Sie gibt außerdem im Falle einer Einlagerung die (eventuell leere) Menge der aus dem Cache zu verdrängenden

Seiten an. Es folgt eine formalere Definition der zur Bestimmung der Zustandsfolge verwendeten Funktionen:

$$C_0 = \emptyset$$

$$C_{t+1} = \mathrm{NextC}(C_t, t)$$

$$\mathrm{NextC}(C,i) = \begin{cases} (C \backslash C') \cup s(t_i) & s(t_i) \notin C \text{ und } (*) \text{ ist erfüllt} \\ C \cup s(t_i) & \sum_{c \in C} |c| + |s(t_i)| \leq \text{max. Cachegröße} \\ C & s(t_i) \in C \\ C & t_i \notin \mathrm{TEnd} \\ C & \text{sonst} \end{cases}$$

(*) Es gibt ein C' mit $\emptyset \subset C' \subseteq C$ und

(1) $\sum_{c \in C'} |c| \geq |s(t_i)|$

(2) alle $c \in C'$ sind nicht gesperrt, d.h. die Seiten werden zum Zeitpunkt t_i gerade nicht aus dem Cache geladen

(3) $\Delta((C - C') \cup s(t_i), i+1) - \Delta(C, i+1)$ ist maximal und > 0

Die Δ-Funktion ermittelt den bei einem Ausgangszustand C des Caches maximal erreichbaren Gewinn bis zum Ende des simulierten Zeitraums (bis t_{max}). Sie ist folgendermaßen definiert:

$$\Delta(C, i) = \begin{cases} \Delta_i(C) + \Delta(\mathrm{NextC}(C, i+1), i+1) & t_i < t_{max}, t_i \in \mathrm{TRef} \\ \Delta(\mathrm{NewC}(C, i+1), i+1) & t_i \notin \mathrm{TRef} \\ 0 & \text{sonst} \end{cases}$$

mit

$$\Delta_i(C) = \begin{cases} \delta(s(t_i)) & s(t_i) \in C \\ 0 & \text{sonst} \end{cases}$$

Die δ-Funktion ist ein allgemeiner Gewinnfaktor – in dieser Definition beschreibt sie den durch Caching erreichbaren Zeitgewinn. Zur Optimierung anderer Kriterien, wie Netzwerkbelastung oder Byte-Hit-Rate kann sie entsprechend modifiziert werden.

Noch einige Erläuterungen zur „NextC"-Funktion: Fallunterscheidung eins versucht eine Menge C' von auszulagernden Seiten zu bestimmen. An diese werden drei Bedingungen gestellt: (1) verlangt, daß genug Platz für die neue Seite geschaffen werden muß. Daß keine Seiten ausgelagert werden, die gerade zu diesem Zeitpunkt t_i aus dem Cache geladen werden, stellt Punkt (2) sicher. Am wichtigsten jedoch ist Punkt (3). Er garantiert, daß eine optimale Seitenauswahl getroffen wird.

Die zweite Fallunterscheidung in der „NextC"-Funktion beschreibt die Einlagerung einer Seite in den Cache, ohne dafür andere auslagern zu müssen. Dies tritt zum Beispiel beim Auffüllen des noch leeren Caches auf. Fallunterscheidung drei kommt bei einem Cache-Hit zum Tragen. Der Cacheinhalt verändert sich nicht, lediglich die referenzierte Seite muß für die Dauer der Übertragung vor einer Auslagerung gesperrt werden.

Für einen vorgegebenen Referenzierungszeitpunkt t gibt nun die Funktion E an, ob die zu dieser Zeit angesprochene Seite neu eingelagert werden muß oder nicht. Ergebnis dieser Funktion ist ein Zweitupel. Die erste Komponente kann die Werte „einlagern" oder „verwerfen" annehmen. Die zweite Tupelkomponente gibt an, welche Seiten im Falle einer Einlagerung verdrängt werden müssen.

$$E((t, (C)_i)) = \begin{cases} (\text{einlagern}, C') & s(t) \in C_t \text{ und } C_{t+1} = (C_t\text{-}C') \cup s(t) \\ (\text{verwerfen}, \emptyset) & \text{sonst} \end{cases}$$

Bei der Betrachtung des Algorithmus fällt der hohe Grad der Rekursion auf. Der betrachtete Zeitpunkt erhöht sich mit jeder Rekursionsstufe, bis das Ende des simulierten Zeitraums erreicht ist. Hier löst sich die Rekursion auf. Eine graphische Darstellung dieser Rekursion ergibt einen Suchbaum, dessen Blätter die überprüften Cachezustände sind. Auf der Höhe i befinden sich also mögliche Zustände, die zum Zeitpunkt t_i auftreten können. Baumäste mit den daraufliegenden Knoten ergeben komplette Zustandsfolgen. Die Gewinne, die mit einer bestimmten Zustandsfolge erzielt werden, können als Gewicht am Ausgangsknoten der Folge abgetragen werden. Das Problem ist offensichtlich mit dem „Traveling Salesman Problem" verwand, wo es gilt, die optimale Besuchsreihenfolge für vorgegebene Orte zu bestimmen, die über verschiedene, unterschiedlich kostenverursachende Wege miteinander verbunden sind.

Die hohe Komplexität des Algorithmus hat zu der Entscheidung geführt, ihn nicht zu implementieren. Dennoch wurden schon die Programmroutinen bereitgestellt, die für jede Seite die komplette Aufzeichnung aller Referenzen und Ladezeiten übernehmen. Eine eventuelle spätere Implementierung müßte sich nur noch um den Algorithmus selbst kümmern.

Es folgt eine Betrachtung der Cachekonsistenz, bevor in Abschnitt 2.3.3 und 2.3.4 auf zwei konkrete Heuristiken eingegangen wird.

2.3.2 Strategien zur Erhaltung der Cachekonsistenz

Jeder WWW-Server bietet eine Menge von Informationen an. Teile dieser Informationen sind zum Beispiel aktuelle Berichte. Verlage bieten Ausschnitte der herausgegebenen Zeitschriften an – unter Umständen werden sie täglich aktualisiert. Das World-Wide Web bietet nur eine eingeschränkte Unterstützung, wie diese Änderungen den Clients bekannt gemacht werden können. Seiten, von denen bekannt ist, wie lange sie gültig bleiben werden, kann ein sog. Expiry-Datum mitgegeben werden (siehe [BER96]). Jeder Client weiß in diesem Fall, daß die Seite zu diesem Zeitpunkt nicht mehr aktuell sein wird. Die neueste Version der Seite kann bei Zugriffen nach dem Verfallsdatum wieder geladen und in den Cache eingelagert werden.

In vielen anderen Fällen ist im voraus nicht bekannt, ob und wann eine Aktualisierung bestimmter Seiten statt gefunden hat. Hier muß das Caching-Verfahren selbst nach einer bestimmten Strategie sicherstellen, daß Seiten aus dem Cache aktualisiert und keine veralteten Informationen daraus geliefert werden. Drei verschiedene Ansätze werden nun genauer behandelt:

Client-Polling

Diese Methode garantiert, daß zu keiner Zeit veraltete Daten aus dem Cache geladen werden können. Bei jedem Cache-Hit wird zunächst bei dem anbietenden Server die Aktualität der Seite überprüft. Ist die Seite auf dem neuesten Stand, wird die Anfrage aus dem Cache bedient, existiert eine neuere Version, wird diese sofort vom Server geladen und die alte Version der Seite ersetzt. Das HTTP-Protokoll unterstützt diese Art der Server-Anfragen durch ein bedingtes GET-Kommando. Dem GET-Request kann zusätzlich das letzte Änderungsdatum der gehaltenen Seite mitgegeben werden. Der Server überprüft nun, ob die benötigte Seite nach dem angegebenen Datum noch einmal verändert wurde. Ist das der Fall wird der Request wie eine normale Anfrage behandelt und die Seite an den Client (hier den Proxy-Server) übertragen. Im anderen Fall wird lediglich der Seiten-Header zurückgesendet, der dem Client die Aktualität der Seite bestätigt.

Als nachteilig für das Verfahren stellt sich die relativ hohe Netzbelastung heraus. Jeder Anfrage an den Cache folgt eine Rücksprache mit dem entsprechenden Server. Bei Seiten mit niedriger Änderungsfrequenz führt dies zu einer unnötig hohen Belastung des Servers und des Netzwerkes. Eine höhere Latenz auf Clientseite ist ebenfalls zu beobachten.

Time-To-Live (TTL)

Eine Linderung des Problems kann man erreichen, wenn man die Forderung nach strikter Cache-Konsistenz auflöst und einen gewissen Prozentsatz an veralteten Seiten toleriert, die aus dem Cache geladen werden. Die TTL-Strategie beruht auf dieser Idee.

Untersuchungen zum Beispiel in [GWE96] haben gezeigt, daß bestimmte Seiten die Tendenz besitzen, sich häufig zu ändern, während andere Seiten dies seltener tun. Eine Seite, die sich erst vor kurzem geändert hat, ändert sich mit hoher Wahrscheinlichkeit auch in der nächsten Zukunft wieder. Seiten, die lange unverändert geblieben sind, werden sich in Zukunft auch selten ändern. Das TTL-Verfahren ordnet jeder Seite ein Zeitfenster zu, in dem sie als aktuell angesehen wird – die Seite wird dann als „gültig" bezeichnet. Erfolgt ein Zugriff auf diese Seite innerhalb des Zeitfensters, wird ohne Rückfragen beim Server die Seite aus dem Cache geladen. Ist das Ende des Zeitfensters überschritten, wird die Seite als möglicherweise veraltet angesehen – die Seite ist „ungültig" geworden. Ein Zuriff führt zu einem bedingten Laden der Seite, wie es beim Client-Polling beschrieben wurde. Wichtig bei der Berechnung der Größe des Zeitfensters ist, daß sie auf der Zeitspanne zwischen Zugriff und letzter Änderung basiert: Die Zeitdifferenz wird mit einem Faktor (TTL-Faktor) multipliziert, das Resultat ergibt die neue Zeitfenstergröße. Auf diese Weise wird sichergestellt, daß für Seiten mit hoher Änderungsfrequenz auch häufigere Überprüfungen der Aktualität vorgenommen werden, für ältere Seiten entsprechend weniger.

Bemerkungen:

- Die Änderungshäufigkeit ist nach [CHA95] oft auch abhängig vom Typ der entsprechenden Seiten. So liegt beispielsweise die Änderungsrate von HTML-Seiten und Bildern unter dem Durchschnitt. [CHA95] schlägt hier statt einem globalen TTL-Faktor seitenspezifische TTL-Faktoren vor, die dynamisch angepaßt werden können, je nachdem wie häufig Änderungen auftreten.

- Ein TTL-Faktor von Null führt dazu, daß alle Zeitfenster die Größe Null haben. Das bedeutet, alle Seiten werden sofort nach ihrer Einlagerung wieder ungültig und müssen beim nächsten Treffer wieder auf Aktualität überprüft werden. Wir erhalten also ein Client-Polling-Verhalten.

Invalidierungsprotokolle

In solchen Fällen, wo eine schwache Cache-Konsistenz nicht toleriert werden kann und eine hohe Netzwerkbelastung vermieden werden soll, muß auf Invalidierungsprotokolle zurückgegriffen werden. Hier wird, um zu verhindern, daß ein Cache veraltete Informationen gespeichert hält, dem Server die Möglichkeit eingeräumt, Caches von einer Seitenänderung zu unterrichten. Dies ist wegen der Größe des World-Wide Web besonders problematisch. Skalierbarkeit ist z.B. eine Eigenschaft, die unbedingt bei dem ständig wachsenden World-Wide Web vorhanden sein muß. Dafür müssen zusätzliche Kosten in Kauf genommen werden: Ein hierarchisches Caching-System muß installiert werden, über das die Invalidierungsnachrichten an die untergeordneten Caches weitergegeben werden. Dazu kommt die notwendige Änderung der Cache- und Serversoftware, um die neuen Protokolle zu unterstützen. Nur weitere Eigenschaften, die im WWW wünschenswert sind (z.B. Fehlertoleranz im Falle von Serverausfällen oder Möglichkeiten der Lastbalancierung), machen eine Einführung sinnvoll. Die dennoch hohen Kosten lassen auch eine Kombination von TTL-basierter und einer invalidierungsprotokoll-basierten Konsistenzerhaltung sinnvoll erscheinen. Letztere kann dann z.B. für wichtige Datenbestände eingesetzt werden, während der kostengünstigere TTL-Ansatz für die übrigen Cachedaten eingesetzt wird.

Es folgen zwei Varianten aus der Klasse der WWW-Caching-Verfahren:

2.3.3 LRU

Das Least Recently Used-Verfahren, das wir als klassisches Verfahren bereits kennengelernt haben, wird für die Arbeit im World-Wide Web um einige Punkte ergänzt.

Besonders zu erwähnen ist die Hinzunahme einer konsistenzsichernden Strategie: In unserem Fall wird die TTL-Strategie verwendet. Sie bietet ein hohes Maß an Flexibilität und läßt sich leicht in alle klassischen Verfahren integrieren. Eine Begrenzung der erlaubten maximalen Seitengröße wurde ebenfalls eingebaut, um die zu Beginn des Abschnitts erläuterten Probleme bei der Seitenverdrängung untersuchen zu können. Auf die variable Seitengröße angepaßt ist ebenfalls die Auslagerungsprozedur. Statt einer Seite werden, falls notwenig genügend viele Seiten verdrängt.

Das angepaßte LRU-Verfahren sieht nun folgendermaßen aus:

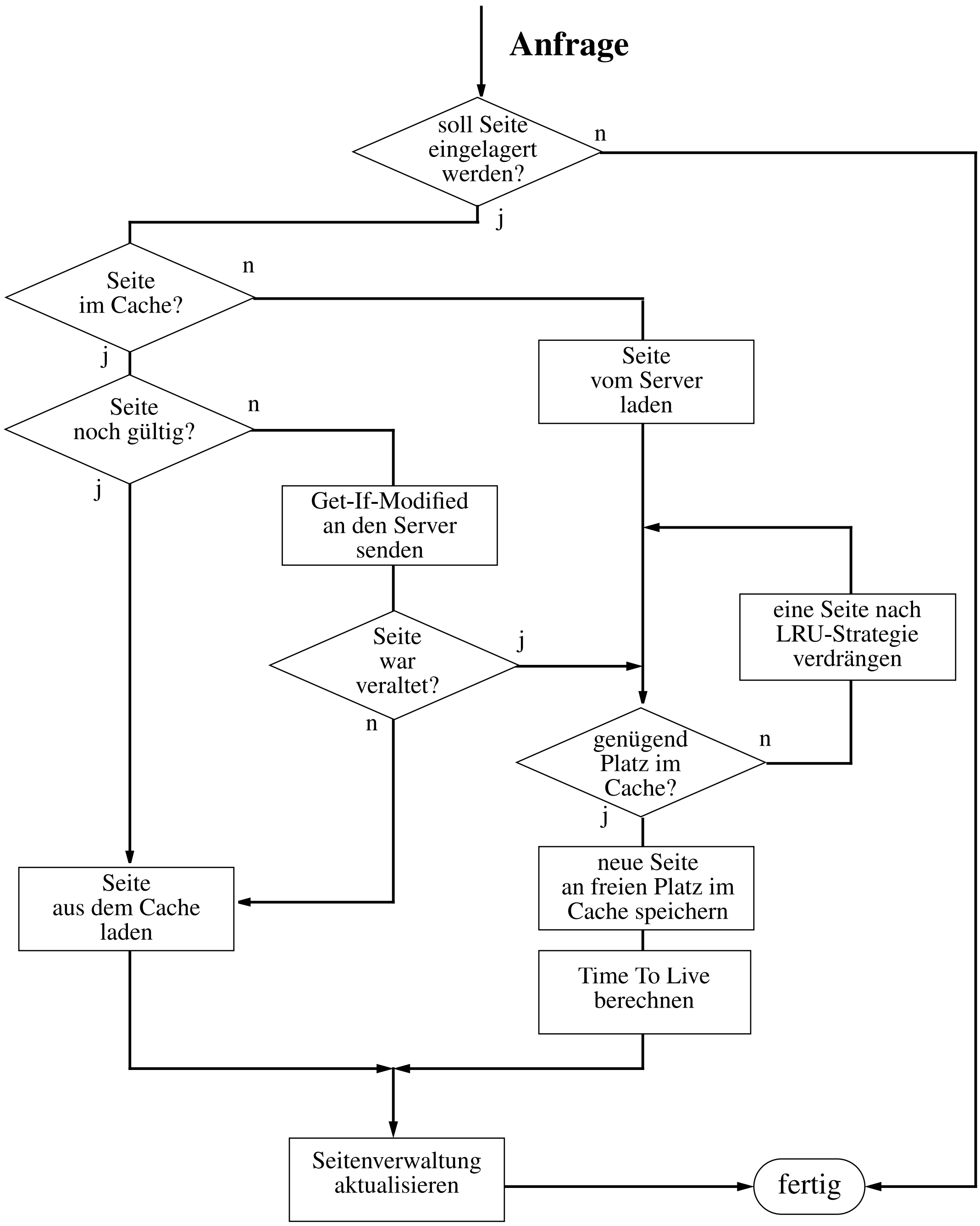


Abbildung 2-5: LRU für WWW-Seiten

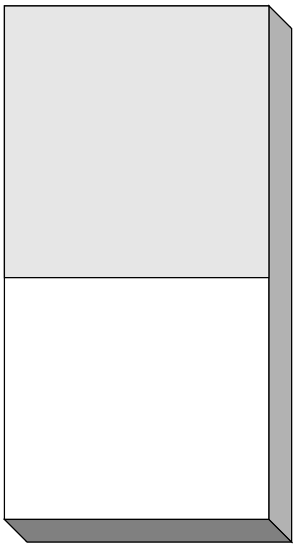
Der einleitende Test auf Einlagerung enthält nun zusätzlich eine Überprüfung der Seitengröße. Liegt sie über einem Maximalwert, findet keine Einlagerung statt. Nur Seiten, die klein genug sind, können in den Cache eingelagert werden. Ist die Seite bereits im Cache vorhanden, führt das Verfahren einen Gültigkeitstest durch. Anhand eines sog. Expiry-Datums, das bei jeder Einlagerung neu berechnet wird, kann der Ablauf der „Time To Live" erkannt werden. Die gültige Seite kann direkt aus dem Cache geladen werden, für die ungültige muß zunächst eine Überprüfung durchgeführt werden. Hat sich die Seite seit der Einlagerung geändert, liegt im Prinzip ein Cache-Miss vor. Die neue Version kann u.U größer sein als die alte, womit eine vollständige Neueinlagerung inklusive der Neuberechnung des Expiry-Datums notwendig wird. Bei der Einlagerung muß jetzt ggf. genug Platz für die Seite geschaffen werden. Erste Auslagerungskandidaten sind hier zunächst einmal alle ungültig gewordenen Seiten. Erst danach erfolgt die Verdrängung der Seite deren letzter Benutzungszeitpunkt am weitesten in der Vergangenheit lag. Der Verdrängungsvorgang wird solange wiederholt, bis genug freier Cachespeicher vorhanden ist (allgemein kann statt der LRU-Strategie auch wieder jede klassische Verdrängungsstrategie eingesetzt werden). Die Bearbeitung der Cache-Anfrage endet mit der Berechnung des Gültigkeitszeitfensters – der „Time To Live" – und der Aktualisierung der Datenstrukturen im Cache.

Der Nachteil dieser Variante ist die Nicht-Berücksichtigung des Caching-Gewinns. Seiten von weit entfernten Servern mit vergleichsweise langen Ladezeiten werden genauso behandelt, wie Seiten von nahen WWW-Servern. Bei gleichem Referenzverhalten auf zwei solcher Seiten sollte aber diejenige im Cache gehalten werden, deren Ladezeit vom Server größer ist. Der Caching-Gewinn – hier muß nicht unbedingt der Zeitvorteil durch Caching gemeint sein; geringere Netzwerkbelastung z.B. kann auch als Gewinn angesehen werden – wird im folgenden Verfahren mit berücksichtigt.

2.3.4 Least Caching Profit (LCP)

LCP steht für Least Caching Profit und soll andeuten, daß im Falle einer Seitenverdrängung zunächst die Seiten aus dem Cache entfernt werden, deren bisheriger Gewinn am niedrigsten war. Wie jede andere Heuristik kann auch das LCP-Verfahren nicht in die Zukunft schauen, es muß also davon ausgehen, daß in der Vergangenheit gemachte Gewinne auch in Zukunft auftreten werden. Dasselbe gilt für das Referenzverhalten. Jede Heuristik muß im Falle einer Referenz auf eine bestimmte Seite davon ausgehen, daß in der nächsten Zukunft wieder eine Referenz auf dieselbe Seite erfolgt und folgerichtig jede referenzierte Seite auf jeden Fall zunächst einmal in den Cache einlagern. Im Gegensatz zum Referenzverhalten bei Prozessen, ist bei den WWW-Seiten ein breiteres Lokalitätsintervall zu beobachten. Verursacht z.B. durch nicht vorhersagbare Browsing-Aktivitäten, wie sie bei einer zielgerichteten Suche auftreten können (vgl. [CAT95]), werden manche Seiten oft nur einmal innerhalb eines kürzeren Zeitraums referenziert. Um zu verhindern, daß solche Seiten andere schon bewährte (sprich, schon mehrfach referenzierte) Seiten verdrängen können, arbeitet das LCP-Verfahren mit einem Puffer, der einen Teil des Cachespeichers belegt (Abb. 2-6 verdeutlicht diese Aufteilung). In den Pufferbereich werden nur Seiten aufgenommen, die bisher einen zu geringen Gewinn erbracht haben, weil sie beispielsweise erst einmal referenziert wurden. Seiten, die so schnell von ihrem Server geladen werden können, daß kein Caching-Gewinn zu erwarten ist oder der Gewinn unter eine vorher festgelegte Schranke fällt, werden von vornherein nicht in den Puffer aufgenommen.

Im Puffer warten nun die Seiten auf eine „Bewährung" in Form einer erneuten Referenz. Ist diese Referenz aufgetreten, wird die entsprechende Seite in einen Bereich privilegierter Seiten aufgenommen, vorausgesetzt der erzielte Caching-Gewinn ist entsprechend groß. Seiten in diesem Bereich genießen einen besonderen Auslagerungsschutz – nur Seiten aus dem Puffer können aus dem Cache verdrängt werden. Eventuell werden für eine Einlagerung in den privilegierten Bereich wieder Seiten daraus verdrängt. Diese Entscheidung trifft die LCP-Strategie, die weiter unten noch präziser beschrieben wird. Die aus dem privilegierten Bereich verdrängten Seiten werden zurück in den Puffer geschrieben, wo sie erneut eine Bewährungschance erhalten, sie werden als erneut referenziert behandelt. Schafft die Seite den Sprung in den privilegierten Bereich mangels Gewinn nicht, verbleibt sie im Puffer. Möglicherweise braucht sie mehrere Zugriffe, um ausreichend Gewinn zu sammeln und dann in den privilegierten Bereich aufgenommen zu werden. Bleiben weitere Referenzen aus, wird die Seite



Pufferbereich:
Seiten mit zu geringem Gewinn oder aus privilegiertem Bereich verdrängte Seiten

- Verdrängung: LRU

Privilegierter Bereich:

- nur mehrfach referenzierte Seiten mit genügend großem Gewinn
- Verdrängung: gewinnabhängig (LCP)

Abbildung 2-6: LCP Aufteilung des Cachespeichers

wie vom LRU-Verfahren her bekannt, irgendwann aus dem Puffer und damit aus dem Cache entfernt.

Seiten im privilegierten Bereich genießen also einen besseren Auslagerungsschutz: Soll eine Seite ganz neu in den schon vollen Puffer eingelagert werden, werden nur Seiten aus dem Puffer selbst – also unbewährte oder nicht mehr bewährte Seiten – nach der LRU-Strategie verdrängt. Seiten des privilegierten Bereichs sind von der Verdrängung nicht betroffen. Die Pufferseiten selbst sind somit aus Sicht des Caching-Gewinns gleich, während im privilegierten Bereich der Caching-Gewinn eine wichtige Rolle spielt.

Bei dem Entwurf der LCP-Strategie mußten folgende Fragen geklärt werden:

- Wie gebe ich neuen Seiten die Möglichkeit Gewinne zu sammeln, ohne daß sie sofort wieder aus dem Cache verdrängt werden?
  Bei einer Lösung ohne Puffer, die Seiten gemäß ihrer Caching-Gewinne sortiert und für eine Verdrängung die Seiten mit dem niedrigsten Gewinn auswählt, füllt sich der Cache langsam mit mehrmals referenzierten Seiten. Die Menge der lediglich nur einmal referenzierten Seiten wird immer geringer. Da diese Seiten aber die ersten Verdrängungskandidaten sind (bisher konnten sie *noch* keinen Gewinn verbuchen), werden sie auch entsprechend schnell wieder aus dem Cache verdrängt. Dies führt zu einer Stagnation – neue Seiten werden sofort wieder verdrängt und eine erneute Referenz auf die Seite führt erneut zu einem Cache-Miss. Der Cacheinhalt spiegelt nicht mehr das aktuelle Referenzverhalten wieder.
  Der als Lösung eingeführte Puffer stellt sicher, daß Seiten genügend lange im Cache gehalten werden (trotz fehlendem Gewinn), bevor sie wieder daraus verdrängt werden.

- Wie verhindere ich, daß Seiten mit einmal gemachtem großen Gewinn ständig im Cache bleiben, obwohl sie in letzter Zeit nicht mehr referenziert wurden?
  Nach welcher Strategie soll eine Verdrängung von Seiten aus dem privilegierten Bereich stattfinden?

Ein einfaches Aufsummieren von Seitengewinnen ignoriert das aktuelle Referenzverhalten. Folgendes Szenario wäre möglich: Seite S hat im Monat X einen Spitzengewinn erzielt und wird zu Recht im Cache gehalten. Im Nachfolgemonat wird die Seite aber überhaupt nicht mehr referenziert. Der Topgewinn aus dem Vormonat verhindert jedoch die Einlagerung anderer Seiten. Um das Referenzverhalten mit zu integrieren, wurde ein privilegierter Bereich eingeführt. Alle Seiten innerhalb dieses Bereichs sind entsprechend ihres Gewinns sortiert. Der notwendige Kompromiß zwischen letztem Benutzungszeitpunkt und dem erzielten Gewinn der auszulagernden Seiten wird durch eine Reduzierung der Seitengewinne erreicht, die an die Referenzierungszeitpunkte jeder einzelnen Seite gekoppelt ist. Möglich wird dies durch ein Zeitfenster, das über die Seitenreferenzfolge gelegt wird. Nur innerhalb dieses Zeitfensters erzielte Gewinne werden einer Seite gutgeschrieben, weiter zurückliegende Gewinne werden ignoriert.
Wie groß muß nun dieses Zeitfenster gewählt werden? Dazu soll Abb. 2-8 zur Erläuterung dienen (die im Vergleich zum bisherigen Gebrauch leicht erweiterte

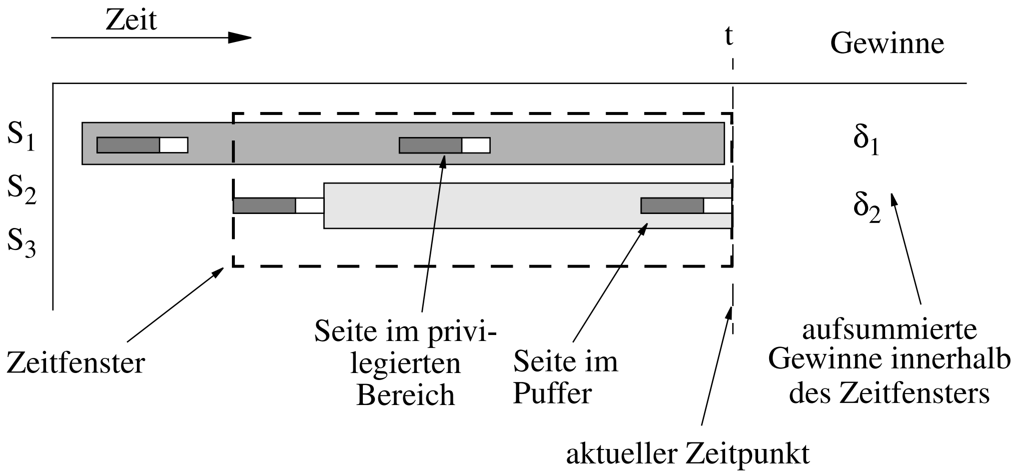


Abbildung 2-7: Notation des LCP-Beispiels

Notation ist Abb. 2-7 zu entnehmen). Das Beispiel zeigt eine Referenzfolge, in der Seite S_2 regelmäßig, Seite S_1 dagegen nur am Anfang des betrachteten Zeitraums häufiger referenziert wird. S_3 wird zu den Zeitpunkten t_1 und t_3 referenziert. Da S_1 und S_2 im privilegierten Bereich gespeichert sind, muß an den Zeitpunkten t_2 und t_4 über eine Einlagerung von S_3 und eine Verdrängung von S_1 und/oder S_2 entschieden werden. Das oben schon erwähnte Zeitfenster wird nun vom aktuellen Zeitpunkt t_1 ausgedehnt, bis zum am weitesten in der Vergangenheit liegenden letzten Referenzzeitpunkt einer Seite aus dem privilegierten Bereich oder der gerade referenzierten Seite. Im Beispiel liegt der letzte Referenzierungszeitpunkt von Seite S_1 am weitesten in der Vergangenheit und bildet somit den linken Rand des Zeitfensters. Der Gewinn von S_2 zu diesem Zeitpunkt wird nicht mehr mit aufsummiert. Für S_1 und S_3 ergibt sich somit der Gewinn δ_1 bzw. δ_3, für S_2 $2*\delta_2$. S_3 hat also noch nicht genug Gewinn erbracht, um entweder S_1 oder S_2 zu verdrängen – sie muß weiterhin im Bewährungspuffer verbleiben.

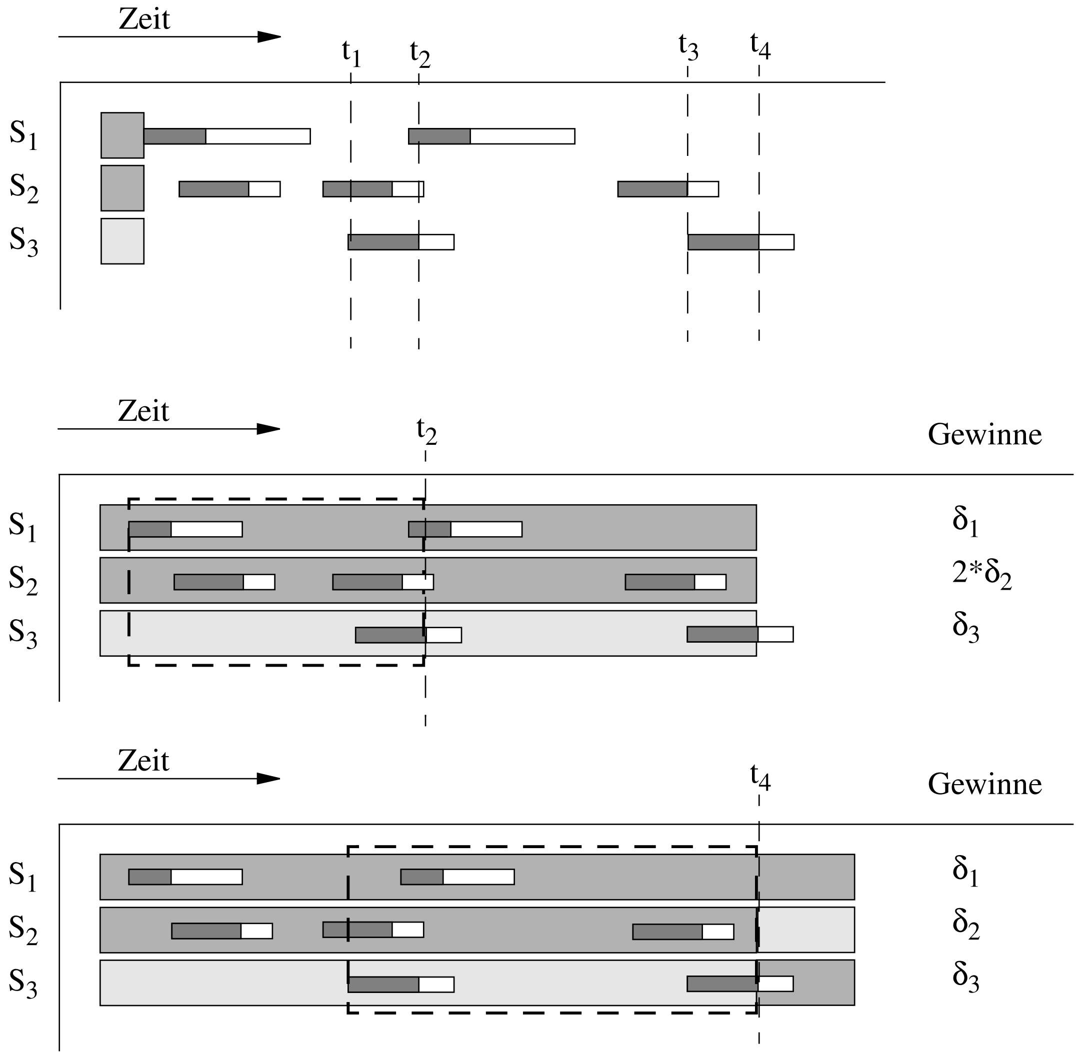


Abbildung 2-8: LCP-Verdrängung

> Die erneute Referenz auf S_3 zum Zeitpunkt t_3 führt zu einer erneuten Prüfung (zum Zeitpunkt t_4). Diesmal ist S_3 die Seite, die am weitesten in der Vergangenheit zum letzten Mal referenziert wurde. Der Gewinn von S_2 reduzierte sich durch die Vorverlegung der Zeitfenstergrenze und S_3 kann nun aufgrund des höheren Gewinns an die Stelle von S_2 treten. S_2 wird dagegen zurück in den Puffer verlegt.

Das LCP-Verfahren bietet einen Kompromiß, zwischen Berücksichtigung des Referenzverhaltens und den unterschiedlichen Gewinnpotentialen verschiedener Seiten. Auf diese Weise kann sich eine gewinnbringende Seite länger im Cache halten, als das beim LRU-Verfahren der Fall wäre. Bleiben jedoch weitere Referenzen auf die Seite aus, verliert sie ihre Berechtigung, im Cache gehalten zu werden. Seiten mit relativ geringem Gewinn, werden schneller wieder ausgelagert, häufige Referenzen auf solch eine Seite machen sie dagegen wieder zu einem guten cachewürdigen Kandidaten.

Noch eine abschließende Bemerkung: Für die Bedingungen eines klassischen Cachingverfahrens (gleiche Größe und gleiche Cachinggewinne für jede Seite) und einem Puffer der Größe Null, führt das o.g. Verdrängungsverfahren zu einer LRU-Verdrängung. Die erste Seite, deren Gewinn nach einer Zeitfensterverkleinerung auf Null abfällt wird verdrängt. Dies ist genau die Seite, die am längsten nicht mehr referenziert wurde.

2.4 Replikation

Mit dem Schritt in das World-Wide Web ergeben sich, wie wir am Anfang des Kapitels schon gesehen haben, neue Anforderungen an ein Caching-Verfahren. Insbesondere die Berücksichtigung der Netzwerkbelastung spielt eine Rolle. Aber auch Eigenschaften wie Skalierbarkeit, Fehlertoleranz, Verfügbarkeit und die Möglichkeit einer Lastbalancierung sind von Interesse. Das in [BAE96] vorgestellte CgR-Konzept macht einen Schritt in diese Richtung. CgR steht für „Caching goes Replication” und beschreibt die Idee, den normalen passiven Cache um eine aktive Komponente zu ergänzen. Der Cachespeicher wird in zwei Teile unterteilt. Während ein Teil des Speichers durch ein vergleichsweise einfaches und kostengünstiges WWW-Cachingverfahren verwaltet wird, enthält der andere replizierte Seiten, die über eine aufwendigere aktive Verwaltung aktualisiert werden (Invalidierungsprotokolle). Welche Seiten in den replizierten Bereich aufgenommen werden, kann durch Heuristiken festgelegt werden (z.B. oft benutzte Seiten oder besonders gewinnbringende Seiten) oder kann manuell bestimmt werden, falls es sich beispielsweise um besonders wichtige Daten handelt, die ständig verfügbar gehalten werden sollen. In der Regel werden ganze Namensräume gespeichert, die damit zu Kopien der Originaldaten werden. Das CgR-Konzept bezeichnet die ursprünglichen Server als „Primary Server” (PS) und Caches, die gespiegelte Daten enthalten und selbst wieder als Server auftreten können als „Replicated Server” (RS). Teile der Daten eines PS können auf diese Weise bei mehren RS

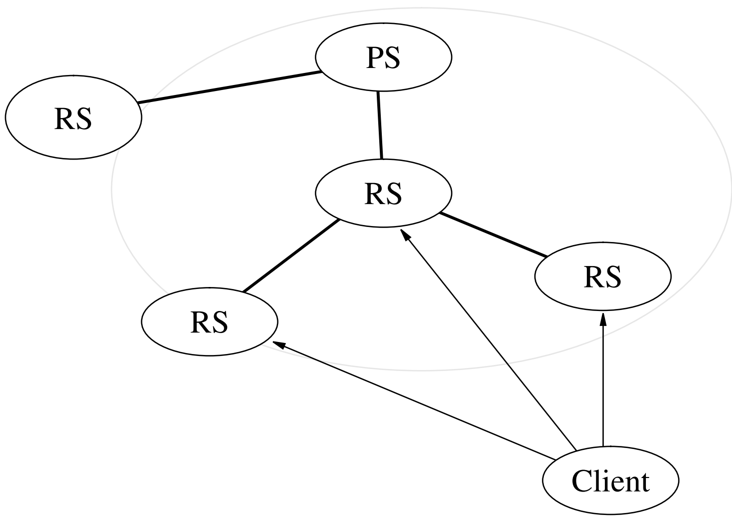


Abbildung 2-9: CgR Konstellation

gespeichert sein (Abb. 2-9). Ein Client-Proxy ist nun in der Lage, über einen speziellen Server (Web Location Information Server WLIS) den Aufenthaltsort von gesuchten Daten zu ermitteln und, falls mehrere Anbieter (PS und RS) zur Verfügung stehen, den besten – sprich, naheliegensten – Server für einen Datentransfer zu wählen. Sind die Daten beispielsweise im eigenen Cache repliziert, wird sich der Client-Proxy für ein Laden der Seite(n) daraus entscheiden.

Durch das CgR-Konzept ergeben sich somit eine Reihe von Vorteilen:

- Durch die Spiegelung von Daten auf anderen Servern (RS) wird der Primary Server entlastet, die Verfügbarkeit kann erhöht werden.

- Ein Ausfall der PS hat nicht mehr sofort den Verlust der angebotenen Daten zur Folge. Die Fehlertoleranz im WWW wird somit verbessert.
- Für die Aktualisierung der replizierten Seiten, können die weniger belasteten Nachtstunden genutzt werden. Eine gleichmäßigere Netzwerkbelastung wäre die Folge.
- Naheliegende Replicated Server entlasten den Cache. Seiten, die von solchen RS angeboten werden, brauchen u.U. nicht mehr in den Cache aufgenommen werden.
- Eventuelle Überprüfungen der Aktualität von ungültigen Seiten entfallen bei der Replikation. Insbesondere in Spitzenbelastungszeiten müssen keine „get-if-modified"-Requests mehr übertragen werden.
- Dadurch, daß die Daten näher an die Benutzer gebracht werden und die oben erwähnten „get-if-modified"-Requests vermieden werden können, verringern sich die Seitenzugriffszeiten.

Dies alles macht einen Einsatz der Replikation in einem Proxy-Cache sehr wünschenswert. Welche Gewinne durch Replikation zu erwarten sind, zeigt Kapitel 4.

2.5 Seitenübertragungszeiten

Wie schnell kann eine Seite von einem bestimmten WWW-Server übertragen werden? Wie stark hat die Tageszeit Einfluß auf diese Übertragungszeit?

Solche Fragen sind von besonderem Interesse, wenn ein Proxy-Server sich z.B. bei einer CgR-Konstellation entscheiden muß, von welchem Server (PS oder RS) eine gewünschte Seite geladen werden soll oder wenn es darum geht, Caching-Gewinne bestimmter Seiten einzuschätzen. Letzteres wird z.B. bei Caching-Verfahren benötigt, die die durchschnittliche Seitenladezeit möglichst niedrig halten sollen und denen deshalb Informationen über die Güte einer Verbindung zur Verfügung stehen müssen. Solche Informationen können in Form von Übertragungsprofilen dargestellt werden. Abb. 2-10 zeigt ein typisches Übertragungsprofil, das die Verbindungsqualität zu einem bestimmten Server beschreibt. In Abhängigkeit von Tageszeit und Seitengröße gibt es die zu erwartende Seitenübertragungszeit an. Die Abbildung zeigt die allgemeine Form eines Übertragungsprofils. Vernachlässigung der Seitengröße oder der Tageszeit ermöglichen eine einfachere und übersichtlichere zweidimensionale Darstellung, sie verbirgt aber oft interessante Details (z.B. Unterschiede zwischen einer Übertragung in den Nachtstunden und einer Übertragung am Tage).

Aufgabe eines Schätzverfahrens ist es zunächst, Übertragungsprofile auf der Basis beobachteter realer Seitenladezeiten zu erstellen. Die Zeiten können z.B. leicht während des laufenden Betriebs mitprotokolliert werden. Aus den so generierten Profilen, muß das Verfahren dann in der Lage sein, für beliebige Tageszeiten und Seitengrößen eine geschätzte Übertragungszeit anzugeben.
Die folgende Aufstellung enthält eine Reihe von Eigenschaften, die für ein Schätzverfahren von Vorteil sind:

- Schritthaltende Aktualisierung der Profile mit den Messungen
  Je mehr Meßpunkte (Übertragungszeiten) zur Verfügung stehen, desto besser kann ein genaues Übertragungsprofil berechnet werden. Ein Verfahren sollte also in der Lage sein, im laufenden Betrieb die Serverprofile zu aktualisieren. Auf der einen Seite werden so die gespeicherten Profile exakter, auf der anderen Seite ist eine

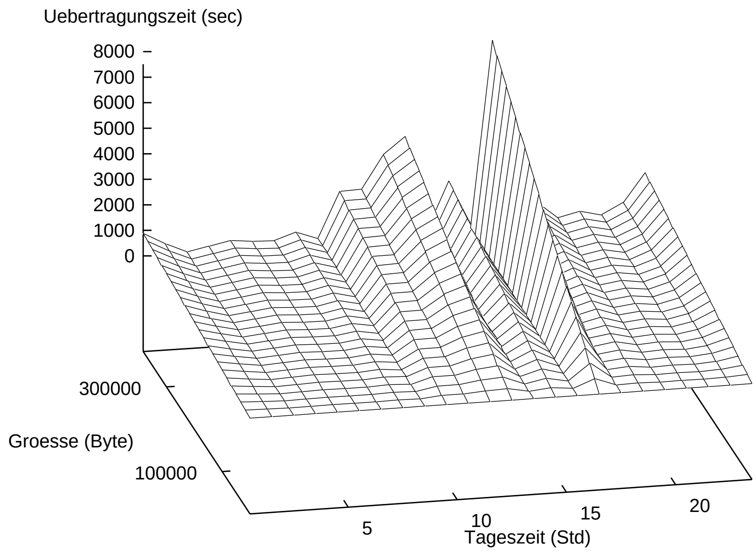


Abbildung 2-10: Beispiel eines Übertragungsprofils

Reaktion auf veränderte Verbindungsverhältnisse möglich: Durch ein Upgrade könnten sich Antwortzeiten eines bestimmten Servers z.B. verbessert haben oder wegen Überlastung könnten die Übertragungszeiten stark gestiegen sein.

- Adaptivität
  Abhängig von der Menge der zur Verfügung stehenden Meßwerte für ein bestimmten Server, sollte ein Verfahren eine feinere bzw. gröbere Tageszeitauflösung unterstützen. Für einen gering frequentierten Server würde das z.B. bedeuten, daß ein Tag in wenige, evtl. vier Abschnitte unterteilt wird (Morgen, Mittag, Abend, Nacht) und für diese Zeiträume jeweils Übertragungszeiten in Abhängigkeit von der Seitengröße berechnet wird. Bei beliebteren und häufiger frequentierten Servern ist z.B. eine Berechnung für jede Stunde des Tages denkbar. Die Unterscheidung mehrerer Tagesabschnitte ermöglicht es, unterschiedliche tageszeitabhängige Netzwerkbelastungen zu erfassen und mit in die verschiedenen Heuristiken (z.B. Seitenaktualisierung bei replizierten Seiten oder Einlagerungsentscheidungen beim Caching) einfließen zu lassen.

- Effizient berechenbar
  Sobald ein neuer Meßwert vorliegt, muß das betroffene Profil schnell aktualisiert werden können. Eine komplette Neuberechnung des Serverprofils kostet in der Regel zu viel Zeit. Außerdem müßten dazu alle bisher gemessenen Übertragungszeiten gespeichert werden, was einen beträchtlichen Speicherplatzbedarf erfordert. Dies führt uns zum letzten Punkt:

- Geringer Speicherplatzbedarf
  Da das World-Wide Web ständig wächst und immer mehr Server Informationen anbieten, ist auch die Wahrscheinlichkeit groß, daß die Menge der referenzierten Server ständig wächst. Ein möglichst geringer Bedarf an Ressourcen insbesondere für die Repräsentation der Serverprofile ist deshalb wüschenswert.

Orientiert an diesen Punkten wurden die folgenden drei Schätzmodelle entworfen. Eine quantitative Bewertung findet sich in Kapitel 4.

2.5.1 Average-Modell

Das erste Modell ist ein besonders einfaches. Abb. 2-11 zeigt, auf welche Weise ein Übertragungsprofil ermittelt wird. Das vereinfacht dargestellte Diagramm enthält die für einen bestimmten Server gemessenen Übertragungszeiten. Um Übertragungszeiten

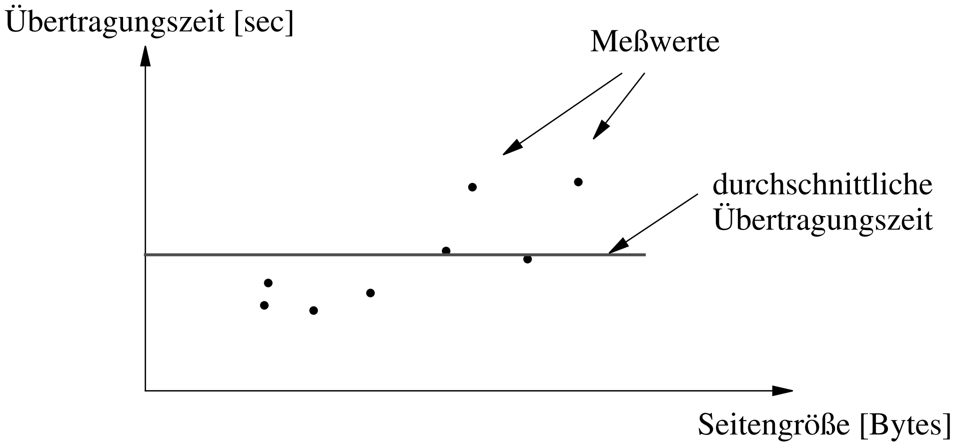


Abbildung 2-11: Average-Modell

beliebiger Seitengrößen schätzen zu können, berechnet das Average-Modell einfach die durchschnittliche Ladezeit aus den Meßwerten. Werden für einen Tag mehrere Zeitabschnitte unterschieden, ergibt sich für jeden dieser Abschnitte solch ein Diagramm – für jeden Zeitabschnitt berechnet das Verfahren dann die durchschnittliche Übertragungszeit. Bei einer Schätzung muß dem Average-Modell lediglich die Tageszeit bekannt sein, an dem die Übertragung stattfinden soll. Der berechnete Durchschnittswert des zugehörigen Zeitabschnitts ergibt die geschätzte Übertragungszeit.

Vorteile dieses Modells sind seine Einfachheit und die damit verbundene leichte Berechenbarkeit sowie der geringe Speicherplatzbedarf der Übertragungsprofile. Übertragungszeiten verschiedener Server können leicht miteinander verglichen werden – um sich zwischen zwei Servern zu entscheiden, müssen lediglich die Durchschnittswerte verglichen werden.

Nachteilig ist die fehlende Berücksichtigung der Seitengröße. Dies führt dazu, daß die Zeiten für kleine Seitengrößen überschätzt, d.h. zu hoch, für große Seiten aber zu niedrig angegeben werden. Diesem Mangel kann durch eine Ersetzung des Durchschnittswertes durch eine einfache Gerade begegnet werden. Das nächste Modell arbeitet mit dieser Methode:

2.5.2 StraightLine-Modell

Um dem Umstand Rechnung zu tragen, daß besonders für große WWW-Seiten die Unterschiede zwischen nahen und weiter entfernten Servern deutlich spürbar werden, verwendet das StraightLine-Modell eine Gerade, um Übertragungszeiten zu schätzen. Abb. 2-12 illustriert dies. Die Steigung und der y-Achsenabschnitt der Geraden, werden dabei so berechnet, daß die Summe aller Punktabstände zu der Geraden minimal wird. Als Abstandsmaß zwischen einem Punkt (x_i,y_i) und der Gerade $g(x) = mx+b$ wird das Quadrat der Differenz zwischen y_i und $g(x_i)$ verwendet. Dabei bezeichnet (x_i,y_i) den i-ten Meßpunkt.

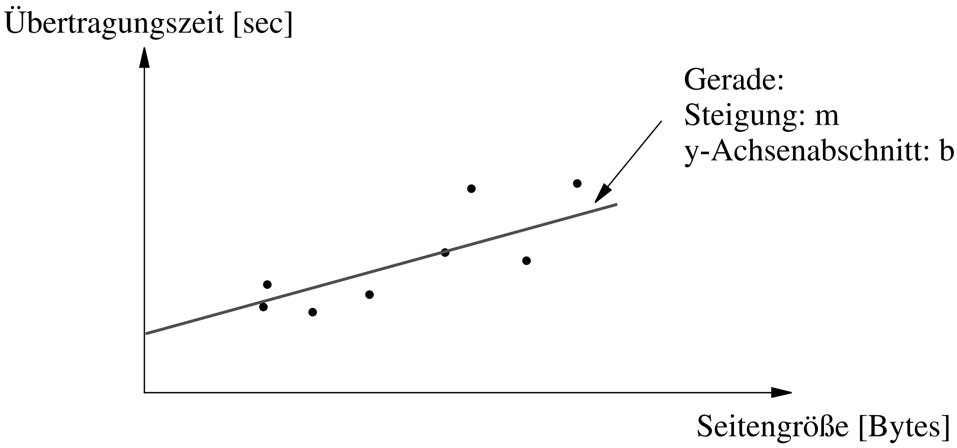


Abbildung 2-12: StraightLine-Modell

Für n Meßpunkte ergibt sich der Gesamtabstand der Punkte zu der Geraden dann folgendermaßen:

$$s(m, b) = \sum_{i=1}^{n} (y_i - (mx_i + b))^2 \tag{1}$$

Für eine Minimierung wird s(m,b) jeweils nach „m" bzw. „b" abgeleitet:

$$\frac{\partial}{\partial m} s = \sum_{i=1}^{n} 2mx_i^2 - 2x_i y_i + 2bx_i \tag{2}$$

$$\frac{\partial}{\partial b} s = \sum_{i=1}^{n} 2mx_i - 2y_i + 2b \tag{3}$$

Zu-Null-setzen der Gleichung (3) und Auflösen nach „b" ergibt:

$$b = -\frac{1}{n}\left(\sum_{i=1}^{n} y_i + m \sum_{i=1}^{n} x_i\right) \tag{4}$$

Einsetzen von (4) in (2), zu-Null-setzen des Resultats und Auflösen nach „m” liefert schließlich:

$$m = \frac{\sum_{i=1}^{n} x_i \left(y_i - \frac{1}{n} \sum_{j=1}^{n} y_j \right)}{\sum_{i=1}^{n} x_i \left(x_i - \frac{1}{n} \sum_{j=1}^{n} y_j \right)} \tag{5}$$

Für die Parameter „b” und „m” ergeben sich somit zwei geschlossene Formeln, die mit Hilfe der Meßpunkte berechnet werden können.

Das Ergebnis macht eine genauere Schätzung der Übertragungszeit möglich. Die Genauigkeit wurde jedoch durch einen höheren Speicherplatzbedarf und größerem Berechnungsaufwand erkauft. Für eine Rekalkulation eines Profils müssen alle vorangegangenen Meßpunkte weiterhin zur Verfügung stehen. Wie auch schon im einführenden Text erwähnt wurde, ist diese Eigenschaft für eine während des normalen Betriebs stattfindenden Aktualisierung nicht akzeptabel. Abhilfe schafft eine einfachere Berechnung der Interpolationsgeraden:

2.5.3 Simple-StraightLine-Modell

Der hohe Berechnungsaufwand des StraightLine-Modells erzwingt einfachere Methoden, um die Interpolationsgrade zu ermitteln. An die Stelle einer optimalen Geraden durch die Punktmenge mit minimalem Gesamtabstand zu den Meßpunkten, tritt beim Simple-StraightLine-Modell eine Näherung. Dazu werden die Übertragungszeiten für die minimale und die maximale Seitengröße betrachtet. Der Meßpunkt, an dem die kleinste Seite am schnellsten übertragen wurde, bildet den ersten Punkt auf der Geraden (vgl. Punkt (1) in Abb. 2-13). Der zweite Punkt (Punkt (2) in der Abbildung), bei

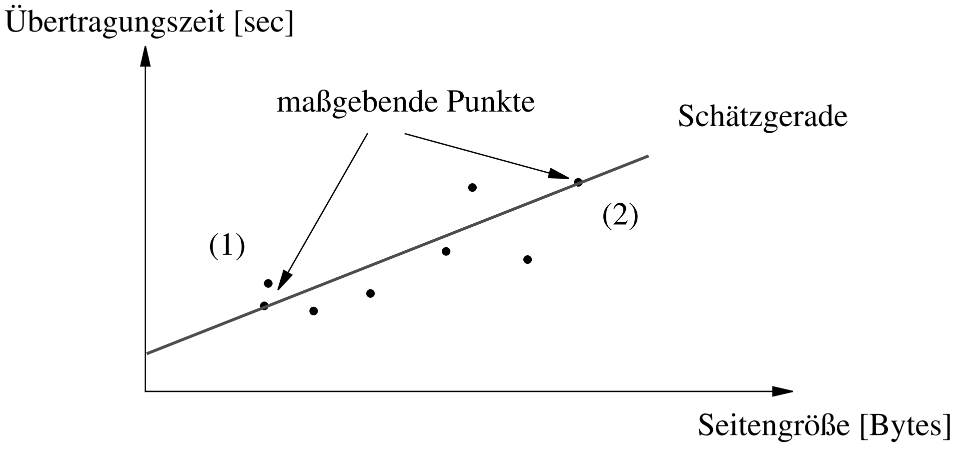


Abbildung 2-13: Simple-StraightLine-Modell

dem die größte Seite am langsamsten übertragen wurde, legt die Gerade vollständig fest. Alle weiteren Meßpunkte haben keinen weiteren Einfluß auf die Gerade.

Die Berechnung der Interpolationsgeraden fällt deutlich einfacher aus als im StraightLine-Modell. Alte Meßwerte müssen nicht gespeichert werden, lediglich die aktuellen die Gerade festlegenden Punkte müssen verfügbar sein. Zu beachten ist, daß neue Meßpunkte die Schätzgerade nur noch steiler machen können. Damit kann das Modell nur auf eine Verschlechterung der Verbindungssituation reagieren, nicht aber auf einer Verbesserung. Um letzteres erreichen zu können, müssen die geradenbestimmenden Meßpunkte in regelmäßigen Zeitabständen verworfen und mit Hilfe aktueller Meßdaten neu bestimmt werden.

Der Vergleich zweier Übertragungsprofile ist bei den beiden zuletzt vorgestellten Modellen nicht mehr ganz so leicht, wie das beim Average-Modell der Fall war. Vorstellbar sind Situationen, in denen von Server A kleine Seiten schneller übertagen werden können als von Server B, für große Seiten aber genau das Gegenteil gilt. Für eine Bewertung von zwei Verbindungen muß also in der Regel auch die ungefähre Größe der zu übertragenden Seite bekannt sein.

StraightLine und Simple-StraightLine-Modell idealisieren das Übertragungsverhalten, indem sie mit einer linearen Zunahme der Übertragungszeit bei wachsender Seitengröße arbeiten. Dadurch wird ihre Empfindlichkeit gegenüber Schwankungen der Meßwerte größer. Insbesondere bei wenigen Meßpunkten, die für ungefähr gleich große Seiten ermittelt wurden, kann eine besonders steile Schätzgerade die Folge sein. Sind beispielsweise nur Übertragungszeiten für ein und dieselbe Seitengröße gemessen worden, wird daraus eine Schätzgerade mit unendlich großer Steigung errechnet. Hier müssen also zusätzliche Sicherheitsvorkehrungen getroffen werden: die in die Berechnung eingehenden Meßwerte dürfen nicht alle in einem Mindestintervall liegen. Tun sie es trotzdem, kann z.B. auf eine Durchschnittsberechnung wie sie beim Average-Modell verwendet wird, zurückgegriffen werden (Steigung $m = 0$) oder es kann mit einer Defaultsteigung der Schätzgeraden gearbeitet werden. In diesem Fall wird der y-Achsenabschnitt so gewählt, daß die Gerade möglichst genau durch die Punktmenge verläuft.

3. Das Simulationssystem

Dieses Kapitel wendet sich der Beschreibung des Simulationssystems zu. Beginnend mit den Zielsetzungen für CaSimiR und den Entwurfsentscheidungen, die für das Simulationssystem getroffen wurden, wird anschließend auf den grundsätzlichen Ablauf einer Simulation eingegangen. Dafür notwendige Ein- und Ausgaben des Systems werden spezifiziert und die vom System bereitgestellte Funktionalität näher erläutert. Dieser Beschreibung folgt eine Vorstellung des Softwaresystems mit Systemaufbau, internem Datenfluß und einer Beschreibung ausgewählter Komponenten.

3.1 Zielsetzung und Entwurfsentscheidungen

Der Cachesimulator CaSimiR soll drei wesentliche Aufgaben übernehmen:

- Unterstützung bei der Auswertung von Logfiles
- Auswertung verschiedener Caching-Heuristiken bei Simulation eines realistischen Anfrageverhaltens
- Integration und Evaluierung von Replikationsstrategien ermöglichen

Der erste Punkt umfaßt die Aufbereitung der in Form von Logfiles vorliegenden Zugriffssequenzen. Die gewünschten Daten, z.B. die zur Verfügung stehende Bandbreite oder das aufgetretene Referenzverhalten, muß aus den Logfile-Einträgen extrahiert und für eine weitere Verarbeitung ausgegeben werden. Punkt 2 der Liste bedingt, daß das Design des Softwaresystems eine entsprechende Austauschbarkeit der verschiedenen Caching-Verfahren garantiert, so daß für alle Verfahren unter identischen Voraussetzungen und Gesichtspunkten untersucht werden können. Punkt drei schließlich erfordert die Bereitstellung entsprechender Infrastruktur für die Verwaltung replizierter Seiten. Neue Replikationsstrategien können dann auf diese Infrastruktur zurückgreifen und sich somit auf die Verwaltung eigener Daten beschränken. Die getroffenen Entwurfsentscheidungen sind im Einzelnen noch einmal genauer erläutert:

Entwurfsentscheidungen:

- Aufgrund der Fülle der einstellbaren Parameter ist die Kontrolle über sie durch ein spezielles Konfigurationsfile realisiert, in dem alle relevanten Einstellungen vorgenommen werden. Auf diese Weise wird automatisch die jeweilige Simulationsumgebung mit den verwendeten Einstellungen festgehalten. Bei einer Kontrolle über Kommandozeilenparameter wäre dies nicht mehr notwendigerweise der Fall.
- Der Simulator bietet zwei verschiedene Arbeitsmodi an: der erste ist speziell für die Analyse der Logfiles konzipiert und bietet entsprechende Analysemöglichkeiten. Der zweite Modus ist ausschließlich für die Auswertung der Caching-Heuristiken gedacht. Hier wird insbesondere die zügige Durchführung von kompletten Meßreihen möglich.

- Um die Simulationsläufe bei den Cachetests zu beschleunigen, werden eigene vorverarbeitete Logfiles verwendet, die nur die notwendigen Informationen für die Simulation enthalten.
- Robustheit gegen Unterbrechung der Meßreihen wird durch einen Mechanismus erreicht, der es ermöglicht, bei nicht vollständig durchgeführten Meßreihen an der zuletzt vor dem Abbruch bearbeiteten Konfiguration aufzusetzten und an dieser Stelle fortzusetzen.
- Die Generierung von Auswertungsdaten und die Visualisierung derselben wird voneinander getrennt. Dieser Ansatz garantiert eine optimale Flexibilität in Bezug auf die Datenauswertung. Andere möglicherweise leistungsfähigere Visualisierungswerkzeuge können Verwendung finden. Die Daten sind auf diese Weise ein guter Ausgangspunkt für weiterführende Analysen.
- Für die Visualisierung der Meßdaten wird das GNU-Tool GNUPLOT [WIL93] eingesetzt. Dieses Werkzeug ist frei verfügbar und bietet eine umfangreiche und einfach zu nutzende Funktionalität zur graphischen Darstellung von Funktionen und Datenfiles.
- Eine Integration von Managementfunktionalität für replizierte Seiten in das Simulationssystem ermöglicht eine leichte Einbindung von Replikationsstrategien. Diese Einbindung ist in Bezug auf die Datenauswertung transparent, d.h. Daten die vor der Einbindung einer neuen Strategie erhoben werden konnten, können ohne weitere Manipulation auch für diese neue Strategie erhoben werden (lediglich die Erfassung beispielsweise neuer Bewertungsmaße erfordert eine Manipulation der entsprechenden Module).
- Für die Implementierung des Simulationssystems wurde die Programmiersprache C++ gewählt.

3.2 Simulationsablauf

Um einen Überblick über die Funktionsweise des Simulators zu geben, stellt dieses Kapitel zunächst die zwei verschiedenen zur Verfügung stehenden Betriebsmodi vor. Der darauffolgende Abschnitt wendet sich dann speziell den Eingabefiles zu und beschreibt deren Inhalt und Funktion.

Logfile-Auswertung
Im Auswertungsmodus für die Logfiles können grundlegende Informationen aus den vorliegenden Logfiles gewonnen werden. Die Logfiles enthalten für jeden Zugriff auf den Proxy-Server einen Eintrag. In diesem Eintrag ist u.a. angegeben, welche Seite angefordert wurde, wie lange die Übertragung der Seite dauerte und ob die Seite aus dem Proxy-Cache geliefert werden konnte oder nicht (genaueres dazu in Abschnitt 3.3). Aus den Logfiles sind also auch Hinweise auf die Leistung des aktuell eingesetzten Proxy-Caches zu entnehmen. Abb. 3-1 zeigt den Ablauf eines Auswertungslaufes. Zunächst liest CaSimiR das Konfigurationsfile. Hier sind u.a. die Namen der zu analysierenden ggf. komprimierten Logfiles angegeben. Außerdem entnimmt der Simulator aus dem Konfigurationsfile, welche Informationen aus den Logfiles gewonnen und welche anderen zusätzlichen Eingabefiles gelesen werden sollen (Übertragungsprofile, Domänenliste). Mit Hilfe der Domänenliste ist es möglich, Server entsprechend ihrer Domänenzugehörigkeit zu einem logischen Server zusammenzufassen. Dadurch wird es möglich, z.B. das Zugriffsverhalten auf die Gesamtheit aller Server innerhalb dieser Domäne zu betrachten (für Informationen zur Servergruppierung, siehe Abschnitt

3.3.2). Nachdem die Konfiguration abgeschlossen ist, werden die Logfiles nun Zeile für Zeile abgearbeitet und die gewünschten Daten erhoben. Am Ende der Analyse werden die entsprechenden Datenfiles generiert. Diese Datenfiles sind einfache ASCII-

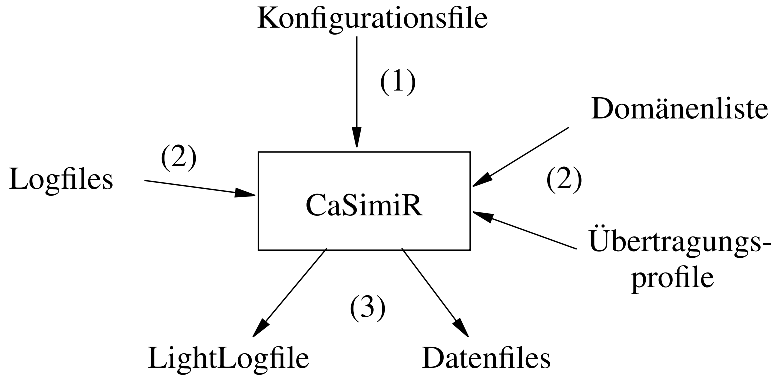


Abbildung 3-1: Datenfluß bei Auswertung der Logfiles

Dateien, welche Zahlenkolumnen enthalten, aus denen GNUPLOT schließlich entsprechende Diagramme generieren kann (Näheres zu den Datenfiles ist dem Abschnitt 3.5 zu entnehmen). Die Generierung der Datenfiles kann in den meisten Fällen noch vom Konfigurationsfile aus beeinflußt werden. Oft ist es in bestimmten Fällen nützlich, Daten mit variierendem Auflösungsgrad erzeugen, um beispielsweise die Auswirkung von zufälligen Schwankungen in den Daten zu mildern.

Über den Logfile-Auswertungsmodus können zusätzlich die sog. Light-Logfiles erzeugt werden. Sie stellen ein Kondensat der originalen Logfiles dar und sind so vorverarbeitet, daß zum einen nur relevante Information für die Simulation darin enthalten sind und zum anderen der Aufwand beim Einlesen der Files minimal ist – insbesondere für das Parsen der einzelnen Einträge.

Cache-Auswertung
Ähnlich zu der Lofile-Auswertung arbeitet der Modus für die Cacheanalyse (siehe Abb. 3-2). Dieser Modus ist jedoch speziell für die Auswertung verschiedener Caching-Heuristiken konzipiert worden. Der erste Unterschied ist, daß die Simulation nur mit vorverarbeiteten Light-Logfiles betrieben werden kann. Sie werden ganz normal im Konfigurationsfile eingetragen. Zusätzlich dazu wird ein Parameterfile mit den zu variierenden Cacheparametern mit angegeben. Für jeden durchzuführenden Simulationslauf enthält dieses File die zu wählenden Einstellungen (z.B. Cachegröße, Mindestseitengröße, TTL-Faktor, etc.).

Optional ist es vor jedem Simulationslauf möglich, den zu simulierenden Cache mit Seiten zu füllen *bevor* die eigentliche Bewertung der Heuristik beginnt. Damit wird auf die Tatsache Rücksicht genommen, daß Daten über real eingesetzte Caches nur aus dem laufenden Betrieb heraus gewonnen werden, d.h mit bereits gespeicherten Seiten. Nur so wird ein Vergleich von neuen Strategien mit bereits vorhanden möglich.
Ist der Cache nun entsprechend mit Seiten gefüllt, kann der eigentliche Simulationsdurchlauf beginnen, bei dem eine Reihe von Eigenschaften wie z.B. durchschnittliche Seitenübertragungszeit, Cache-Hit-Rate oder Byte-Hit-Rate gemessen und in das angegebene File gespeichert wird. Details zu den Messungen sind in Abschnitt 3.5 zu finden, der genaue Simulationsablauf ist in Abschnitt 3.6.3 erläutert.

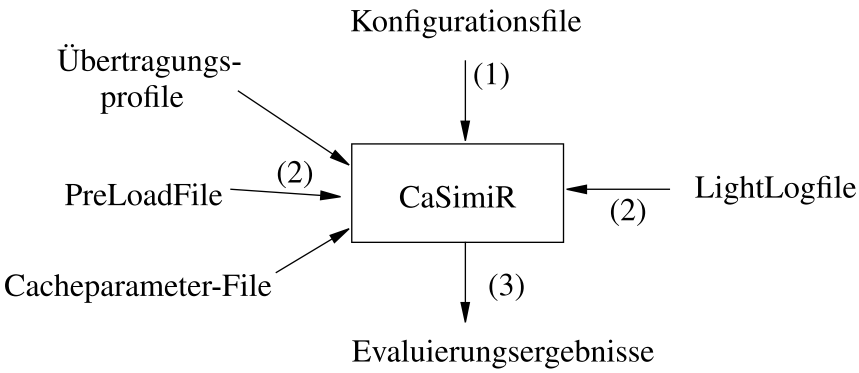


Abbildung 3-2: Datenfluß bei Auswertung der Cache-Heuristiken

3.3 Spezifikation der Eingaben

Dieser Abschnitt enthält eine Auflistung aller Eingaben für das Simulatorsystem CaSimiR. Zu einer Erläuterung wird außerdem eine Strukturbeschreibung in der erweiterten Backus-Naur-Form (EBNF) angegeben. Dafür wird folgende Notation verwendet:

- Nichtterminalsymbole sind in „<” und „>” eingeschlossen.
- Im Eingabetext vorkommende Zeichenfolgen sind in Anführungszeichen gesetzt.
- { a }: das Symbol „a” kann beliebig oft wiederholt werden (auch keinmal).
- [a]: das Symbol „a” kommt höchstens einmal im Eingabetext vor.
- a | b: entweder das Symbol „a” oder das Symbol „b” kommt im Eingabetext vor.
- Ausdrücke können mit Klammern ’(’ und ’)’ zusammengefaßt werden.
- Das Nichtterminalsymbol „CR” steht für ein Carriage-Return.

3.3.1 Logfiles

Wie weiter oben schon angedeutet, enthalten die Logfiles Informationen über den Zugriff auf den Proxy-Server der Universität Kaiserslautern. Für jeden Zugriff, der aus der Universität heraus an einen außerhalb gelegenen WWW-Server getätigt wurde und stellvertretend vom Proxy-Server weitergeleitet wurde, existiert ein Eintrag in den Logfiles. Die Logfiles sind dabei in drei verschiedene Typen aufgeteilt: Proxy-Logs, Cache-Logs und LastModified-Logs. In den Proxy-Logs sind alle Zugriffe vermerkt, die tatsächlich an die jeweiligen außenstehenden Server abgesetzt wurden. Darunter fallen sowohl normale Ladezugriffe (vgl. GET-Kommando des HTTP-Protokolls) als auch bedingte Ladekommandos (Get-If-Modified). Zugriffe, bei denen die angeforderte Seite aus dem Proxy-Cache entnommen werden konnte, werden in den Cache-Logs gespeichert. Auf diese Weise läßt sich die Leistung des zu diesem Zeitpunkt eingesetzten Caches ermitteln und macht einen Vergleich mit anderen Caching-Verfahren möglich. Eine Zeile aus den Logfiles könnte folgendermaßen aussehen:

```
xxx.rhrk.uni-kl.de - - [02/Jan/1996:06:32:44 +0100]
        “GET http://www.dpunkt.de/index.html HTTP/1.0” 200 3921
        +1:234502 (Mozilla/2.0b4 (Win95; I)
```

Zunächst wird der Rechnername des Clients vermerkt, von dem die Anfrage an den Proxy-Server ausging. Es folgt die Angabe des Zeitpunkts, an dem der Transfer beendet wurde (lokale Zeit). Der Zeitzonenoffset hinter der Datumsangabe beschreibt die Zeitdifferenz zwischen der lokalen Zeit und der sog. „Greenwhich Mean Time" (GMT). Die ersten zwei Ziffern geben die Stunden, die letzten zwei die Minuten der Zeitverschiebung an. Dann folgt der HTTP-Request, und die Zeile wird fortgesetzt mit dem Server-Reply-Code und der Größe der Seite, gefolgt von der Dauer des Zugriffs und den Angaben über verwendete Client-Software bzw. eventuell benutzter Gateways. Die letztgenannten Angaben werden von dem Simulator selbst nicht benötigt.

EBNF-Darstellung der Proxy- und Cache-Logs

```
<Logfile>              :=  { <Eintrag> }
<Eintrag>              :=  <ClientRechner> " - - " <Transferende>
                           <HTTPRequest> <ServerReply>
                           <Seitengröße> <Übertragungszeit>
                           <ClientSW+GateWays> CR
<TransferEnde>         :=  '[' <Datum> ':' <Uhrzeit>
                           <Zeitzonenoffset> "]"
<Datum>                :=  <Tag> '/' <Monat> '/' <Jahr>
<Uhrzeit>              :=  <Std> ':' <Min> ':' <Sek>
<ZeitzonenOffset>      :=  ( '+' | '-' ) <Std> <Min>
<Std>                  :=  <Ziffer> <Ziffer>
<Min>                  :=  <Ziffer> <Ziffer>
<Sek>                  :=  <Ziffer> <Ziffer>
<HTTPRequest>          :=  '"' Request '"'
<ServerReply>          :=  <Zahl>
<Seitengröße>          :=  <Zahl>
<Übertragungszeit>     :=  '+' <Zeitdauer>
<ClientSW+GateWays>    :=  '(' Angaben ')'
<Zeitdauer>            :=  <Zahl> ':' <Zahl>
<Zahl>                 :=  <Ziffer> { <Ziffer> }
<Ziffer>               :=  '0' | ... | '9'
```

Der dritte Logfiletyp enthält detailliertere Informationen zu den Abläufen im Cache selbst. Insbesondere Informationen über das LastModified-Datum der im Cache gelagerten Seiten werden hier gespeichert. Für jeden Cachezugriff wird u.U. eine Folge von Aktionen dokumentiert, die der Cache bei einer Anfrage durchführte – zu einem Zugriff gehören nun u.U. mehrere Zeilen. Festgehalten wird, für welche Seiten-URL ein Test auf Vorhandensein im Cache durchgeführt wurde, welches LastModified-Datum die Seite hatte – soweit sie im Cache gespeichert war – und ob die Aktualität der Seite aufgrund des LastModified-Datums bei dem entsprechenden WWW-Server zunächst überprüft werden muß. Je nach der dokumentierten Aktion sind im LastModified-Logfile nun Zeilen verschiedenen Typs gespeichert. Die folgenden Aktionen führen im LastModified-Logfile zu den beispielhaft dargestellten Aufzeichnungen:

- Auf Vorhandensein der Seite im Cache testen. Result = 1 deutet an, daß die Seite im Cache gespeichert war, das entsprechende LastModified-Datum kann deshalb angegeben werden. Ein Resultwert von Null dokumentiert, daß die Seite nicht im Cache gespeichert war. In diesem Fall ist kein LastModfied-Datum verfügbar.
```
Cache-test for URL "http://www.nba.com/images/swrcolor.gif":
          LM = 831681691 / Now = 833525457. Result = 1
```
- Die Seite wurde neu in den Cache eingelagert.
```
Cache: Created entry for URL 'http://www.focus.de/E/EC/ECD/kecd01.gif'
          (LM = 823213044)
```
- Die Seite konnte aus technischen Gründen nicht aus dem Cache entnommen werden.
```
Cache....... file is locked:
          "/tools/WWW.cache/http/207.30.50.110/crazed06.zip"
```
- Bestimmte Seiten werden prinzipiell nicht im Cache gehalten. Das sind die Seiten, die innerhalb der Universität Kaiserslautern angeboten werden.
```
Cache-miss(no caching) for URL "http://www.uni-kl.de/"
```
- Cache-Miss ohne Einlagerung der Seite.
```
Cache-miss for URL "http://x500.belwue.de:8888/Mou%3DFakultaet"
```
- Laden einer Seite aus dem Cache
```
Cache-Hit: http://www.ironman.de/gif/butbar2.gif not expired
          (LM = 831822911)
```
- Get-If-Modified-Kommando an den entsprechenden Server senden.
```
Cache-IMS: http://www.lycos.com/assist/banners/result-bar.gif
          (LM = 833314427)
```
- Antwort eines Servers auf ein Get-If-Modified-Kommando entgegennehmen.
```
Cache: Reply: 304 (1) for
          Url "http://www.lycos.com/assist/banners/visit2.gif"
          (LM = 833314427)
```
  Als Reply-Code kommen die Antworten 200 (Seite wurde neu geladen) und 304 (Seite war noch aktuell) in Frage. Die Zahl in Klammern dahinter gibt an, warum das Get-If-Modified-Kommando abgesendet wurde: (1): Clientseitiger Reload Button wurde gedrückt, (2): Seite war laut Caching-Algorithmus ungültig oder (3): Seite sollte nach dem Expires-Algorithmus erneuert werden.

Ein Cachezugriff kann bis zu drei Einträge im LastModified-Logfile erzeugen, wie das folgende Beispiel zeigt:

```
Cache-test for URL "http://www.lycos.com/ads/telebase6.gif":
          LM = 833484946 / Now = 833525696. Result = 1
Cache-IMS: http://www.lycos.com/ads/telebase6.gif (LM = 833484946)
Cache: Reply: 304 (1) for Url "http://www.lycos.com/ads/telebase6.gif"
          (LM = 833484946)
```

Zunächst erfolgt ein Test, ob die Seite im Cache gespeichert war. Der an den Server abgesetzte Get-If-Modified-Request in Zeile 2 wird mit einem „not modified” (304) beantwortet und die Seiten aus dem Cache geladen (Zeile 3).

EBNF-Darstellung des LastModified-Logsfiles

```
<LMLogfile>           :=  <Eintrag> { <Eintrag> }
<Eintag>              :=  <CacheTestEintrag> | <CreatedEintrag> |
                          <CacheLockedEintrag> |
                          <NoCachingEintrag> | <CacheMissEintrag>
                          | <CacheHitEintrag> | <IMSEintrag> |
                          <ReplyEintrag>
<CacheTestEintrag>    :=  ‘Cache-test for URL “‘ URL-Angabe ‘”:
                          LM = ‘ <LastModifiedDatum> ‘ / Now =‘
                          <Zugriffszeit> ‘ Result =’
                          <TestErgebnis> CR
<CreatedEintrag>      :=  “Cache: Created entry for URL ’” URL-
                          Angabe “’ (LM=“ <LastModifiedDatum “)”
                          CR
<CacheLockedEintrag>  :=  “Cache....... file ist locked: “
                          Fileangabe CR
<NoCachingEintrag>    :=  ’Cache-miss(no caching) for URL ”’ URL-
                          Angabe ’”’
<CacheMissEintrag>    :=  “Cache-miss for URL “ URL-Angabe CR
<CacheHitEintrag>     :=  “Cache-Hit: “ URL-Angabe “(LM = “
                          <LastModifiedDatum> “)” CR
<IMSEintrag>          :=  “Cache-IMS: “ URL-Angabe “ (LM = “
                          <LastModifiedDatum> “ )” CR
<ReplyEintrag>        :=  ’Cache: Reply: ’ <ReplyCode> ’(’
                          <IMSInfoCode> ’) for Url “’ URL-Angabe
                          ’”’ CR
<TestErgebnis>        :=  “0” | “1”
<ReplyCode>           :=  “200” | “304”
<IMSInfoCode>         :=  “0” | .. | “3”
<LastModifiedDatum>   :=  <Zahl>
<Zugriffszeit>        :=  <Zahl>
<Zahl>                :=  <Ziffer> { <Ziffer> }
<Ziffer>              :=  ’0’ | ... | ’9’
```

Ein Logfile enthält jeweils die Informationen für den Zeitraum eines Monats. Insgesamt existieren also drei Logfiles für jeden Monat. Die Einträge innerhalb eines Logfiles sind (weitgehend) nach dem Zeitpunkt sortiert, an dem der jeweilige Zugriff abgeschlossen wurde.

Ein vierter von CaSimiR selbst generierter Logfiletyp wird aus einem Kondensat der in den oben genannten drei Logfiletypen enthaltenen Informationen als Eingabe für den Simulator verwendet. Da dieser Logfiletyp nur die für die Cachesimulation notwendige Information enthält, die zudem noch so aufbereitet sind, daß eine möglichst schnelle Verarbeitung ohne umfangreiche Parse-Aktivität stattfinden kann, werden Logfiles von diesem Typ auch als „Light-Logfiles” bezeichnet. Folgende Vorverarbeitungsschritte werden für die Erzeugung durchgeführt:

- Informationen über Clientrechner und -software sowie über die verwendeten Gateways bei der Übertragung sind gestrichen.

- Datumsangaben sind in GMT konvertiert und werden im Universal Time Count-Format (UTC) abgespeichert.[1]
- Die HTTP-Requests werden normalisiert. Darunter fällt die einheitliche Verwendung von Kleinbuchstaben, Elimination der Defaultportnummer 80, Abtrennen eventueller Argumente (lediglich ein Flag, ob Argumente vorhanden waren oder nicht, wird gespeichert) und die Umwandlung von mit %-Zeichen kodierten Sonderzeichen. Sonderzeichen werden nur umgewandelt, falls ihr ASCII-Code größer als 32 ist. Damit wird verhindert, daß Leerzeichen und Kontrollcodes wie Carriage-Return in den Logfiles auftreten.
- Statt dem Transferende-Zeitpunkt wird der Zeitpunkt gespeichert, an dem der Zugriff beim Proxy-Server eintraf. Er wird aus dem Ende des Transfers und seiner Dauer berechnet.
- Die Angabe der Protokollversion “HTTP/1.0” am Ende des HTTP-Requests wird gelöscht und aus der Seiten-URL bereits die Server-URL extrahiert.

Für die Light-Logfiles ergibt sich folgende Struktur:

```
<LightLogfile>        :=  { <Eintrag> }
<Eintrag>             :=  <Zugriffszeipunkt> <Seitengröße>
                          <Cacheladezeit> <Serverladezeit>
                          <LastModifiedDatum> <ArumentFlag>
                          <ServerURL> <Seitenname> CR
<Zugriffszeipunkt>    :=  <Zahl>
<Seitengröße>         :=  <Zahl>
<Cacheladezeit>       :=  <Zeitdauer>
<Serverladezeit>      :=  <Zeitdauer>
<LastModifiedDatum>   :=  <Zahl>
<ArumentFlag>         :=  ’0’ | ’1’
<ServerURL>           :=  Stringliteral
<Seitenname>          :=  ’"’ Name ’"’
<Zeitdauer>           :=  <Zahl> ’.’ <Zahl>
<Zahl>                :=  <Ziffer> { <Ziffer> }
<Ziffer>              :=  ’0’ | ... | ’9’
```

3.3.2 Domänenliste

CaSimiR besitzt die Möglichkeit, bestimmte referenzierte Server aufgrund ihrer Domänenzugehörigkeit zu einem einzigen logischen Server, einem sog. Domain-Server, zusammenzufassen. Die Daten, die sonst für jeden einzelnen dieser Server erhoben worden wären, werden dann alleine für den neuen logischen Server gespeichert. Auf diese Weise sind Messungen von ganzen Servergruppen möglich – bei der Bandbreitenanalyse ist dies nützlich, wenn man die Übertragungsqualität zu allen Servern beispielsweise der `com`-Domäne betrachten will. Um die Zusammenfassung zu aktivieren, muß ein entsprechender Eintrag in die Domänenliste geschrieben werden. CaSimiR unterscheidet dabei zwei verschieden Arten: die normale Domänenzusammenfassung und eine Muli-Domain-Zusammenfassung, bei der mehrere Domänen

1. Bei dem UTC-Format wird das Datum anhand der seit dem 1. Januar 1970 vergangenen Sekunden gemessen.

angegeben werden, aus den die zugehörigen Server zu einem einzigen Domain-Server zusammengefaßt werden. Im Beispiel sieht das folgendermaßen aus:

```
.com
.de .fr .it
```

Die erste Zeile enthält einen einfachen Domänen-Eintrag, und es wird ein Domain-Server für alle WWW-Server der com-Domäne eingerichtet (der Punkt vor der Domänenangabe darf nicht fehlen). Die zweite Zeile enthält einen Multi-Domain-Eintrag. Hier werden alle Server der Domänen de, fr und it zu einem Domain-Server vereinigt.

3.3.3 Cacheparameterfile

Mit dem Cacheparameterfile gibt der Benutzer die gewünschten Cachekonfigurationen an, für die eine Auswertungsserie angefertigt werden soll. Jede Zeile steht für einen durchzuführenden Auswertungslauf. In ihr sind folgende Angaben enthalten:

- Gesamt-Cachegröße (-1 = unendlich großer Cache)
- maximale Seitengröße (-1 = keine Beschränkung)
- prozentuale Größe des Caches, der für replizierte Seiten verwendet werden soll
- Mindestcachingvorteil, den eine Seite für die Einlagerung benötigt (nur für LCP-Verfahren)
- TTL-Faktor

Innerhalb des Parameterfiles können Kommentarzeilen mit einem führenden '#'-Zeichen eingefügt werden. Ein mögliches File könnte so aussehen:

```
#Size     maxPageSize    repliziert  minDelta TTL
1000000   -1             0.2         0         0.1
5000000   100000         0.2         0         0.1
```

Mit diesem Parameterfile werden zwei Simulationsläufe initialisiert. Der erste mit einer Cachegröße von 1000000 Byte und ohne maximale Seitenbeschränkung (20% des Caches werden für Seitenreplikation verwendet, Mindest-Cachingvorteil 0 Sekunden und TTL-Faktor 10%). Der zweite Durchlauf wird mit einer Cachegröße von 5.000.000 Byte und einer Beschränkung der Seitengröße auf 100.000 Byte durchgeführt.

3.3.4 Konfigurationsfile

Mit dem Konfigurationsfile ist die zentrale Kontrolle über den Simulator möglich. Die zahlreichen Parametereinstellungen werden dabei über Wertzuweisungen an sog. *KonfigurationsIds* durchgeführt. Das gesamte Simulationssystem ist in eine hierarchische Modulstruktur untergliedert, wobei jedes konfigurierbare Modul für jeden veränderbaren Parameter ein KonfigurationsId besitzt. Um eine gewünschte Parametereinstellung zu ändern, muß eine Wertzuweisung an das entsprechende KonfigurationsId erfolgen. Dazu muß dem KonfigurationsId, zur eindeutigen Lokalisierung, die Namen der übergeordneten Module vorangestellt werden. Bei KonfigurationsIds im Hauptmodul entfällt diese Angabe natürlich. Eine Zuweisung könnte dann so aussehen:

```
Modul.Submodul.KonfigurationsId = Wert
```

Mit dem Id ist zusätzlich ein Wertetyp verbunden, der die Möglichkeit bietet, numerische oder boolsche Werte zu setzen, Strings anzugeben – beispielsweise um Filenamen zu setzen – oder Aufzählungsnamen zuzuweisen. Letztere Möglichkeit kann z.B. bei der Wahl der Cachingstrategie Verwendung finden. Mit dem Aufzählungstypen sind dann mehrere zulässige Identifier definiert, von denen einer auf der rechten Seite der Zuweisung auftreten darf.

Im Konfigurationsfile sehen die Zuweisungen mit den unterschiedlichen Argumenttypen beispielsweise so aus:

```
Modul.IntegerId = 100
Modul.StringId = ’StringArgument’
Modul.BooleanId = true
Modul.Enumeration = Enum2
```

Man erkennt, daß Stringargumente in Anführungszeichen eingeschlossen werden müssen, in diesem Falle sind einfache Anführungszeichen gewählt worden, doppelte Anführungszeichen sind aber genauso erlaubt. Die Aufzählungsargumente (in unserem Fall „Enum2”) dagegen stehen ohne zusätzliche Begrenzer. Eine Zuweisung ist hier nur dann gültig, wenn der angegebene Identifier in der Liste der erlaubten Aufzählungsidentifier vorkommt. Für den Argumenttyp Boolean sind die Werte `true` und `false` zugelassen. Die folgende EBNF-Darstellung faßt die Syntax noch einmal formaler zusammen.

```
<Konfigurationsfile>  :=  { <KonfigZuweisung> }| <Kommentar>
<Kommentar>           :=  ’#’ Komentar-Text CR
<KonfigZuweisung>     :=  <KonfigurationsId> ’=’ <Argument> CR
<Konfigbezeichner>    :=  <Modulbezeichner> <KonfigID>
<Modulbezeichner>     :=  { <Modulname> ’.’}
<KonfigID>            :=  NAME
<ModulName>           :=  NAME
<Argument>            :=  <String> | <Integer> | <EnumId>
<String>              :=  ’”’ Zeichenfolge ’”’
                          | “’” Zeichenfolge “’”
<Integer>             :=  <Zahl>
<EnumId>              :=  Zeichenfolge
<Zahl>                :=  <Ziffer> { <Ziffer> }
<Ziffer>              :=  ’0’ | ... | ’9’
```

Bemerkung: Der Simulator ist in der Lage, alle Identifier inclusive ihrer Typen aufzulisten (näheres dazu steht bei den Erläuterungen zu den Komandozeilen-Parametern im folgenden Abschnitt). Genauere Informationen, wie die KonfigurationsIds verwendet werden können, enthält die Beschreibung in Abschnitt 3.4.

3.3.5 Kommandozeile

Der Simulator wird über die Kommandozeile aufgerufen. Die für die Simulation einzustellenden Parameter sind dabei komplett in das Konfigurationsfile ausgelagert worden. Das allgemeine Aufrufformat sieht folgendermaßen aus:

```
CaSimiR { “-” <Option> } <Konfigurationsfile>
```

Als Optionen können verwendet werden:

- q: Unterdrückt die Ausgabe während der Simulation bzw. Logfileanalyse.
- h: Gibt eine Liste aller definierten KonfigurationsIds und deren Typen aus.

Das angegebene Konfigurationsfile enthält dann alle übrigen für die Simulation notwendigen Anweisungen (siehe auch folgenden Abschnitt).

3.4 Funktionalität

Dieser Abschnitt enthält eine Beschreibung der KonfigurationsIds von CaSimiR und deren Funktion. Die fett gedruckte Zeile gibt jeweils den kompletten Namen des KonfigurationsIds an. Zusätzlich ist der zugehörige Argumenttyp angegeben und der Modus in dem dieser Identifier eingesetzt werden kann. Im Logfile-Auswertungsmodus z.B. werden Parameter ignoriert, die ausschließlich für den Cache-Auswertungsmodus bestimmt sind und umgekehrt. Zwei Beispiele am Ende des Abschnitts verdeutlichen die Funktionsweise des Konfigurationsfiles nochmals.

RunMode

- Typ: ENUM(LogEval, CacheEval)
- Beschreibung: Über dieses KonfigurationsId wird der Arbeitsmodus des Simulators festgelegt. „LogEval” versetzt den Simulator in den Logfile-Auswertungs-Modus (vgl. Abschnitt 3.2), während „CacheEval” den Simulator im Cache-Analyse-Modus arbeiten läßt. Je nachdem welcher Modus aktiviert ist, beeinflussen andere KonfigurationsIds den Ablauf. Zu jedem KonfigurationsId ist jeweils vermerkt in welchem Modus er aktiv ist. Die Defaulteinstellung für das „RunMode”-Id ist der „LogEval”-Wert.

ProxyLog

- Typ: STRING
- Modus: LogEval
- Beschreibung: Trägt einen neues Logfile als Proxy-Log in die Liste der zu untersuchenden Logfiles ein. Sollen mehrere Proxy-Logs eingelesen werden, kann dem KonfigurationsIds auch mehrfach ein Wert zugewiesen werden. Die Reihenfolge der Zuweisung bestimmt dabei auch die Reihenfolge der Auswertung. Sollen mehrere hintereinander liegende Monate analysiert werden, müssen die Logfiles auch in dieser Reihenfolge angegeben werden.

CacheLog

- Typ: STRING
- Modus: LogEval
- Beschreibung: Wie auch beim „ProxyLog”-Identifier wird ein neues Cache-Logfile für die Analyse angemeldet. Es gelten dieselben Reihenfolgeabhängigkeiten. Sind zusätzlich ein oder mehrere Proxy-Logfiles eingetragen worden, versucht CaSimiR aus je zwei an derselben Position in die Proxy- bzw. Cache-Logfile-Liste eingetragenen Logfiles zu verschmelzen und die in beiden Logfiles enthaltenen Zugriffe gemäß ihres Anfangszeitpunktes zu sortieren (die Einträge in den Logfiles selbst sind nach dem Zeitpunkt der Beendigung geordnet). Diese Umsortierung ist notwendig, um die verschiedenen Caching-Strategien mit dem ursprünglichen Referenzverhalten zu testen und geht bei der Erzeugung der Light-Logfiles mit ein.

LastModifiedLog

- Typ: STRING
- Modus: LogEval
- Beschreibung: Trägt ein LastModified-Logfile für die Analyse ein. Sind zusätzlich Proxy- und Cache-Logs angemeldet, werden die Zugriffsinformationen aus Proxy- und Cache-Logfiles um LastModified-Informationen ergänzt.

NewLogFileName

- Typ: STRING
- Modus: LogEval
- Beschreibung: Ist dieser Name angegeben, erzeugt CaSimiR ein Light-Logfile mit diesem Namen. Das File wird aus den angegebenen Proxy-, Cache- und LastModified-Logs erzeugt. Für die Erzeugung muß ebenfalls zwecks Übertragungszeitenschätzung auf entsprechende Profile zurückgegriffen werden, die dazu unbedingt hinzu geladen werden müssen (siehe weiter unten: Statistics.LoadProfiles und Statistics.ProfilesFile).

EvaluationDataFileName

- Typ: STRING
- Modus: LogEval, CacheEval
- Beschreibung: Hier können Informationen über die Cache-Performance gespeichert werden. Ist der LogEval-Modus eingestellt, wird der zum Zeitpunkt der Logfileerstellung aktive Proxy-Cache bewertet. Ist der CacheEval-Modus gewählt worden, wird das aktuell gewählte Caching-Verfahren bewertet (siehe auch Proxy.CachingStrategy). Es werden mittlere Seitenzugriffszeit, Cache-Hit-Rate, Byte-Hit-Rate, die Quote der 304-Server-Replys, durchschnittlich übertragene Bytes pro Zugriff und insgesamt für den Meßzeitraum übertragene Bytes gespeichert.

LightLog

- Typ: STRING
- Modus: CacheEval
- Beschreibung: Wählt ein Light-Logfile für den Cache-Auswertungslauf. Auch hier ist eine Mehrfachangabe möglich.

CachePreLoadFileName

- Typ: STRING
- Modus: CacheEval
- Beschreibung: Wählt ein Light-Logfile aus, mit dessen Hilfe der zu testende Cache vor dem eigentlichen Auswertungslauf mit Seiten gefüllt wird. Das Cache-Verhalten während dieses Auffüllvorgangs hat keinerlei Einfluß auf die Messungen.

ParamFileName

- Typ: STRING
- Modus: CacheEval
- Beschreibung: In diesem File sind alle Cache-Parameter-Konfigurationen gespeichert, für die das Caching-Verfahren untersucht werden soll. Die Ergebnisse jedes einzelnen Durchlaufs werden in derselben Reihenfolge in das Ergebnisfile (vgl. EvaluationFileName) geschrieben, wie sie im Parameterfile auftraten.

ReplicatedPagesFileName

- Typ: STRING
- Modus: CacheEval
- Beschreibung: Sollen in die Simulation einer Caching-Strategie auch mit einer Menge von replizierten Seiten gearbeitet werden, muß hier ein entsprechendes File angegeben werden, das eine entsprechende Seitenliste enthält. Der simulierte replizierte Bereich wird dann mit Seiten aus dieser Liste aufgefüllt. Eventuell wird nur ein Teil der Seiten geladen, falls der replizierte Bereich schon vollständig gefüllt wurde.

Proxy.CachingStrategy

- Typ: ENUM(None, Random, TTL, LRU, LCP)
- Modus: CacheEval
- Beschreibung: Mit diesem KonfigurationsId kann die gewünschte Caching-Strategie gewählt werden (siehe Abschnitt 2.3 auf Seite 7 und Beschreibung auf Seite 62).

Statistics.ServerNamesFile

- Typ: STRING
- Modus: LogEval
- Beschreibung: Wird hier ein Filename angegeben, speichert CaSimiR eine Liste aller referenzierten Server.

Statistics.PageNamesFile

- Typ: STRING
- Modus: LogEval, CacheEval
- Beschreibung: Eine Liste aller referenzierten Seiten wird im Falle eines angegebenen Filenamens gespeichert.

Statistics.LoadProfile

- Typ: BOOL
- Modus: LogEval, CacheEval
- Beschreibung: Signalisiert CaSimir, daß bereits gespeicherte Profile aus der bei dem KonfigurationsId `Statistics.ProfilesFile` angegebenen Datei geladen werden soll.

Statistics.SaveProfile

- Typ: BOOL
- Modus: LogEval
- Beschreibung: Speichert neu berechnete Übertragungsprofile in die bei dem KonfigurationsId `Statistics.ProfilesFile` angegebene Datei. Das Speichern von Profilen ist nur für den LogEval-Modus vorgesehen.

Statistics.ProfilesFile

- Typ: STRING
- Modus: LogEval, CacheEval
- Beschreibung: Hier wird der Filename der Datei angegeben, aus der Übertragungsprofile gelesen bzw. in die Profile gespeichert werden sollen.

Statistics.CalculateProfiles

- Typ: BOOL
- Modus: LogEval
- Beschreibung: Ist das KonfigurationsId auf true gesetzt, berechnet CaSimiR aus den Übertragungszeiten der eingelesenen Logfiles und unter Berücksichtigung der gewählten Zeitauflösung sowie des gewünschten Schätzmodells die Übertragungsprofile.

Statistics.TimingSamplesFile

- Typ: STRING
- Modus: LogEval
- Beschreibung: Zur Visualisierung von Übertragungszeiten wird ein Datenfile mit dem angegebenen Namen erzeugt. Eine Beschränkung der dazustellenden Meßpunkte in Bezug auf die Seitengröße läßt sich mit den KonfigurationsIds `Statistics.MinGridPageSize` und `Statistics.MaxGridPageSize` erreichen (s.u.).

Statistics.MinGridPageSize

- Typ: INT
- Modus: LogEval
- Beschreibung: Mit diesem KonfigurationsId läßt sich die untere Schranke für die in der Übertragungszeitenvisualisierung darzustellenden Meßpunkte festlegen.

Statistics.MaxGridPageSize

- Typ: INT
- Modus: LogEval
- Beschreibung: Analog zu `Statistics.MinGridPageSize` kann hier eine obere Schranke für die darzustellende Seitengröße festgelegt werden.

Statistics.ProfileGridDataFile

- Typ: STRING
- Modus: LogEval
- Beschreibung: Erzeugt ein Datenfile zur Profilevisualisierung. Aus den gespeicherten Profilen werden dreidimensionale Gitter generiert. Dabei ist dessen Auflösung auf der x-Achse von der Tageszeitauflösung des Profils abhängig. Die Auflösung auf der y-Achse kann über `Statistics.YGridResolution` eingestellt werden.

Statistics.YGridResolution

- Typ: INT
- Modus: LogEval
- Beschreibung: Legt die Auflösung des Profile-Visualisierungsgitters auf der y-Achse fest.

Statistics.ProfileErrorDistribution

- Typ: STRING
- Modus: LogEval
- Beschreibung: Schätzt die aus den Logfiles eingelesenen Übertragungszeiten und vergleicht sie mit den ursprünglichen Werten. Die daraus errechneten Schätzfehler werden zu einer relativen Häufigkeitsverteilung zusammengefaßt. Für die Berechnung müssen unbedingt Profile geladen oder neu generiert worden sein.

Statistics.MeanPageError

- Typ: STRING
- Modus: LogEval
- Beschreibung: Analog zu `Statistics.ProfileErrorDistibution` wird der mittlere Schätzfehler in Abhängigkeit zur Seitengröße errechnet.

Statistics.BWDataFilePrefix

- Typ: STRING
- Modus: LogEval
- Beschreibung: Um die verschieden Bandbreiteninformationen abzuspeichern, muß über dieses KonfigurationsId ein Fileprefix angegeben werden. Er kann z.B. auch schon Pfadangaben enthalten. Ein Prefix 'data/bw' sorgt z.B. dafür, daß alle erzeugten Datenfiles im 'data' Verzeichnis angelegt werden. CaSimiR verwendet diesen Prefix als Basis für die Namen der generierten Datenfiles: Sind keine speziellen Domänen angegeben worden (siehe `Statistics.Server.LoadDomains`), führt CaSimir zunächst eine künstliche Domäne mit dem Namen 'WORLD' ein, die Zugriffe auf sämtliche WWW-Server werden dann zu einer Messung zusammengefaßt. Sind Domänen angegeben worden, wird für jede einzelne eine Bandbreiten-Datenerhebung durchgeführt. Jede dieser Erhebungen wird in ein eigenes Datenfile geschrieben. Der Name ergibt sich aus dem Prefix und dem Domänennamen wie folgt: '<Prefix>.<Domänenname>.dat'. Ohne angegebene Domänen würde mit dem oben genannten Beispielpräfix ein Datenfile mit dem Namen 'data/bw.WORLD.dat' erzeugt.

Statistics.Bandwidth.UnitSize

- Typ: INT
- Modus: LogEval
- Beschreibung: Mit dieser Einstellung läßt sich die Zeitauflösung der Bandbreitendiagramme festlegen. Der angegebene Wert gibt dabei an, wieviel Sekunden zu einer Einheit zusammengefaßt werden sollen. Ein Wert von 60 bedeutet also, daß im Bandbreitendiagramm ein dargestellter Wert einen Zeitabschnitt von einer Minute repräsentiert.

Statistics.Servers.LoadDomains

- Typ: STRING
- Modus: LogEval
- Beschreibung: Mit Hilfe dieses KonfigurationsIds ist die Gruppierung von Servern nach ihrer Domänenzugehörigkeit möglich. Die gruppierten Server werden vom Simulator in diesem Fall als ein einziger Server behandelt, für den ohne Unterschied zu nicht gruppierten Servern alle Daten erhoben werden können.

Statistics.Servers.LoadHopInfo

- Typ: STRING
- Modus: LogEval
- Beschreibung: Dem Simulator soll hier die Möglichkeit eingeräumt werden, Server nicht nur nach Domänen zu gruppieren, sondern auch entsprechend ihrer Entfernung zum Proxy-Server (gemessen in Hops). Diese Fähigkeit ist aber z.Z. noch nicht implementiert.

Statistics.Timing.DefaultTimeSlots

- Typ: INT
- Modus: LogEval
- Beschreibung: Setzt die bei den Übertragungsprofilen zu verwendende Zeitauflösung fest. Ein Wert von 24 bedeutet z.B. eine Tageszeitauflösung von 24 Abschnitten.

Statistics.Timing.EstimationModel

- Typ: ENUM(Average, StraightLine, SimpleStraightLine)
- Modus: LogEval
- Beschreibung: Wählt das Schätzmodell, mit dem die Übertragungsprofile errechnet werden sollen.

Zwei Beispiele zur Konfiguration sollen den Gebrauch des Konfigurationsfiles klar machen:

```
RunMode = LogEval
ProxyLog = 'logs/ProxyJan96.gz'
CacheLog = 'logs/CacheJan96.gz'
ProxyLog = 'logs/ProxyFeb96.gz'
CacheLog = 'logs/CacheFeb96.gz'
Statistics.Server.LoadDomains = 'domains.txt'
```

```
Statistics.Bandwidth.BWDataFilePrefix = 'data/bw'
```

Mit diesem Konfigurationsfile wird ein Logfile-Auswertungslauf gestartet. Ausgewertet werden die Proxy- und Cache-Logs aus den Monaten Januar und Februar. Die Reihenfolge der „ProxyLog”-Anweisungen bzw. der „CacheLog”-Anweisungen untereinander muß dabei unbedingt eingehalten werden. CaSimiR lädt aus der Datei „domains.txt” eine Liste der Domänen, für die die in der nächsten Zeile aktivierte Bandbreitenanalyse durchgeführt wird. Diese Informationen werden in den Dateien „data/bw.<Domänenname>.dat” gespeichert. Beispiel zwei zeigt, wie ein Cacheauswertungslauf gestartet werden kann:

```
RunMode = CacheEval
LightLog = 'logs/LightFeb96.gz'
Proxy.CachingStrategy = LCP
CachePreLoadFileName = 'logs/LightJan96.gz'
ReplicatedPagesFileName = 'pagelist.txt'
ParamFileName = 'cacheparams.par'
EvaluationDataFileName = 'data/results.dat'
```

Der Auswertungslauf wird für das LCP-Verfahren mit den Einträgen aus dem Februar-Light-Logfiles durchgeführt. Dazu wird vor jeder neuen Cache-Konfiguration der Cache mit Hilfe der Einträge aus dem Monat Januar gefüllt. Die zu testenden Konstellationen stehen in der Datei „cacheparams.par”. Ist bei einer der angegebenen Konstellationen ein Teil des Caches für replizierte Seiten vorgesehen, wird dieser mit den Seiten der Seitenliste „pagelist.txt” gefüllt (die Seitenliste hat das gleiche Format wie die von CaSimiR erzeugte Seitenliste). Die Ergebnisse werden schließlich in der Datei „data/results.dat” abgelegt.

3.5 Spezifikation der Ausgaben

CaSimiR erzeugt je nach Wusch eine Reihe von Datenfiles, die dann z.B. von GNUPLOT zur Erzeugung von Graphiken verwendet werden können. Auch eine Weiterverwendung der Daten für eine weiterführende Analyse, z.B. mit anderen Werkzeugen ist denkbar und wünschenswert. Dieser Abschnitt enthält deshalb eine Beschreibung der verschiedenen von CaSimiR erzeugbaren Datenfiles.

Zuvor jedoch noch eine Bemerkung zu der gewählten Darstellung: bis auf Server- und Seitenliste sind die Datenfiles auf die Verwendung durch GNUPLOT zugeschnitten. GNUPLOT benötigt für die Darstellung eines zweidimensionalen Graphen eine Menge von Wertepaaren, welche aus zwei verschiedenen Spalten des Datenfiles gelesen werden (für ein dreidimensionales Diagramm entsprechend drei Spalten). Die Auflistungen weiter unten geben für jedes Datenfile jeweils an, welche Informationen in den einzelnen Spalten enthalten sind. Für die Generierung der Diagramme müssen somit nur die entsprechenden Spalten für x-, y- und ggf. z-Achse gewählt werden.

Serverliste
In dieser Datei werden alle während der Analyse referenzierten Server aufgelistet. Für jeden Server existiert eine Zeile, die folgende Informationen enthält (die Angaben im Datenfile sind jeweils durch ein Leerzeichen voneinander getrennt):

- Anzahl der Zugriffe auf den Server
- Anzahl der Cache-Hits bei Zugriffen auf Seiten dieses Servers
- Anzahl der Cache-Misses
- Server-URL (normalisiert)

Seitenliste
Analog zu der Serverliste kann eine Seitenliste generiert werden. In ihr sind diese Informationen enthalten:

- Anzahl der Zugriffe auf die Seite
- Anzahl der Cache-Treffer bei Zugriffen auf die Seite
- Größe der Seite in Bytes
- Insgesamt während des Durchlaufs erzielter Caching-Gewinn (diese Information ist nur im Cache-Auswertungs-Modus verfügbar)
- Seiten-URL (normalisiert)

Datenfile für Übertragungszeiten
Zur Visualisierung der Übertragungszeiten, die für die Server erhoben wurden, kann ein entsprechendes Datenfile erzeugt werden. Die Datei ist ein sog. „Multi Mesh"-File, d.h. in ihm sind durch doppelte Leerzeichen getrennt die Übertragungszeiten für mehrere Server enthalten [WIL93]. Vor jedem neuen Datenfeld wird der aktuelle Index und die Server-URL angegeben (mit führendem „#"). Die Datenfelder enthalten diese Werte:

- Uhrzeit des Zugriffs (Auflösung: 1 Stunde)
- Größe der übertragenen Seite
- Dauer der Übertragung in Sekunden

Übertragungsprofil-Visualisierung
Um die berechneten Übertragungsprofile graphisch darstellen zu können, wird dieses Ausgabefile generiert. Das File enthält die Daten, mit denen ein dreidimensionales Gitter verschiedener Profile, in y-Richtung sowohl mit äquidistanten als auch mit logarithmischen Linienabständen, dargestellt werden können. Das File ist ebenfalls ein „Multi Mesh"-File, das im einzelnen folgende Daten gespeichert:

- Tageszeit (Auflösung ist Profilabhängig)
- Seitengröße (logarithmischer Abstand)
- Übertragungszeit (logarithmischer Abstand)
- Seitengröße (äquidistanter Abstand)
- Übertragungszeit (äquidistanter Abstand)
- „Fast Null" (aus GNUPLOT-technischen Gründen mit eingefügt)
- Bei der Berechnung eingegangene Anzahl der Meßwerte

Die Tageszeit wird auf der x-Achse, die Seitengröße auf der y-Achse abgetragen.

Fehlerverteilung bei der Übertragungszeitenschätzung
Dieses File enthält sowohl die normale als auch die summierte Fehlerverteilung, die für die gespeicherten Übertragungszeiten bei einer Schätzung mit den geladenen Profilen aufgetreten ist. Die Spalten enthalten folgende Daten:

- Schätzfehler in Sekunden
- Absolute Fehlerzahl
- Normierte Anzahl der Fehler

- Summierte relative Häufigkeit der Schätzfehler

Mittlerer Schätzfehler bezogen auf die Seitengröße
Mit diesen Daten können Diagramme über den mittleren Schätzfehler in Bezug auf die Seitengröße erstellt werden. Im einzelnen enthalten die Spalten des Datenfiles folgende Informationen:

- Seitengröße in Bytes
- mittlerer Schätzfehler in Sekunden
- absoluter Schätzfehler in Sekunden
- Anzahl der Schätzungen

Übertragungsbandbreite
Die aus den Logfiles ermittelte Bandbreite ist in diesen Datenfiles gespeichert. Zusätzlich aufgenommen wurden die Messung der Zugriffsaktivität und einer gewichteten Aktivität. Während die einfache Aktivität die Anzahl der Übertragungen im Laufe eines Tages registriert, gewichtet das zweite Aktivitätsmaß diese Übertragungen entsprechend ihrer Dauer. Somit ergeben viele kurze hintereinanderliegende Zugriffe gleiche Aktivitätswerte, wie bei einer einzigen Übertragung über denselben Zeitraum. Die gewichtete Aktivität wird damit zu einem Maß für die Belastung der Netzverbindungen durch die einzelnen Zugriffe. Die Angaben verteilen sich folgendermaßen auf die Spalten des Datenfiles:

- Tageszeit
- Bandbreite in Bytes pro Sekunde
- Gewichtete Aktivität
- Zugriffsaktivität

Cache-Bewertung
Diese Datei enthält die Ergebnisse aus den Simulationsläufen mit den im Cacheparameter-File angegebenen Einstellungen. Für jede simulierte Einstellung wird eine eigene Zeile in diese Datei eingetragen. Zur Identifikation wird den Ergebnissen deshalb die zugehörige Cachekonfiguration vorangestellt. Damit ergibt sich folgende Spaltenverteilung:

Cache-Konfiguration:

- maximale Cachegröße in Byte
- maximale Seitengröße in Byte
- Prozentualer Anteil des Cachespeichers, der für die Verwaltung replizierter Seiten genutzt werden soll.
- Mindest-Caching-Gewinn für einzulagernde Seiten
- TTL-Faktor

Bewertungsdaten:

- Durchschnittliche Zugriffszeit
- Cache-Hit-Rate
- Byte-Hit-Rate

- Rate der Fälle, in denen veraltete Seiten aus dem Cache geliefert wurden (Stale-Rate)
- Anteil der nicht notwendigen Get-If-Modified-Requests (403-Reply-Rate)
- Durchschnittlich übertragene Bytes pro Seite
- Anzahl der Zugriffe im Simulationszeitraum
- Insgesamt von den Servern übertragene Bytes im Simulationszeitraum

Bemerkung: Als nicht notwendig gilt ein Get-If-Modified-Request, falls der entsprechende Server auf den Request mit einem "Not Modified" (Reply-Code: 304) antwortet. Bei den durchschnittlich übertragenen Bytes pro Seite und den insgesamt von allen Servern übertragenen Bytes wird jede dieser Nachrichten mit 100 Byte vermerkt – dies ist die ungefähre Größe des HTTP-Headers, der diese Nachricht enthält.

3.6 Systembeschreibung

Die Beschreibung des Softwaresystems in diesem Abschnitt folgendermaßen gegliedert: zunächst wird in Abschnitt 3.6.1 die Systemstruktur vorgestellt. Die einzelnen Softwarekomponenten werden vorgestellt und ihre Beziehung zueinander wird in einem Strukturgraphen veranschaulicht. Der Darstellung des internen Datenflusses in Abschnitt 3.6.2 folgt dann abschließend eine Komponentenbeschreibung, die auf die verwendeten Konzepte und Algorithmen genauer eingeht.

3.6.1 Systemstruktur

In der folgenden Aufstellung wird zwischen Infrastrukturkomponenten, welche elementare Grundaufgaben übernehmen, und Hauptkomponenten, die die eigentliche Funktionalität erbringen, unterschieden. Zu jeder Komponente gehört ein Headerfile

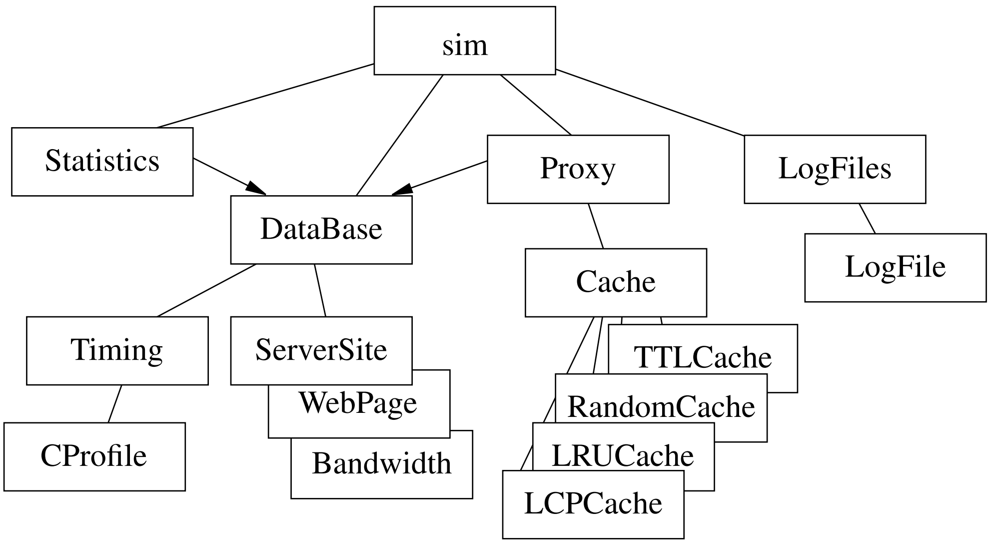


Abbildung 3-3: Systemstruktur des Cache-Simulators CaSimiR

(<Modulname>.h) und ein Implementationsfile (<Modulname>.cc). In jeder Komponente ist mindestens eine Klasse mit identischem Namen definiert, über die die angebotene Funktionalität des Moduls genutzt werden kann. Gegebenenfalls sind zusätzliche

Hilfsklassen in die Komponente mit integriert oder werden aus untergeordneten Komponenten eingebunden. Den Zusammenhang der Komponenten wird in dem aus Übersichtlichkeitsgründen vereinfacht dargestellten Strukturgraphen (Abb. 3-3) aufgezeigt.

Infrastruktur-Komponenten

- Message
  Implementiert eine Klasse für eine während der Laufzeit aus- und einschaltbare Ausgabe.
- Time
  Klasse zur Verwaltung von Zeitangaben und der Umrechnung von lokaler Zeit in GMT und umgekehrt.
- TimeConversion
  Konvertierung von Datumsangaben zwischen der UTC-Repräsentation und der systemabhängigen „tm"-Struktur. Außerdem: Konvertierung der „timeval"-Repräsentation von Zeitdauern in einen „double"-Wert.
- ConfigMan
  Komponente zur Konfiguration der Systemmodule (siehe Seite 56).
- Argument
  Stellt Verwaltungsroutinen für die bei der Konfiguration verwendeten Argumenttypen bereit.
- ErrorsNWarnings
  Definiert Klassen zur Definition von Warnungen und Fehlermeldungen sowie deren einheitlichen Aufruf.
- GZFile
  Implementiert das Einlesen eines mit „gzip" [GZIP] komprimierten Files.

Hauptkomponenten

- sim
  Enthält die Definition der Funktion „main". In diesem Modul wird der Ablauf einer Simulation bzw. Analyse kontrolliert (siehe Seite 54).
- LogFile
  Realisiert das Einlesen von Einträgen der verschiedenen Logfiletypen (Proxy, Cache, LastModified und Light). Die Komponente realisiert eine identische Sicht auf komprimierte und unkomprimierte Logfiles.
- LogFiles
  Faßt die Logfiles der Typen Proxy, Cache und LastModified zu einem entsprechend mehr Informationen enthaltenden logischen Logfile zusammen und abstrahiert von den typenspezifischen Eigenschaften der Einträge (siehe Seite 64).
- LogEntry
  Hier stehen die Datenstrukturen, die die Logfile-Einträge speichern. Für alle vier verschiedenen Logfiletypen sind entsprechende Klassen definiert.
- Statistics
  Diese Komponente ist zuständig für die Erhebung und Speicherung gewünschter Daten. Dabei kann sie die Dienste der DataBase-Komponente in Anspruch nehmen.

- DataBase
  Diese kontrolliert die verschiedenen Verwaltungskomponenten, die auf die Speicherung bestimmter Daten spezialisiert sind (WebPage, ServerSite, AccessTime, Bandwidth, Timing).
- WebPage
  Übernimmt die Aufgabe der Seitenverwaltung und der mit den Seiten assoziierten Information, wie z.B. Referenzzähler, Seitengröße usw. (siehe Seite 58).
- ServerSite
  Analog zur Komponente „WebPage” werden hier serverspezifische Informationen verwaltet (siehe auch Seite 59).
- AccessTime
  Komponente zur Erfassung von Zugriffszeiten.
- Bandwidth
  Komponente zur Erfassung von Bandbreiten-Informationen. Jeder Zugriff (mit Uhrzeit, Seitengröße und Übertragungsdauer) wird notiert und zu den entsprechenden Diagrammen aufbereitet.
- Timing
  Stellt die Aufrufschnittstelle rund um die Übertragungszeiten zur Verfügung. Dazu gehören Laden und Ausgeben der Übertragungszeiten, Berechnung, Laden, Speichern und Visualisierung der Übertragungsprofile sowie die Schätzung von Übertagungszeiten (siehe auch Seite 60).
- CProfile
  Definiert die Schnittstelle für die Komponenten zur Übertragungsprofilberechnung und der Übertragungszeitenschätzung. Alle Klassen, die in den folgenden drei Komponenten definiert werden, erben diese Schnittstelle, was ihre Austauschbarkeit untereinander garantiert.
- AvgProfile
  Implementiert das Schätzverfahren „Average”.
- StraightLineProfile
  Implementiert das Schätzverfahren „StraightLine”
- SimpleStraightLineProfile
  Implementiert das Schätzverfahren „SimpleStraightLine”
- Proxy
  Diese Komponente stellt die Verbindung zwischen der zentralen Kontrollroutine und den cacheimplementierenden Komponenten dar. Die Komponente ist als Basis für weitere Erweiterungen (z.B. Replikationsheuristik, zusätzliche WLIS-Einbindung) eingeführt worden.
- Cache
  Definiert die Schnittstelle für die einzelnen cacheimplementierenden Komponenten (RandomCache, TTLCache, LRUCache und LCPCache). Infrastruktur-Funktionalität insbesondere für die Verwaltung replizierter Seiten wird bereitgestellt (siehe Seite 61).
- RandomCache
  Implementiert ein Caching-Verfahren mit einer zufälligen Seitenverdrängung.

- TTLCache
  Implementiert ein Caching-Verfahren, das lediglich durch Löschen von Seiten mit abgelaufenem Expires-Datum Platz für neu einzulagernde Seiten geschafft (der TTL-Faktor ist deshalb mit für eine Auslagerung entscheidend).
- LRUCache
  Implementiert ein Caching-Verfahren mit einer LRU-Verdrängungsstrategie.
- LCPCache
  Implementiert das LCP-Verfahren (vgl Seite 62).
- Activities
  Die hier definierte Klasse speichert Informationen über den letzten Cachezugriff. Es wird vermerkt, ob es beim letzten Zugriff zu einem Cache-Hit kam, ob dabei ein Get-If-Modified-Request an den entsprechenden Server abgesetzt wurde, ob eine neue Seite in den Cache eingelagert wurde oder ob eine aus dem Cache geladene Seite zu diesem Zeitpunkt nicht mehr aktuell war. Diese Angaben werden dann in der Statistics-Komponente ausgewertet und die Ergebnisse entsprechend gespeichert.

3.6.2 Interner Datenfluß

Abb. 3-4 stellt den Datenfluß innerhalb des Softwaresystems dar. Das Diagramm zeigt das Zusammenspiel der einzelnen Komponenten und gibt einen Eindruck davon, welche Informationen zwischen den Komponenten ausgetauscht werden. Die Logfiles

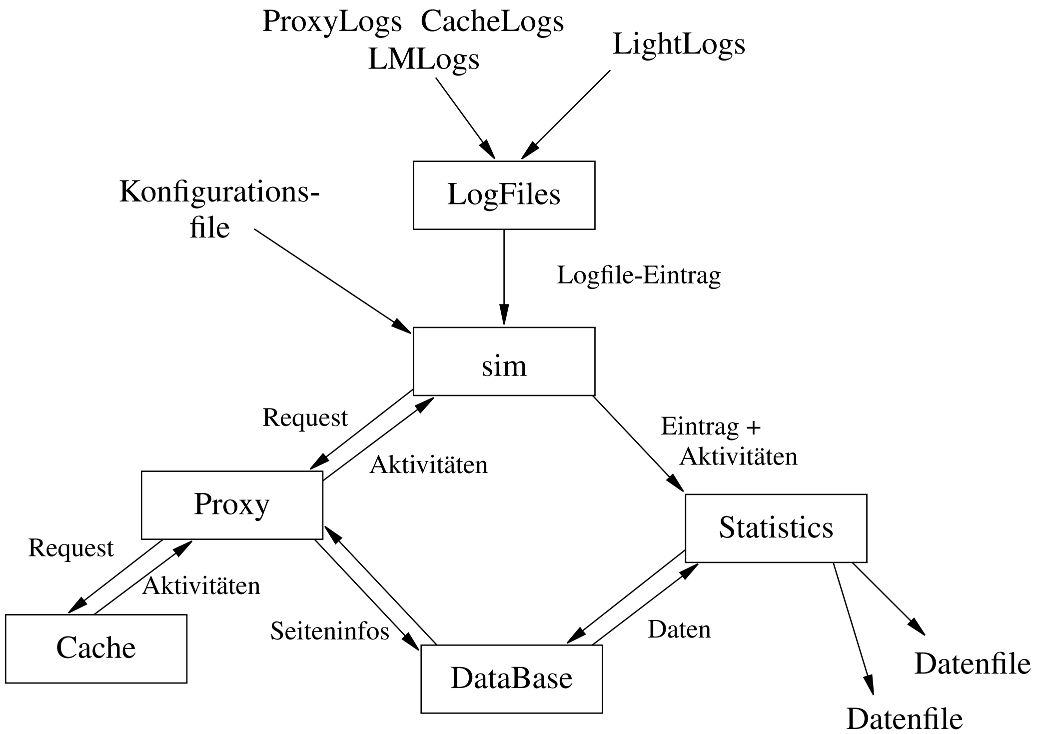


Abbildung 3-4: Interner Datenfluß des Cache-Simulators CaSimiR

werden über die Komponente „LogFiles” eingelesen. Konfigurationsfile und weitere Eingabefiles werden während der Konfigurationsphase bei den entsprechenden Komponenten eingelesen. Die zentrale Steuerkomponente „sim” liest nun, je nach Betriebs-modus, einen Eintrag nach dem anderen aus den Logfiles – diese sind zeitlich so geordnet, wie die Zugriffe ursprünglich an den Proxy-Server abgesetzt wurden, also nach dem Startzeitpunkt jeder Anfrage. Arbeitet der Simulator im LogEval-Modus, wird der eingelesene Logfile-Eintrag an die Statistics-Komponente übergeben, die daraus die im Konfigurationsfile angegebenen Daten extrahiert und zur späteren Ausgabe in die entsprechenden Datenfiles speichert.

Arbeitet CaSimiR dagegen im CacheEval-Modus, muß zunächst (nach dem Einlesen des Logfile-Eintrags) ein simulierter Cachezugriff ausgeführt werden. Dazu werden der Proxy-Komponente der Logfile-Eintrag mit dem darin enthaltenen HTTP-Request übergeben. Der anschließende Cacheaufruf und die damit verbundenen Aktivitäten innerhalb des Caches werden in einer Activities-Struktur festgehalten. Daraufhin erfolgt ebenfalls eine Auswertung durch die Statistics-Komponente. Während der Simulation können sowohl Proxy und Cache als auch die Statistics-Komponente auf Informationen aus der Datenbank zugreifen.

3.6.3 Komponentenbeschreibung

Komponente „sim"

Wie bereits bei der Darstellung der Systemstruktur erwähnt, enthält die sim-Komponente die zentralen Kontrollroutinen, die den Analyse- bzw. Simulationsablauf steuern. An zwei Flußdiagrammen sollen die Abläufe im LogEval- bzw. CacheEval-Modus noch einmal dargestellt werden:

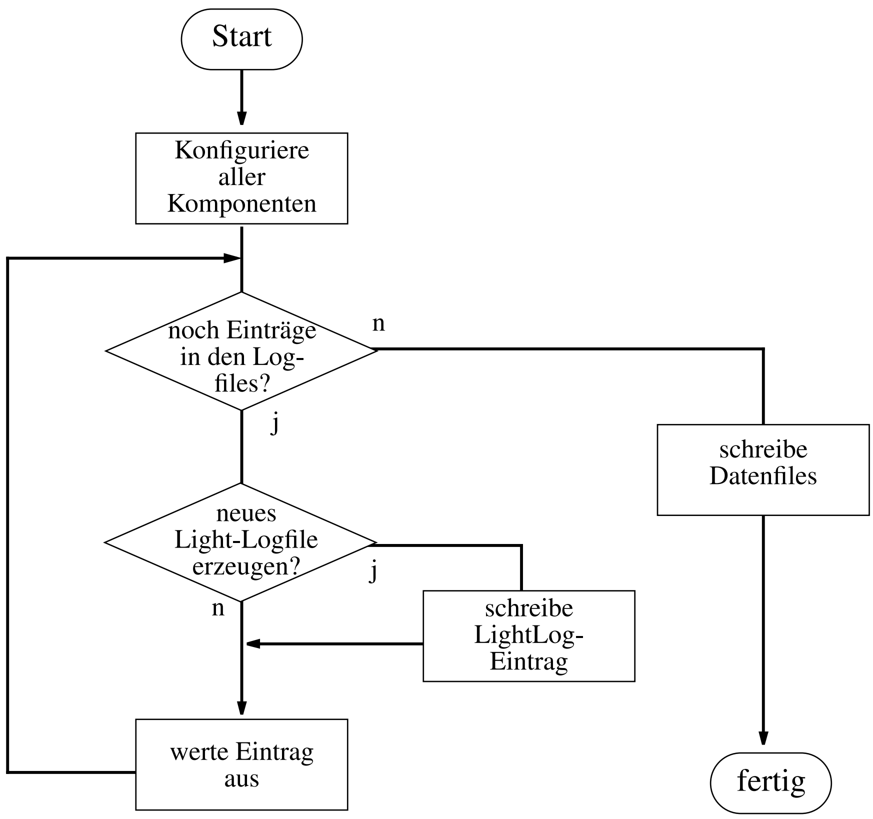


Abbildung 3-5: Ablauf einer Logfile-Analyse

Ein Auswertungslauf startet, genauso wie der Simulationslauf, mit der Konfiguration der Komponenten. Hier werden unter anderem auch die zu lesenden Logfiles angegeben (siehe Abschnitt 3.3.4). Aus diesen Logfiles werden die Einträge der Reihe nach eingelesen (über die Komponente „LogFiles” Seite 64). Soll ein Light-Logfile erzeugt werden, wird ein entspechender Eintrag generiert, ansonsten wird jeder Eintrag entsprechend der Konfiguration von der Statistics-Komponente ausgewertet. Sind alle Einträge gelesen, bereitet die Statistics-Komponente die gespeicherten Daten auf und erzeugt die gewünschten Datenfiles.

Der Ablauf bei einer Cachesimulation ist etwas umfangreicher: nach der Konfigura-

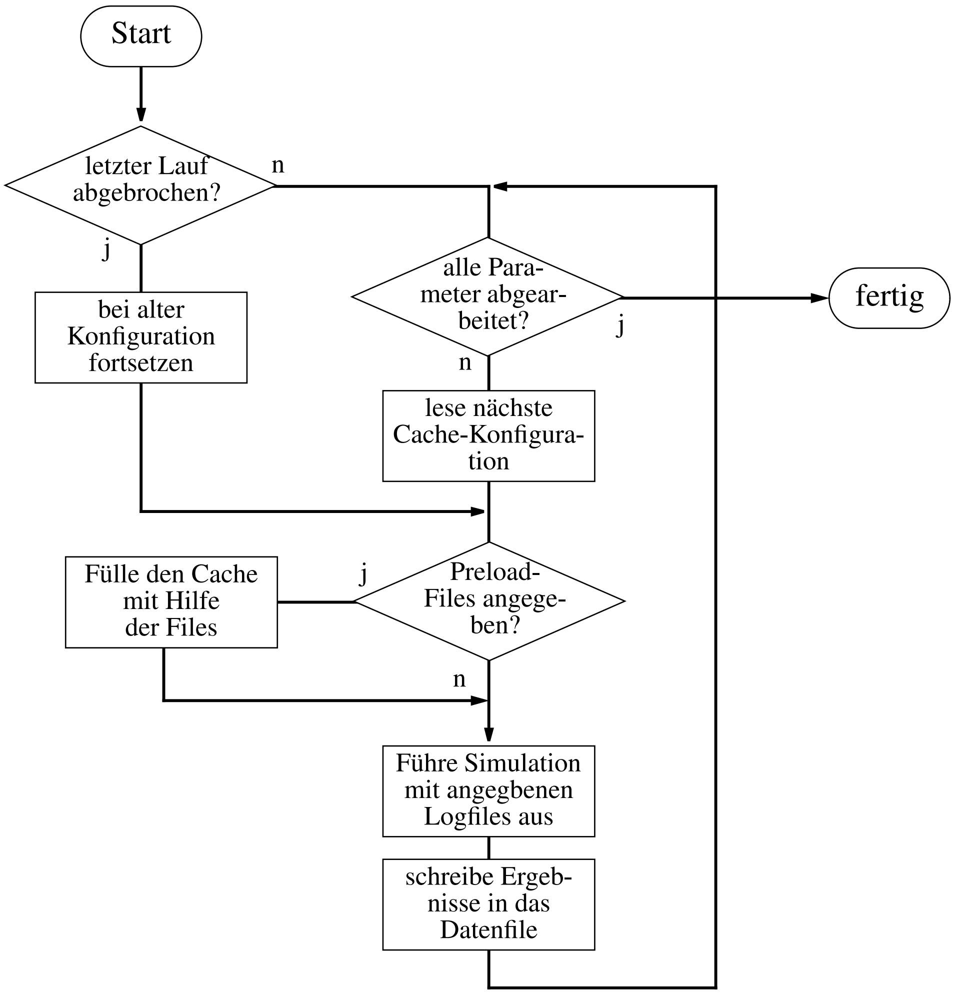


Abbildung 3-6: Ablauf einer Cachesimulation

tion muß CaSimiR zunächst überprüfen, ob eine vergangener Simulationslauf abgebrochen wurde. Dazu testet er die Existenz des Files „sim.current”. Dieses File enthält die aktuell bearbeitete Cachekonfiguration – sie wird zu Beginn eines jeden Durchlaufs dort abgespeichert – und wird nach der Beendigung einer Auswertungsreihe wieder gelöscht. Aus der Existenz dieses Files kann also darauf geschlossen werden, daß die letzte Auswertungsreihe nicht ordnungsgemäß abgeschlossen wurde und bei der gespeicherten Konfiguration neu aufgesetzt werden muß. CaSimiR beginnt den zuletzt abgebrochenen Simulationslauf von neuem, die bis zum Abbruch simulierten Zugriffe des nicht vollendeten Simulationslaufs müssen wiederholt werden. Nachdem die zu simulierende Cachekonfiguration ermittelt wurde, muß der Cache eingerichtet und evtl.

mit Seiten gefüllt werden. Grundlage dafür sind ein oder mehrere Light-Logfiles. Die dort enthaltenen Einträge werden, wie im späteren Simulationslauf auch, als Anfragen für den Cache verwendet, der dann über eine Einlagerung der Seite und evtl. Auslagerung anderer Seiten entscheidet.

Sind die Preload-Files abgearbeitet, beginnt die eigentliche Simulation (bisher wurden keine Daten über das zu testende Caching-Verfahren erhoben). Wieder werden alle Einträge aus den angegebenen Logfiles gelesen und als Anfragesequenz für das Caching-Verfahren verwendet. Im Gegensatz zum Preload von Seiten, werden die in einer Activities-Struktur festgehaltenen Cacheaktivitäten während der Anfrage nun für die Datenerhebung genutzt. Die Activities-Struktur speichert für jede simulierte Anfrage folgende Informationen:

- Zugriff war ein Cache-Hit.
- Es wurde ein Get-If-Modified abgesetzt, um die Aktualität der referenzierten Seite zu überprüfen.
- Get-If-Modified hat die Aktualität der Seite bestätigt (304-Server-Reply).
- Seite wurde eingelagert.
- Gelieferte Seite war nicht mehr aktuell (stale).

Entsprechend dieser Angaben kann also nun der jeweilige Zugriff von der Statistics-Komponente ausgewertet werden. Dieser Vorgang wird für alle im Parameterfile notierten Cachekonfigurationen durchgeführt. Das Zurücksetzten der Datenbank vor jedem neuen Lauf darf zudem nicht vergessen werden.

Komponente „ConfigMan”

Um eine flexible und automatisierte Konfiguration des Simulationssystems zu erreichen, die bei Änderungen der Systemstruktur ohne Modifikationen angepaßt werden kann, wurde die Komponente „ConfigMan” entworfen. Sie definiert zwei Klassen `ConfigMan` und `Configure`, die das Konfigurationsmanagement bereitstellen. Das Parsen der einzelnen Konfigurationszuweisungen (vgl. Abschnitt 3.3.4) wird von den Klassen genauso übernommen, wie der Aufruf des entsprechenden modulspezifischen Konfigurationshandlers. Der Handler führt dann die gewünschte mit der Konfigurationszuweisung assoziierte Wertebelegung durch. Um eine korrekte Konfigurierung zu erreichen, muß zunächst bei der Initialisierung eine Anmeldung der Module stattfinden, damit der Konfigurationsmanager weiß, welche konfigurierbaren Module im System existierenden und welche KonfigurationsIds (mit dem zugehörigen Typ) für jedes Modul existieren. Diese Anmeldung wird in den Konstruktoren der Klassen vorgenommen:

```
int main()
{
 Modul m;
 Configure cfg;

    cfg.addCfgSubModule( "Modul", &m );
    cfg.configure( "Modul.Sub1.Id = 500" );
...
}

class Modul : public ConfigMan
{
```

```
 Sub1 s;
  Modul() { addCfgSubModule( "Sub1", &s ); }
...
}

class Sub1: public ConfigMan
{
  Sub1()
   { addCfgEntry( "Id", Id, ArgumentType::INTEGER ); }

  cfgHandler( ConfigId cfgId, const Argument& )
  {
    switch( cfgID )
   { case Id: //Zuweisung
...
   }
  }
...
}
```

Die Funktionalität von ConfigMan wird über eine „public"-Vererbung an die jeweiligen Klassen weitergegeben. Über die Methoden `addCfgModule` und `addCfgEntry` kann jede Klasse ihre direkt untergeordneten Module mit ihrem Namen anmelden. Bei der Definition der KonfigurationsIds wird zu dem Id-Namen und dem Argumenttyp zusätzlich ein Id-Kode mit angegeben, anhand dessen der Konfigurationshandler die entsprechende Anweisung identifiziert und die zugehörige Zuweisung ausführen kann (siehe switch-Anweisung). Das Einlesen der Konfigurationsanweisungen geschieht in der Funktion „main". Die Klasse `Configure` steuert von hier aus den Ablauf der Konfiguration. Dem beim Aufruf der Methode `configure` angegebene

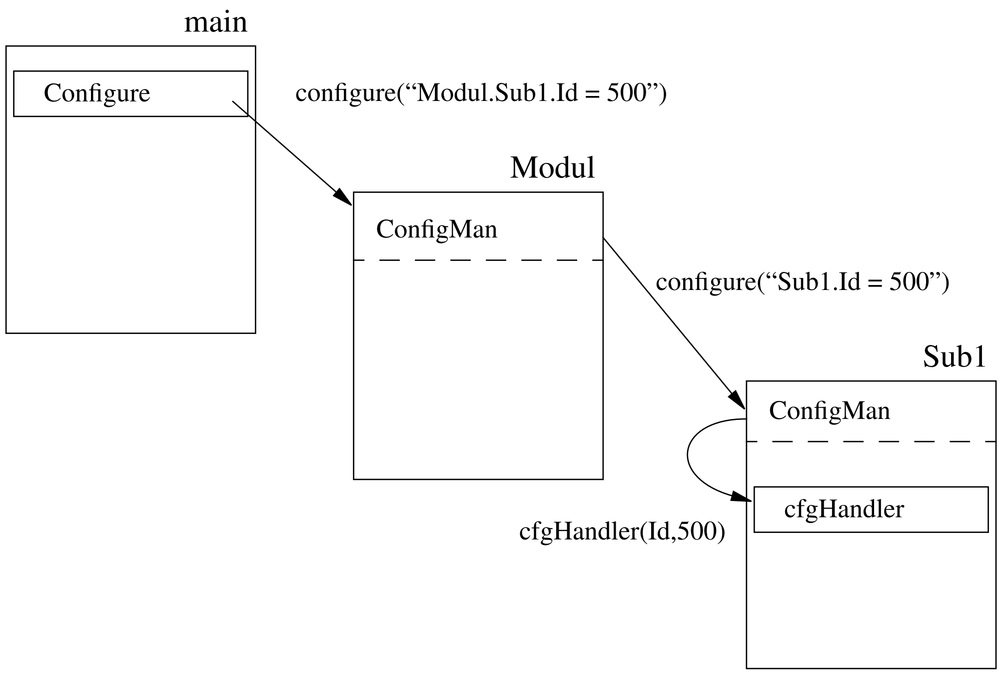


Abbildung 3-7: Ausführung einer Konfigurationsanweisung

Anweisungszeile wird geparst und entsprechend der darin enthaltenen Angaben an das zuständige Modul weitergereicht (siehe Abb. 3-7). Hier wird schließlich der Konfigurationshandler aufgerufen, der für eine Ausführung der Anweisung sorgt.

Zusätzlich besteht die Möglichkeit in jedem Modul eine Prüfroutine einzubinden, mit der die eingestellte Konfiguration abschließend auf ihre Konsistenz hin überprüft werden kann. Die Routine muß den Namen `chkConfiguration` tragen. Aus dem Hauptprogramm heraus kann somit durch den Aufruf der `check`-Methode der `Configure`-Klasse die Konsistenzüberprüfung aller angemeldeten Module angestoßen werden. Unter Konsistenz wird hierbei ein Zustand verstanden, in dem keine widersprüchlichen oder ungültigen Wertebelegungen (bezüglich der Konfiguration) vorliegen. Der Konsistenzbegriff ist also modulabhängig.

Komponente „WebPage"

Die Komponente WebPage enthält die Datenstrukturen für die Verwaltung der Seitennamen (URLs) und den mit den einzelnen Seiten assoziierten Informationen.

Da im Verlauf einer Simulation häufig auf die Seiteninformationen zugegriffen wird, muß das Design der Datenstrukturen und Zugriffsfunktionen besonders deren Effizienz gewährleisten. Die WebPage-Komponente arbeitet, um diesen effizienten Zugriff zu erreichen, mit sog. Pagehandles (siehe auch Abb. 3-8). Im Gegensatz zu den

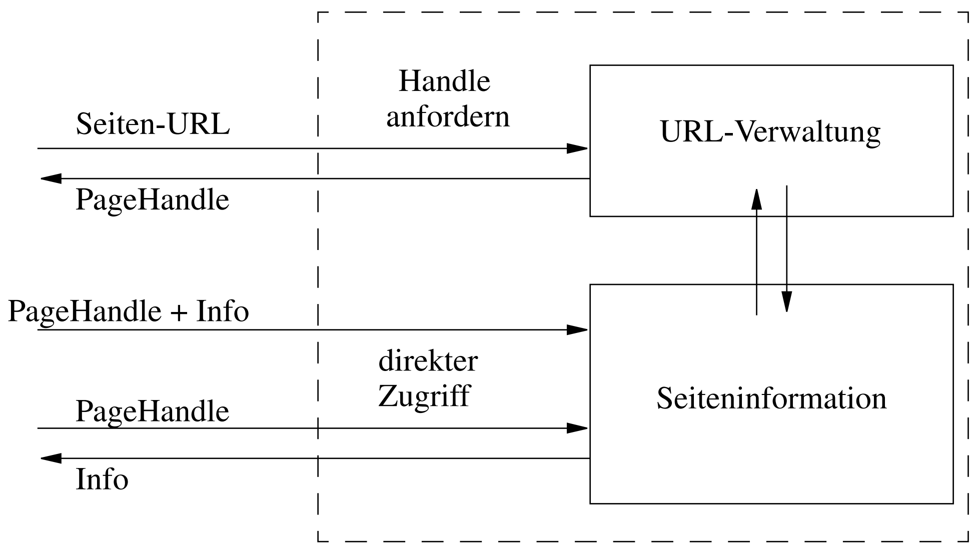


Abbildung 3-8: Zugriff auf WWW-Seiteninformationen

Handles wie sie in z.B. [STR92] beschrieben werden und wo durch sie eine Aufteilung der Objektrepräsentation in einen größeren, die eigentlichen Daten enthaltenden Teil und in einen kleinen, leicht zu kopierenden und den Zugriff auf die Daten erlaubenden Teil realisiert wird, sind die Pagehandles kein Teil der seiteninformation-speichernden Datenstruktur. Vielmehr spielen PageHandles die Rolle eines Zugriffsschlüssels, mit dem Informationen mit konstanter Komplexität (O(1)), unabhängig von der Anzahl der gespeicherten Seiten, gelesen und gespeichert werden können. Wie Abb. 3-8 zeigt, muß zunächst in einem ersten Schritt ein Pagehandle von der URL-Verwaltung angefordert werden. Mit diesem Handle ist dann in einem zweiten Schritt der Zugriff auf die gespeicherten Daten möglich (sowohl lesend als auch schreibend). Für die einzelnen Seiten werden folgende Informationen gespeichert:

- Größe
- Anzahl der Referenzen auf die Seite

- Anzahl der Cache-Treffer
- Insgesamt erbrachter Zeitvorteil durch Caching
- Letztes Änderungsdatum

Allgemein für Seitenzugriffe wird die Verwaltung folgender Werte unterstützt:

- Gesamtzahl der Zugriffe
- Gesamtzahl der Cache-Treffer bei Seitenzugriffen
- Anzahl der veralteten aus dem Cache geladenen Seiten
- Anzahl der 304-Server-Replys (Bestätigungen der Seitenaktualität)
- Byte-Hit-Rate

Komponente „ServerSite”

Die ServerSite-Komponente arbeitet nach demselben Prinzip wie die WebPage-Komponente. Auch hier werden Serverhandles für den Zugriff auf serverspezifische Informationen verwendet. Das Handle-Konzept bietet dabei einen idealen Ansatzpunkt, um die Zusammenfassung von Servern einer oder mehrerer Domänen (vgl. Abschnitt 3.3.2) transparent für das übrige Simulationssystem zu realisieren. Wird ein Serverhandle vor dem ersten Zugriff für einen Server angefordert, der in einer der angegebenen Domänen liegt, gibt die URL-Verwaltung einfach ein Handle für den entsprechenden dafür eingerichteten Domain-Server aus. Alle ausgegebenen Handles für Server aus dieser Domäne sind somit gleich und die Datenzugriffe können ohne weitere Überprüfungen ablaufen, Informationen werden nur für den Domain-Server gelesen bzw. gespeichert. Abb. 3-9 verdeutlicht diese Funktionsweise noch einmal. In

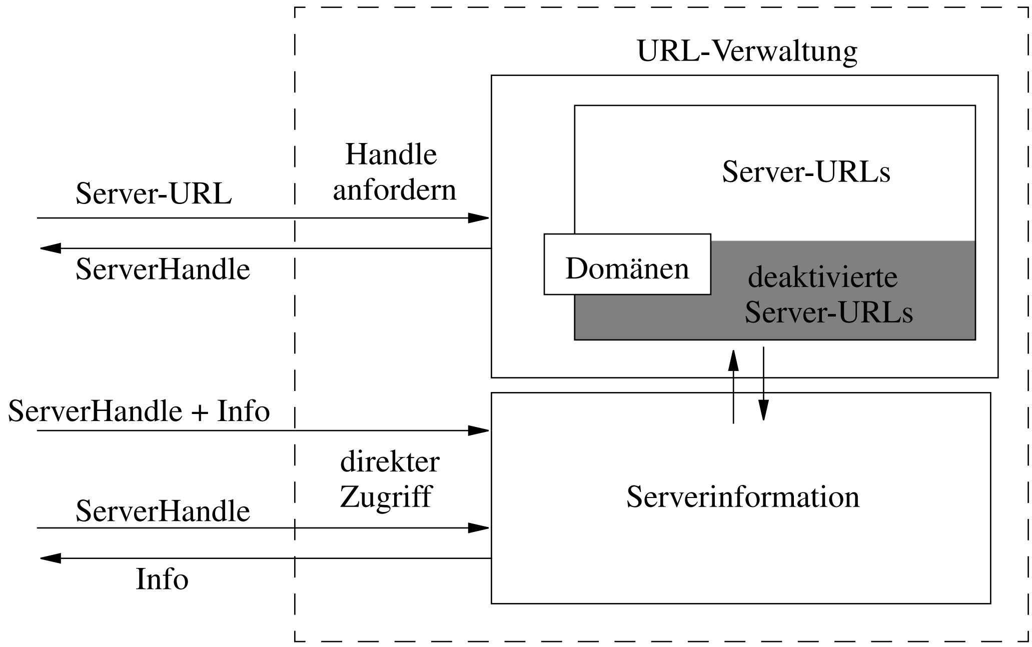


Abbildung 3-9: Zugriff auf WWW-Serverinformationen

Schritt 1 wird ein Serverhandle angefordert. Zu Domain-Servern zusammengefaßte Server werden in dem Bild als deaktiviert markiert. Die Zugriffe auf Seiteninformationen laufen dann nach erhaltenem Serverhandle identisch zu denen in der Komponente WebPage ab.

Für die WWW-Server können folgende Informationen gespeichert werden:

- Gesamtzugriffe auf den Server
- Cache-Hits bei Zugriffen auf den Server
- Entfernung in Hops (wird noch nicht unterstützt)

Als allgemeine Informationen werden geführt:

- Gesamtzahl der Serverzugriffe
- Gesamtzahl der von den Servern übertragenen Bytes

Komponente „Timing"

Verwaltung der während des Simulationsverlaufs benötigten Übertragungsprofile ist die zentrale Aufgabe dieser Komponente. Die Komponenten zur Realisierung verschiedenen Schätzverfahren (CProfile, AverageProfile, StraightLineProfile und SimpleStraightLineProfile) sind der Timing-Komponente untergeordnet. Verfahrensspezifische Aufgaben, wie z.B. die Berechnung von Übertragungsprofilen oder die Schätzung von Übertragungszeiten werden in diesen Komponenten übernommen. Die Timing-Komponente bietet somit eine Aufrufschnittstelle auf einem höheren Abstraktionsniveau – Aufrufe auf diesem Niveau „kennen" Serverhandles und arbeiten mit absoluten Datumsangaben (im UTC-Format), während auf dem darunterliegenden Abstraktionsniveau nur noch mit Tageszeiten zwischen 0 und 24 Uhr gearbeitet wird und hier der Begriff Serverhandle nicht mehr bekannt ist. Die Timing-Komponente muß also zusätzlich zur Auflösung der Zuordnung Server/Profil bzw. Server/Übertragungszeiten auch für eine Anpassung der unterschiedlichen Zeitsichten sorgen. An Abb. 3-10 soll der Aufbau der Timing-Komponente erläutert werden: in der Mitte sind die zwei Speichereinheiten für Übertragungszeiten und Übertragungsprofile zu erkennen. Für einzelne Server sind hier die aus den Logfiles entnommenen Übertragungszeiten bzw. die berechneten oder geladenen Übertragungsprofile gespeichert. Anfragen (auf der linken Seite durch den Pfeil symbolisiert) werden ggf. von den profileverwaltenden Klassen mit entsprechend angepaßten Parametern übernommen. Die Anpassung der Parameter und der Aufruf der untergeordneten Methoden fällt in den Aufgabenbereich der Koordinationseinheit. Die einheitlich durch die CProfile-Komponente festgelegte Schnittstellendefinition sorgt dafür, daß die verschiedenen Profilearten beliebig jedem Server zugeordnet werden können (sowohl die Art als auch die gewählte Tageszeitauflösung der Profile kann ohne Beschränkung variieren).

Auf dem Abstraktionsniveau der Timing-Komponente werden folgende Dienste angeboten:

- Eine Übertragungszeit für einen bestimmten Server eintragen (in Abhängigkeit des Datums und der Seitengröße).
- Die für ein bestimmten Server gespeicherten Übertagungszeiten ausgeben.
- Ein Profile für einen bestimmten Server laden, speichern oder berechnen.
- Visualisierungsdaten für ein bestimmtes Serverprofil ausgeben.
- Fehlerverteilung für alle Profile anhand der gespeicherten Übertragungszeiten berechnen.

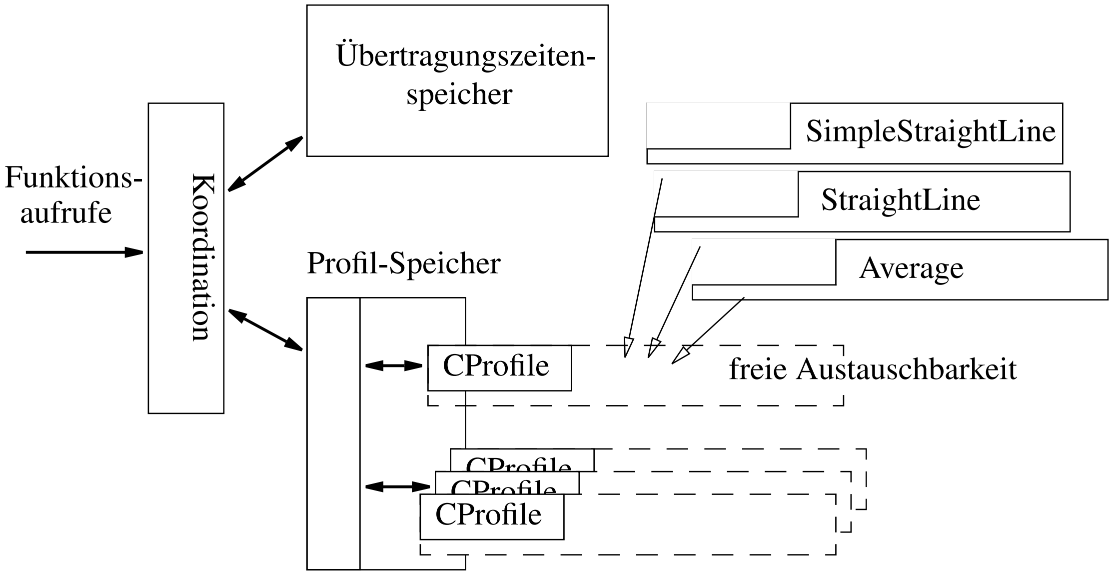


Abbildung 3-10: Aufbau der Timing-Komponente

- Durchschnittlichen Schätzfehler bezogen auf die Seitengröße für alle Profile berechnen.
- Übertragungszeit einer Seite mit angegebener Größe und angegebenen Zeitpunkt für einen bestimmten Server schätzen.

Auf dem Abstraktionsniveau der CProfile-Komponente sieht das Dienstangebot folgendermaßen aus:

- Profile laden, speichern oder aus einer Menge von Übertragungszeiten berechnen.
- Profildaten für die Visualisierung ausgeben.
- Übertragungszeit einer Seite mit angegebener Größe zu einer bestimmten Tageszeit schätzen.

Komponente „Cache"

Diese Komponente definiert die Schnittstelle für alle Module, die Caching-Verfahren implementieren. Wie auch schon in der Timing-Komponente, ist damit die volle Austauschbarkeit der verschiedenen Komponenten gewährleistet, ohne daß Änderungen bei der Initialisierung oder Nutzung der jeweiligen Klassen nötig wird. Für die Implementierung der einzelnen Verfahren selbst wird gleichfalls eine Menge von unterstützenden Methoden bereitgestellt. Diese umfassen auch Methoden, die für die Verwaltung eines eventuell eingerichteten Bereichs für replizierte Seiten zuständig sind. Die Schnittstelle bietet folgende Dienste:

- Initialisierung mit einer Menge von Cache-Parametern. Diese sind maximale Cachegröße, minimale Seitengröße, prozentualer Anteil des Caches in dem replizierte Seiten verwaltet werden sollen, Mindest-Zeitgewinn für einzulagerne Seiten und TTL-Faktor.

- Eine Anfrage an den Cache auswerten und über die vom jeweiligen Caching-Verfahren durchgeführte Aktivitäten buchführen (nur die Signatur des Aufrufs wird vorgeschrieben, eine Implementierung kann nur durch die einzelnen Verfahren selbst geschehen).
- Testfunktion: ist eine Seite zu groß für die Einlagerung in den Cache?
- Testfunktion: ist genug Platz im Cache für eine Einlagerung einer best. Seite?
- Testfunktion: ist eine Seite nicht mehr gültig (TTL-Zeitfenster zu Ende)?
- Hilfsfunktion: neues Expiry-Datum aus aktuellem Datum und letztem Änderungsdatum errechnen.

Management-Funktionalität für replizierte Seiten:

- Replizierten Bereich mit einer Menge von vorgegebenen Seiten laden. Die Seitenliste ist dabei aus einer Datei einzulesen.
- Einzelne Seiten in den replizierten Bereich aufnehmen.
- Einzelne Seiten aus dem replizierten Bereich löschen.
- Testfunktion: wird eine bestimmte Seite derzeit im replizierten Bereich gehalten?

Komponente „RandomCache", „TTLCache", „LRUCache" und „LCPCache"

Dieser Abschnitt erläutert einige Implementierungsaspekte der Caching-Verfahren Random, LRU, LCP und TTL. Die ersten drei Verfahren sind in Kapitel 2.3 auf Seite 7 schon zur Sprache gekommen. Lediglich das TTL-Verfahren bedarf einer weiteren Erwähnung: TTL steht bekanntlich für "Time To Live" und beschreibt eine Strategie zur Konsistenzerhaltung von Seiten in einem WWW-Cache. Das Caching-Verfahren trägt denselben Namen, da hier das vom TTL-Verfahren berechnete Expiry-Datum maßgeblich ist für die Auslagerung von Seiten verantwortlich ist. Ist nämlich für eine Seiteneinlagerung nicht mehr genug Platz im Cache vorhanden, werden solange ungültig gewordene Seiten, d.h. Seiten mit überschrittenem Expiry-Datum, daraus verdrängt, bis genügend Platz geschaffen wurde. Existieren keine ungültigen Seiten mehr im Cache und ist trotzdem nicht genügend Platz vorhanden wird *keine* Einlagerung vorgenommen. Das Verfahren wurde in die Tests mit einbezogen, um den Einfluß des TTL-Faktors genauer ohne Beeinflussung einer bestimmten Verdrängungsstrategie untersuchen zu können.

Zu den verwendeten Datenstrukturen:
Random, LRU und TTL sind in der Verwaltung der Daten fast identisch und sollen deshalb auch zusammen behandelt werden. Für das LCP-Verfahren sind dazu einige Erweiterungen notwendig, es soll deshalb im Anschluß daran separat betrachtet werden.

Die drei Verfahren Random, LRU und TTL speichern für jede im Cache gehaltene Seite ein eigenes Informationsfeld (siehe Abb. 3-11). Dieses Informationsfeld enthält das bei einer Neueinlagerung berechnete Expiry-Datum und den zu der jeweiligen Seite gehörenden Pagehandle. Alle Felder werden in einer Liste verwaltet. Löschen eines Elements ist gleichbedeutend mit der Auslagerung einer Seite und analog bedeutet das Anfügen eines neuen Elements die Einlagerung einer neuen Seite in den Cache. Über den Pagehandle ist der Zugriff auf die auch von anderen Komponenten innerhalb des Simulators benötigten Informationen wie Seitengröße oder letztes Änderungsda-

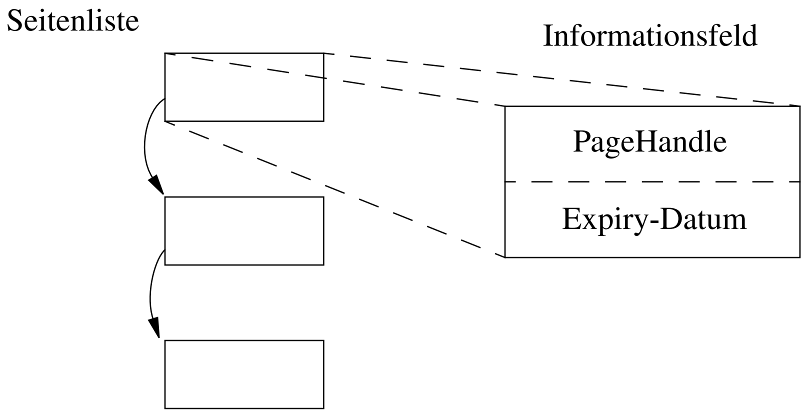


Abbildung 3-11: Cache-Datenstrukturen

tum möglich. So wird eine doppelte Speicherung vermieden und die vorhandenen effizienten Zugriffsmöglichkeiten können genutzt werden.

Mit Hilfe der Liste läßt sich nun durch Anhängen oder Löschen eines Elements leicht eine Ein- bzw. Auslagerung realisieren. Das TTL-Verfahren durchsucht bei einer Suche nach einem Verdrängungskandidaten diese Liste, bis ein entsprechendes Informationsfeld mit abgelaufenem Expiry-Datum gefunden wird, und löscht das zugehörige Element (Informationsfeld). Beim Random-Verfahren wird das zu löschende Element zufällig bestimmt, der Löschvorgang selbst bleibt unverändert. Die Information über die letzten Seitenreferenzzeitpunkte, die das LRU-Verfahren benötigt, brauchen nicht extra gespeichert zu werden. Sie können implizit in die Speicherungsreihenfolge der Seitenliste eingeflochten werden: Bei jeder Referenz auf eine Seite wird dazu das zugehörige Listenelement an den Anfang der Liste verschoben. Das Element bzw. die entsprechende Seite am Ende der Liste ist somit automatisch das am längsten nicht mehr referenzierte Element und damit der Löschkandidat im Falle einer Verdrängung.

Für das LCP-Verfahren müssen an dieser Struktur einige Erweiterungen vorgenommen werden: Zunächst sind für den Puffer und den privilegierten Bereich jeweils eine eigene Liste notwendig. Die zu den Seiten gehörenden Informationsfelder sind nicht mehr in den Listen selbst gespeichert, sondern werden separat angelegt. In den Listen wird dann mit Referenzen auf die Informationsfelder gearbeitet. Zusätzlich zum Pagehandle und dem Expiry-Datum enthalten sie Informationen darüber, wann der letzte Zugriff stattfand und welcher Gewinn in dem aktuellen Zeitfenster zu verbuchen ist. Aufgrund dieses Gewinns sind die Einträge der Privilegierten-Bereichs-Liste sortiert, die Elemente der Puffer-Liste sind in LRU-Reihenfolge angeordnet.

Für die Realisierung des verschiebbaren Zeitfensters wird weiterhin eine doppelt verkettete Liste der vergangenen Referenzreihenfolge benötigt (eine sog. History-Liste). Für jede Referenz muß der Zeitpunkt und der dann jeweils erzielte Gewinn inklusive einem Verweis auf die referenzierte Seite notiert werden. Der Gewinn muß später bei der Verschiebung des Zeitfensters zu den summierten Gewinnen der betroffenen Seite addiert bzw. subtrahiert werden. Solange nur Seiten aus dem Puffer verdrängt werden müssen, ist der Ablauf identisch zu dem des LRU-Verfahrens: Das Element am Ende der Liste wird entfernt und das neue am Anfang eingefügt. Wird die Einlagerung einer Pufferseite in den privilegierten Bereich überprüft, muß ggf. zunächst das Zeitfenster angepaßt werden (vgl. Abschitt 2.3.4 auf Seite 20). Dazu wird der Anfang des Zeitfensters auf den Zeitpunkt (Eintrag in der History-Liste) gesetzt,

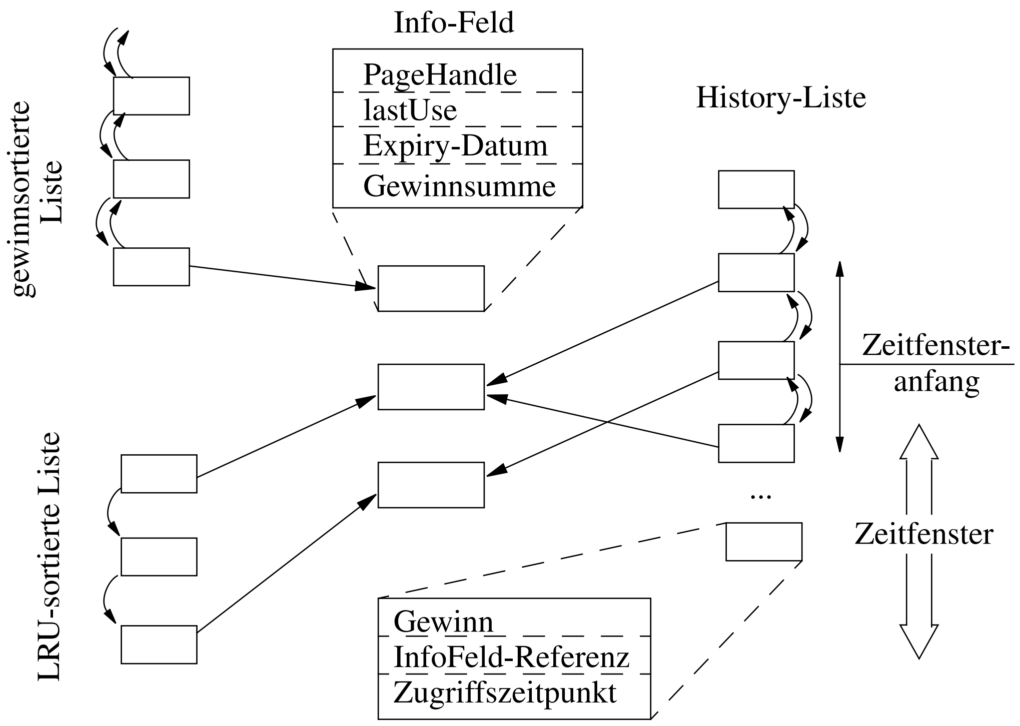


Abbildung 3-12: LCP-Datenstrukturen

der der am weitesten in der Vergangenheit liegende Referenzierungszeitpunkt einer Seite aus dem privilegierten Bereich oder der aktuell referenzierten Seite ist. Während der Verschiebung werden u.U. mehrere History-Einträge passiert. Geschieht dies bei einer Verschiebung in Richtung Gegenwart, werden die an diesen Zeitpunkten gemachten Gewinne bei der entsprechenden Seiteninformation subtrahiert und die Privilegierte-Bereichs-Liste aktualisiert. Bei einer Verschiebung in Richtung Vergangenheit, werden die einzelnen Gewinne addiert. Im ersten Fall wird das Zeitfenster verkleinert, im zweiten vergrößert. Je nachdem, ob eine Einlagerung in den privilegierten Bereich erfolgen kann oder nicht, werden die Listen des Puffer- und des privilegierten Bereichs abschließend aktualisiert.

Bemerkung: Die Einträge der History-Liste können bis zu dem am weitesten in der Vergangenheit liegenden Referenzierungszeitpunkt einer eingelagerten Seite gelöscht werden. Die Informationsfelder für die eingelagerten Seiten können endgültig gelöscht werden, falls keine Referenzen mehr auf sie verweisen.

Komponente „LogFiles"

Die Aufgabe der Logfile-Komponente ist es, eine einheitliche Sicht auf Proxy-, Cache- und LastModified-Logfiles zu realisieren. Dazu werden die drei verschiedenen Logfiletypen zu einem einzigen logischen Logfile zusammengefaßt.

Die Einträge, die dann aus dem neu entstandenen Logfile gelesen werden, enthalten alle relevanten, vorher in den alten Logfiles gespeicherten Informationen. Die Details, die bei einer solchen Zusammenführung beachtet werden müssen, sind in der Komponente gekapselt. Das Lesen von Einträgen aus den Light-Logfiles und dem neuen logischen Logfile kann in gleicher, einheitlicher Weise geschehen: Ein Eintrag nach dem anderen wird solange aus den Logfiles gelesen, bis keine weiten Einträge mehr verfügbar sind. Die für die Anfragegenerierung an einen Cache wichtige Umsortierung der

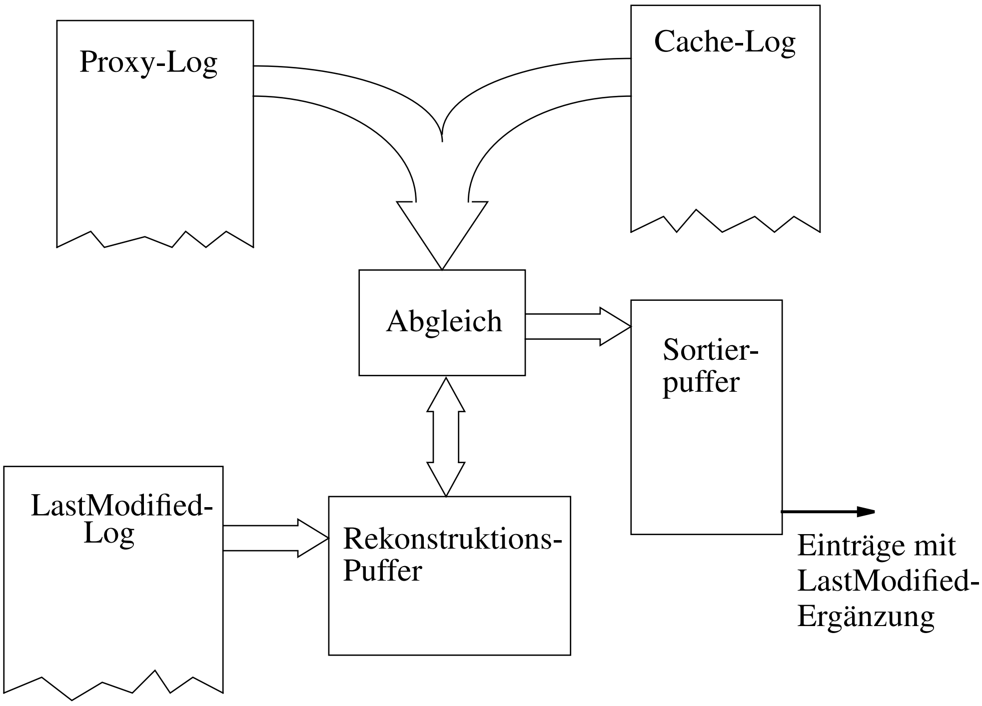


Abbildung 3-13: Zusammenführung der Logfiles

Einträge ist ebenfalls in die LogFiles-Komponente integriert worden (die Einträge in den vorliegenden Logfiles sind in der Reihenfolge sortiert, in der die zugehörigen Anfragen beendet wurden; für ein Cachetest ist es aber notwendig, die Reihenfolge zu verwenden, in der die Anfragen bei dem Proxy-Server eingetroffen sind).

Die Zusammenführung der Logfiles erfolgt in drei Schritten (vgl. Abb. 3-13): Schritt Nr. 1 führt Proxy- und Cache-Logfiles zusammen. Die Einträge beider Logfiles werden hierbei auf der Basis der Zugriffsendezeiten in eine schwache[1] zeitliche Reihenfolge gebracht. Im zweiten Schritt erfolgt dann Eintrag für Eintrag ein Abgleich mit den LastModified-Logfiles. Dieser ist aber nicht direkt möglich, da sich ein Cache-Zugriff in den LastModified-Logfiles u.U. auf mehrere z.T. weiter auseinanderliegende Einträge verteilt (besonders in Zeiten hoher Aktivität ist dies häufig der Fall). Aus diesem Grund muß aus den LastModified-Logfiles der Ablauf für jeden Cachezugriff rekonstruiert werden. Für jeden Cachezugriff können auf diese Weise Beendigungszeitpunkt und LastModified-Datum der referenzierten Seite in einem Rekonstruktionspuffer gespeichert werden. Ein LastModified-Datum ist allerdings nur Verfügbar, falls die Seite bei dem Zugriff aus dem Cache geladen oder in den Cache eingelagert wurde. Die zusammengeführten Einträge der Proxy- bzw. Cache-Logfiles werden nun in einem zweiten Arbeitsschritt mit den rekonstruierten Zugriffen abgeglichen, d.h. für jeden Eintrag wird im Rekonstruktionspuffer das entsprechende Gegenstück gesucht, das Datum der letzten Änderung, falls ermittelbar, zum Eintrag hinzugefügt und an den Sortierpuffer weitergeleitet. Hier werden die Einträge gemäß ihres Anfangszeitpunktes umsortiert und können in dieser neuen Reihenfolge vom aufrufenden Programmteil nacheinander ausgelesen werden. Ein Sortierpuffer muß nun sicherstellen, daß neu in

1. Eine strikte Sortierung, bei der ausnahmslos alle Einträge gemäß ihres Endedatums angeordnet werden, wäre erstens zu aufwendig – ein zweiter Sortierpuffer müßte eingerichtet werden – und würde zweitens den Abgleich mit den LastModified-Logfiles erschweren.

den Puffer gelangende Einträge kein Anfangsdatum aufweisen, das jünger ist als das des zuletzt ausgegebenen Eintrags. Je länger die Dauer eines Zugriffs war, desto später gelangt der entsprechnde Eintrag in den Sortierpuffer und desto weiter nach vorne verschiebt sich seine Position im Puffer. In dem Sortierpuffer müssen also soviel Einträge gespeichert sein, daß für jeden neuen Eintrag, der in den Puffer gelangt, bereits andere bisher zurückgehaltene Einträge mit einem noch früheren Anfangszeitpunkt existieren.

Bemerkungen: Durch Aufzeichnungsfehler in den LastModified-Logfiles ist es oft nicht möglich Cachezugriffe zu rekonstruieren. In diesem Fall scheitert ein Abgleich, dem Eintrag aus dem Proxy- oder Cache-Logfile kann kein entsprechender Eintrag aus dem Rekonstruktionspuffer zugeordnet werden – für die in dem mit dem Eintrag festgehaltenen Zugriff referenzierten Seite muß also auf ein LastModified-Datum verzichtet werden. Insgesamt ergibt sich eine Erfogsquote von 80%, d.h. den restlichen 20% der Einträge konnte kein zugehöriges LastModified-Datum zugeordnet werden (in diesen Fällen wird ein Default-Alter von einem Monat angenommen).

Außerdem muß bei der Rekonstruktion darauf geachtet werden, daß der Zeitraum aus dem Einträge rekonstruiert werden übereinstimmt mit dem Zeitraum aus dem die Einträge der Proxy- bzw. Cache-Logfiles stammen, ansonsten wird ein Abgleich in jedem Fall scheitern.

4. Auswertung der Meßergebnisse

4.1 Überblick

Dieses Kapitel beschäftigt sich ausführlicher mit den Erkenntnissen, die mit Hilfe von CaSimiR gewonnen werden konnten. Abschnitt 4.2 wendet sich zunächst grundlegenden Beobachtungen zu. Dazu gehört ein Blick auf die Logfiles, die die Basis für die Simulation darstellen. Genauer wird ebenfalls auf das Referenzverhalten eingegangen, das aus den in den Logfiles gespeicherten Zugriffen abgeleitet werden kann. Abschnitt 4.2.3 ist dann der Untersuchung von Übertragungsbandbreiten gewidmet. Hier werden Fragen geklärt, wie: Wann ist die günstigste Tageszeit, um Seiten von bestimmten WWW-Servern zu laden? Welche Bandbreite kann maximal bei einer Verbindung z.B. zu Servern eines bestimmten Kontinents erwartet werden?

Abschnitt 4.3 wendet sich danach der Schätzung von Seitenübertragungszeiten zu. An einem Beispiel-Server werden die Stärken und Schwächen der einzelnen Modelle aufgezeigt, gefolgt von einer detaillierteren Fehleranalyse.

Den abschließenden Teil des Kapitels bildet die Untersuchung verschiedener Caching-Strategien. Unter Variation aller Cacheparameter werden u.a. die Strategien Random, LRU und LCP nach verschiedenen Gesichtspunkten beurteilt. Aussagen über die theoretischen Grenzen, die allen Verfahren gesetzt sind, fehlen ebensowenig wie eine Untersuchung der Verbesserungsmöglichkeiten durch eine Ergänzung durch Replikation.

4.2 Grundlageninformation

4.2.1 Logfiles

Dieser Abschnitt soll zunächst einmal einen Eindruck von der Größenordnung und der Qualität der zur Verfügung stehenden Daten geben. Die für die Monate Januar bis Mai '96 aufgezeichneten Logfiles sind zu diesem Zweck einer Untersuchung unterzogen worden, die sich mit der Anzahl der Einträge und den auftretenden Fehlern genauer beschäftigt hat. Die folgende Tabelle enthält für die o.g. Monate eine Aufstellung, wie viele Einträge insgesamt in jedem Monat in den jeweiligen Logfiles aufgezeichnet wurden. Weiterhin wird die Zahl der Einträge aufgeführt, die für eine Simulation oder Auswertung nicht verwendet werden können. Der Großteil dieser Einträge bilden solche, bei denen Fehler während der Bearbeitung der Anfrage auftraten und vom jeweiligen Server nicht bearbeitet werden konnten (zu erkennen an einem Server-Reply-Code mit dem Wert größer oder gleich 400). Die übrigen dieser nicht verwendbaren Einträge gehen auf Fehler bei der Logfile-Aufzeichnung zurück. Eine genauere Beschreibung dieser Fehler ist im Anschluß an die Tabelle zusammen mit einer Reihe von zusätzlichen Bemerkungen zur Logfilequalität zu finden.

Proxy-Logfiles	**Einträge**	**nicht verwendbar**	**ServerReply ≥ 400**	**sonstige Fehler**
Januar ‘96	624801	36400	36358	42
Februar ‘96	494627	27830	27717	113
März ‘96	436635	30430	30421	9
April ‘96	466197	25462	25453	9
Cache-Logfiles	**Einträge**	**nicht verwendbar**	**ServerReply ≥ 400**	**sonstige Fehler**
Januar ‘96	149280	1	0	1
Februar ‘96	124076	0	0	0
März ‘96	118830	3	0	3
April ‘96	121780	0	0	0
LastModified-Logfiles[1]	**Einträge**	**nicht verwendbar**	**ServerReply ≥ 400**	**sonstige Fehler**
April ‘96	550891	2142	-	-
Mai ‘96	754886	2937	-	-
Juni ‘96	710553	2830	-	-
Juli ‘96	710553	2830	-	-

1. LastModified-Logfiles sind erst ab April ‘96 verfügbar

Die folgenden Fehlerarten wurden in den Logfiles festgestellt:

- Relative Pfadangaben

```
xx.informatik.uni-kl.de - - [02/Jan/1996:14:35:28 +0100]
        “GET http://../.netscape-bookmarks.html HTTP/1.0”
        404 368 +1:111833 (Mozilla/1.1N (X11; I; SunOS 4.1.3))
```

- Einträge mit Server-Reply 400

```
xx.zbt.uni-kl.de - - [02/Jan/1996:08:38:17 +0100]
        “GET http://www.moneymaxx.de/ HTTP/1.0” 500 315
        +81:250978 (Mozilla/2.0b3 (Win95; I))
```

- Fehlerhafte Datumsangaben

```
xx.rhrk.uni-kl.de - - [11/Janinformatik.uni-kl.de - - [12/Jan
        /1996:01:04:24 +0100] “GET http://www.cs.cornell.edu
        /Info/People/ckline/humor/msg00330.html HTTP/1.0” 200
        4105 +3:488005 (Mozilla/1.1N (X11; I))
```

- Angaben der Übertragungszeit mit vorangestelltem „+-”

```
xx.uni-kl.de - - [03/Jan/1996:16:03:01 +0100]
        “GET http://www.meridian.co.uk/ara/ara.gif HTTP/1.0”
        200 3541 +-22:610823 (Mozilla/2.0b3 (X11; I; SunOS))
```

- Fehlende Angaben zum Server-Reply-Code und zur Seitengröße

```
xx.rhrk.uni-kl.de - - [02/Jan/1996:09:15:59 +0100]
        “GET http://www.geopages.com/Hollywood/1818/covers.gif
        HTTP/1.0” - - +2:66546 (Mozilla/1.12 (X11; I; AIX 2))
```

- Einträge im LastModified-Logfile mit vorangestellten Steuercodes

Als weitere Besonderheiten sind folgende Punkte zu vermerken:

- Bei Zugriffen, die mit einem Server-Reply-Code von 304 beantwortet wurden, ist die Größe der übertragenen Seite mit 0 Bytes angegeben. In diesem Fall wird stattdessen für die Simulation mit einer Größe von 100 Bytes gearbeitet, was ungefähr der Größe des gesendeten HTTP-Headers entspricht, mit dem der Reply-Code gesendet wurde.
- Die Reihenfolge, in der die Einträge in den Logfiles aufgeführt sind, entspricht nicht immer der Reihenfolge, in der die entsprechenden Zugriffe auch beendet wurden.
- Die Einträge des LastModified-Logfiles für einen bestimmten Monat enthalten fälschlicherweise auch die Zugriffe des letzten Tages vom Vormonat. Entsprechend fehlen die Aufzeichnungen für den letzten Tag des aufzuzeichnenden Monats.
- Zusammenhängend mit dem letzten Punkt und einer unvollständigen Aufzeichnung der Cacheabläufe in den LastModified-Logfiles, ergibt sich bei ca. 80% der Einträge aus Proxy- und Cache-Logfile die Möglichkeit, für die beim Zugriff referenzierte Seite ein LastModified-Datum anzugeben. Für die restlichen 20% der Seiten ist dies nicht der Fall, da keine Beziehung zwischen den Logfiles hergestellt werden kann.
- Die von CaSimiR generierten Light-Logfiles enthalten sowohl die Angabe einer Cacheladezeit wie auch die einer Serverladezeit. Da eine der beiden Zeiten auf der Basis der gespeicherten Profile geschätzt wurde, ist es möglich, daß die Serverladezeit niedriger als die Cacheladezeit ausfällt. Solche Konstellationen können auftreten, falls der Cache durch viele gleichzeitige Anfragen überlastet wird. In solchen Fällen wäre ein Laden ohne Zuhilfenahme des Caches erfolgversprechender (für das Light-Logfile April '95 ist dies in 11.6% und bei den Mai-Logfiles in 10.7% der Einträge zu beobachten).

4.2.2 Referenzverhalten

Das Funktionsprinzip von Caching beruht auf Referenzlokalität, d.h. bestimmte Seiten werden zu einem Zeitpunkt häufig referenziert, andere selten oder gar nicht. Auch das Caching für WWW-Seiten beruht auf genau diesem Prinzip. Der folgende Abschnitt widmet sich aus diesem Grund einer qualitativen Untersuchung des Referenzverhaltens im World-Wide Web. Die Frage, ob eine Art von Referenzlokalität bei den Zugriffen vorhanden ist, soll hier geklärt werden.

Die Anfragen eines Monats (April '96) wurden daraufhin untersucht, wie viele Zugriffe auf welche WWW-Server bzw. welche WWW-Seiten aufgetreten sind. Die Server (bzw. Seiten) wurden danach entsprechend ihrer Beliebtheit geordnet und die Anzahl der Zugriffe in die Diagramme für Server bzw. Seiten eingetragen. Abb. 4-1 zeigt das Ergebnis für die referenzierten WWW-Server, das typisch für alle Monate ist. Das Gefälle zwischen häufig referenzierten und weniger häufig referenzierten Servern ist deutlich zu erkennen. Nur wenige hundert Server können den Hauptanteil der Zugriffe auf sich vereinigen, während die übrigen Server – insgesamt wurden 11000 Server referenziert – mehr oder weniger selten frequentiert waren. Dasselbe Bild zeigt sich bei den WWW-Seiten (Abb. 4-2). Mit dem Unterschied, daß die maximale Anzahl der Referenzen hier niedriger ausfällt (die Referenzen auf einen Server verteilen sich u.U. nun auf mehrere Seiten), ist ein ausgeprägterer Knick zu erkennen, an dem die Kurve steil abfallend in einen flachen Verlauf übergeht. Bei einer referenzierten Menge von insgesamt ca. 220000 verschiedenen WWW-Seiten ist offensichtlich eine noch

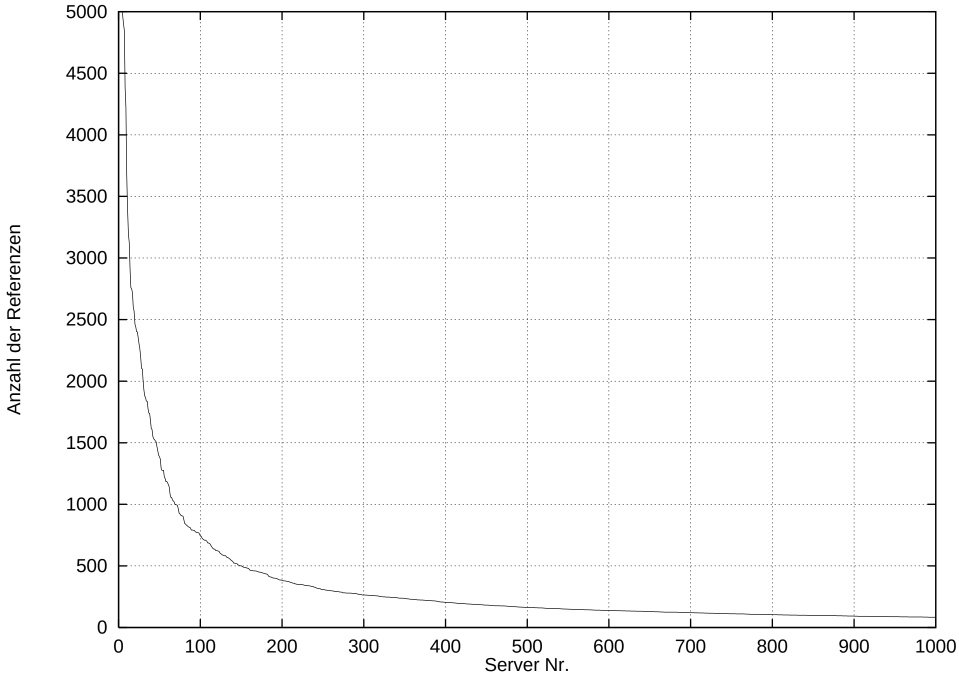


Abbildung 4-1: Referenzverhalten auf WWW-Server

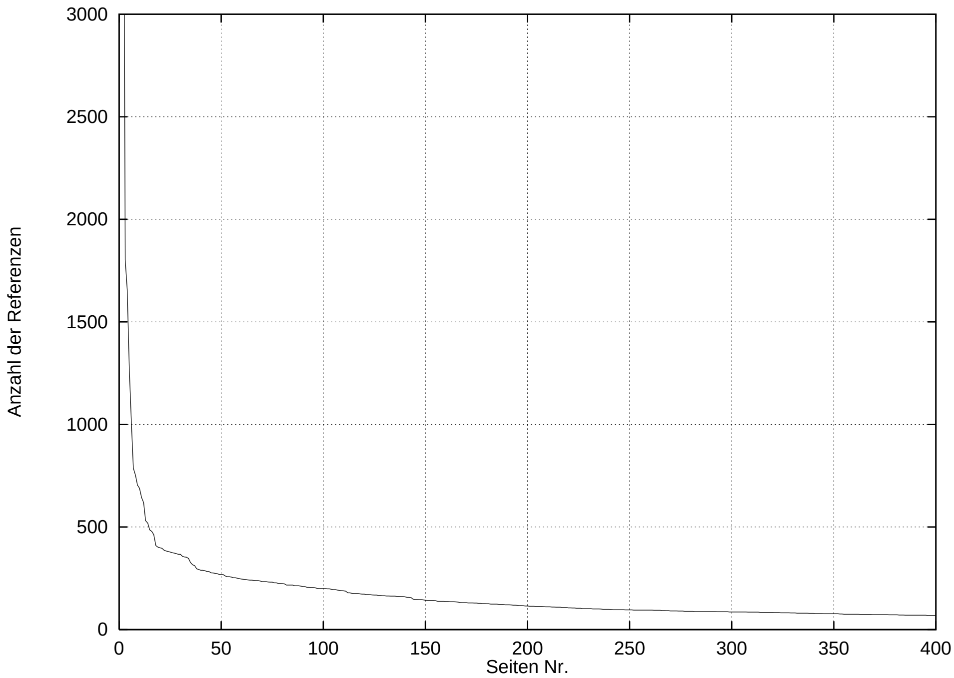


Abbildung 4-2: Referenzverhalten für WWW-Seiten

ausgeprägtere Referenzlokalität vorhanden, als dies schon für die Server zu beobachten war. Somit ist auch das World-Wide Web ein ausgezeichneter Ansatzpunkt, um Caching-Strategien nutzbringend einzusetzen.

4.2.3 Bandbreite

Zu verschiedenen Tageszeiten ist im World-Wide Web eine unterschiedliche Belastung zu erkennen. Dies macht sich deutlich bemerkbar an den Seitenladezeiten. Von weit entfernten Servern ist es oft nur außerhalb von Spitzenbelastungszeiten möglich, größere Seiten zu laden. Wie Abschnitt 2.4 bereits gezeigt hat, ist es besonders wichtig zu wissen, zu welchen Zeiten hohe Netzwerkaktivität herrscht und in welchen Zeiträumen die Belastung so gering ist, daß sich z.B. eine Aktualisierung replizierter Seiten lohnt.

Die Unterschiede in der zur Verfügung stehenden Bandbreite und die damit verbundenen unterschiedlichen Ladezeiten wurden deshalb auch für den Proxy-Server der Universität Kaiserslautern untersucht. Die folgenden Daten wurden in einem Zeitraum von drei Monaten erhoben (Februar bis April '96). Abb. 4-3 zeigt zunächst die gemessene Bandbreite zwischen dem Proxy-Server in Kaiserslautern und allen im Erhebungszeitraum referenzierten WWW-Servern aus aller Welt. Die Bandbreite wird dabei in Abhängigkeit von der Tageszeit (Ortszeit Kaiserslautern) abgetragen und stellt eine Mittelung über alle Tage des Meßzeitraums dar. Die Übertragungsraten werden aus Seitengrößen und Übertragungsdauer errechnet. Ferner wird davon ausgegangen, daß keine Schwankungen während der Übertragung aufgetreten sind. Eine genauere Unter-

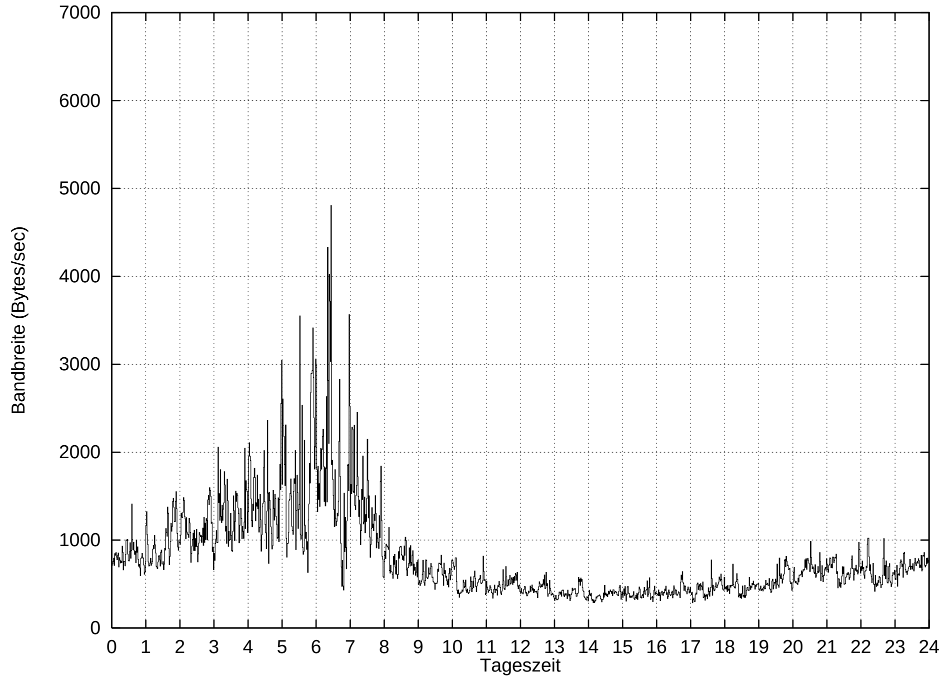


Abbildung 4-3: Tageszeitabhängige Übertragungsbandbreite

scheidung der referenzierten Server wird zunächst noch nicht vorgenommen, um eine allgemeine Tendenz im Verhältnis Tageszeit zu Bandbreite aufzeigen zu können.

Deutlich auffallend ist die stark schwankende Bandbreite im Verlauf eines Tages. Sie reicht von knapp 500 Bytes/Sek. in der Mittagszeit bis fast 5000 Bytes pro Sekunde in den frühen Morgenstunden. Der unruhige Verlauf der Kurve, besonders im Bereich zwischen 3 und 8 Uhr, deutet darauf hin, daß in diesem Bereich die Unterschiede zwischen den einzelnen Servern vergleichsweise hoch sind. Ein weiterer Grund dafür ist die geringe Aktivität, die in Abb. 4-6 ergänzend dargestellt wird. Die dort gezeigte Kurve beschreibt, wieviele Seiten zu einer bestimmten Tageszeit gerade von anderen Servern geladen wurden. Für das Zeitintervall, in der eine Seite von dem anbietenden Server geladen wird, erhöht sich die Aktivität um eins. Ist der Ladevorgang beendet, wird das Aktivitätsmaß entsprechend wieder um eins erniedrigt. Das Aktivitäten-Diagramm hält auf diese Weise alle Ladevorgänge fest, die in dem Erhebungszeitraum von drei Monaten stattfanden. Ein Aktivitätswert von 3000 zu einer bestimmten Tageszeit

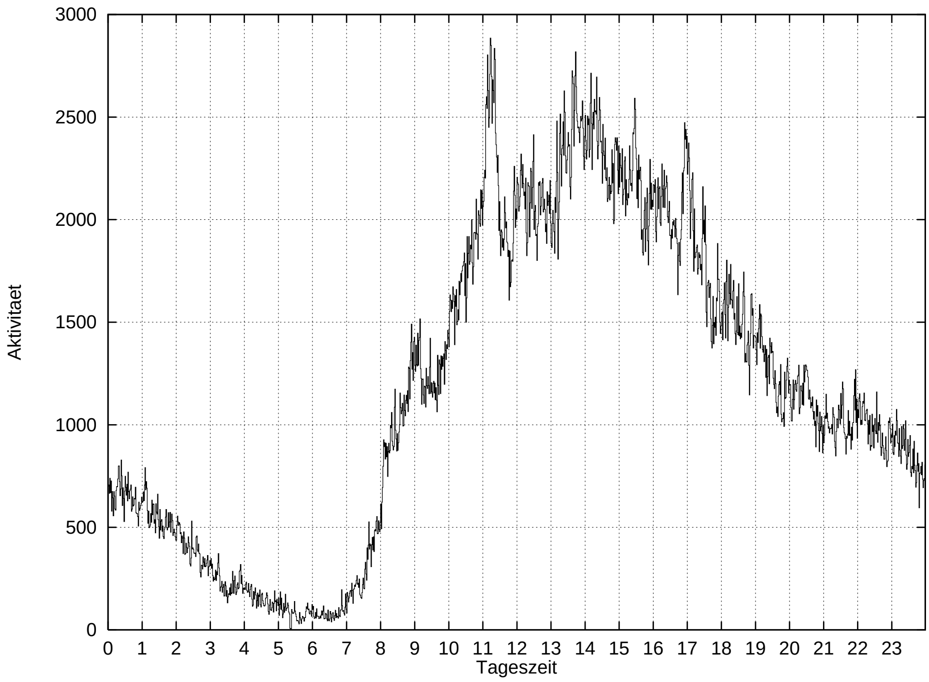


Abbildung 4-4: Tagesabhängige Aktivität zwischen Proxy- und WWW-Servern

bedeutet somit, daß innerhalb der drei Monate zu diesem Zeitpunkt insgesamt 3000 Seiten gerade geladen wurden. Eine hohe Aktivität ist somit gleichzeitig auch ein Maß für die Signifikanz der Bandbreiteninformationen. Je mehr Aktivität zu einem Zeitpunkt herrschte, desto mehr Übertragungen sind in die Berechnung der Bandbreite eingegangen und desto besser beschreibt das Bandbreitendiagramm die Situation zu diesem Zeitpunkt. Umgekehrt gilt: Je weniger Aktivität in einem bestimmten Zeitraum herrschte, desto stärker können sich zufällige Schwankungen der Übertragungsbandbreite im Diagramm manifestieren und desto höher ist die Varianz des Bandbreitenwerts.

Das Aktivitäten-Diagramm zeigt, daß der überwiegende Teil der Anfragen in den Zeitraum von 8 bis 21 Uhr fällt. Der Spitzenwert wird dabei in der Mittagszeit zwischen 11 und 12 Uhr bzw. 13 und 14:30 Uhr erreicht. In diesem Zeitraum sinkt die zur Verfügung stehende Bandbreite auf ihren niedrigsten Wert. In den frühen Morgenstunden dagegen sind die Aktivitäten deutlich reduziert, entsprechend hoch ist die zur Verfügung stehende Bandbreite. Der Spitzenwert von knapp 5000 Bytes pro Sekunde fällt in diesen Zeitraum, der sich deshalb besonders gut für eine Seitenaktualisierung eignen würde. Eine Aktualisierung könnte zum einen die höhere Bandbreite nutzen und zum anderen für eine gleichmäßigere Belastung des Netzes sorgen.

Die oben abgebildeten Diagramme erfassen die Verbindungen zu allen von der Universität aus adressierten WWW-Servern. Welche Unterschiede existieren aber nun zwischen den verschiedenen Servern? Mit den vielen verschiedenen Seiten, die potentiell repliziert oder normal im Cache gehalten werden können, müssen auch die Unterschiede zwischen den einzelnen Servern oder Servergruppen näher untersucht werden. Dazu wurde die Gesamtheit der referenzierten Server auf der Basis ihrer Domänenzugehörigkeit partitioniert. Die speziell für solche Aufgaben konzipierte Fähigkeit von CaSimiR, mehrere Server beliebig vorgegebener Domänen zusammenzufassen und als einzigen logischen Server zu behandeln, wurde hier genutzt. Die Auswahl der betrachteten Domänen orientierte sich bei dieser Untersuchung an der Häufigkeit, mit der Seiten aus den jeweiligen Domänen geladen wurden. Hier aufgeführt sind die Domänen `com`, `de` und `net`. Außerdem, um eine geographische Abhängigkeit zu untersuchen, wurden Server aus Nordamerika (USA, Kanada), Europa (Spanien, Frankreich, Italien, Niederlande, Dänemark, Norwegen, Finnland, Schweden, Polen und Rußland), Asien (Japan, Taiwan, Südkorea) und Australien (incl. Neuseeland) zu separaten Gruppen zusammengefaßt und jeweils die Bandbreiten ermittelt.

Zunächst zu den Domänen, aus denen ein Großteil aller Seiten geladen wurden: Die Abhängigkeit der Bandbreite von der Tageszeit ist bei den ersten beiden auf Seite 75 und Seite 76 abgebildeten Diagrammen sehr ähnlich. Beide dargestellten Diagramme weisen den schon von den obigen Kurven bekannten Verlauf auf. Während alle Server aus der `com`-Domäne in den USA angesiedelt sind, ist solch eine geographische Zuordnung bei den Servern der `net`-Domäne nicht mehr möglich. Dieser Umstand erklärt die deutlich höhere maximal erreichbare Bandbreite und die stärkeren Schwankungen. Abschwächend muß hinzugefügt werden, daß der Spitzenwert in einer Phase niedriger Aktivität verzeichnet wurde und deshalb die Signifikanz dieses Wertes entsprechend niedrig ist.

Die dritte Domäne, die hier erwähnt werden soll, ist die `de`-Domäne (vgl. Abbildung 4-10 auf Seite 76). Vergleichsweise nahegelegene Server aus Deutschland sind hier zusammengefaßt, was sich auf die durchgehend höhere Bandbreite niederschlägt. Die Schwankungen zwischen niedrigster (ca. 800 Bytes pro Sek.) und höchster gemessener Bandbreite (ca. 13000 Bytes pro Sek.) fällt weiterhin sehr deutlich aus, wenngleich der letztgenannte Spitzenwert wieder in einer Zeit niedriger Aktivität gemessen wurde und deshalb die Wahrscheinlichkeit entsprechend hoch ist, daß zufällig sehr gute Übertragungen in die Messung mit eingegangen sind. Nimmt man den Spitzenwert aus der Betrachtung heraus, ergibt sich eine nächtliche Übertragungsbandbreite, die zwischen 2000 und 3000 Bytes pro Sekunde schwankt. Insgesamt gesehen, bringt die Nachtzeit also nicht mehr die starken Vorteile, wie sie bei Servern aus der `com`- oder `net`-Domäne zu beobachten sind. Der Zeitraum, in dem die Bandbreite im nachmittäglichen Betrieb absinkt, ist nun schärfer begrenzt; zwischen ca. 11 und 18 Uhr ist dieser Abfall zu beobachten. Die insgesamt besseren Verbindungsverhältnisse können mit den geringeren zu überbrückenden Distanzen und damit auch der geringeren Zahl der Zwischenknoten (Hops), über die eine Nachricht laufen muß, erklärt werden. Eine Rolle spielt sicherlich aber auch die Netzwerktopologie. Übertragungen in die USA laufen über Verbindungen, die auch von anderen europäischen Ländern insbesondere am Tage verwendet werden. Zu diesen Zeiten ist das Datenaufkommen besonders hoch, entsprechend niedrig ist die Bandbreite. Nicht zuletzt die Aktivität an der Universität selbst beeinflußt in eingeschränktem Maße die zu Verfügung stehende Bandbreite.

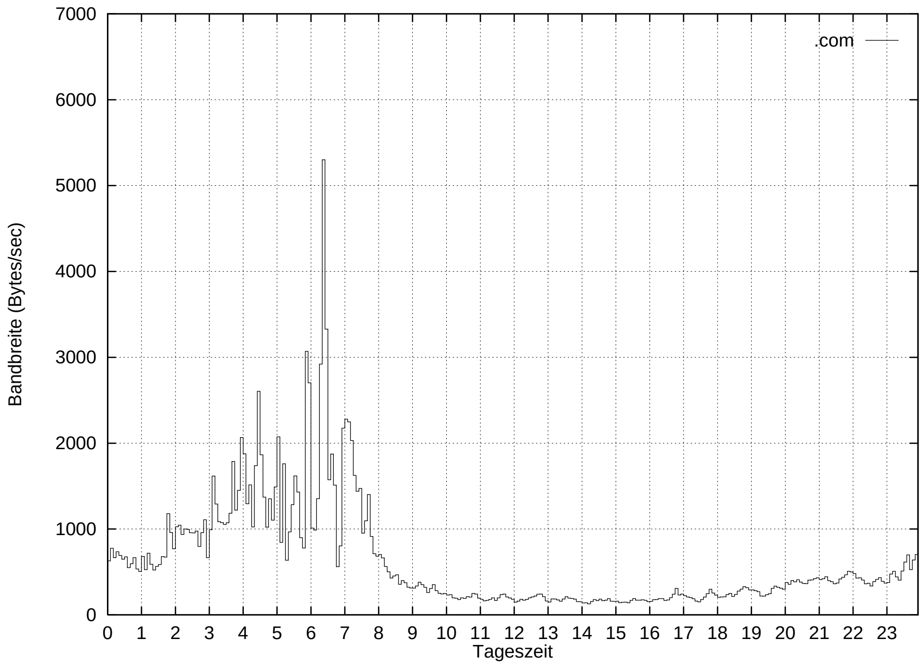


Abbildung 4-5: Gemessene Bandbreite für Server der `com`-Domäne

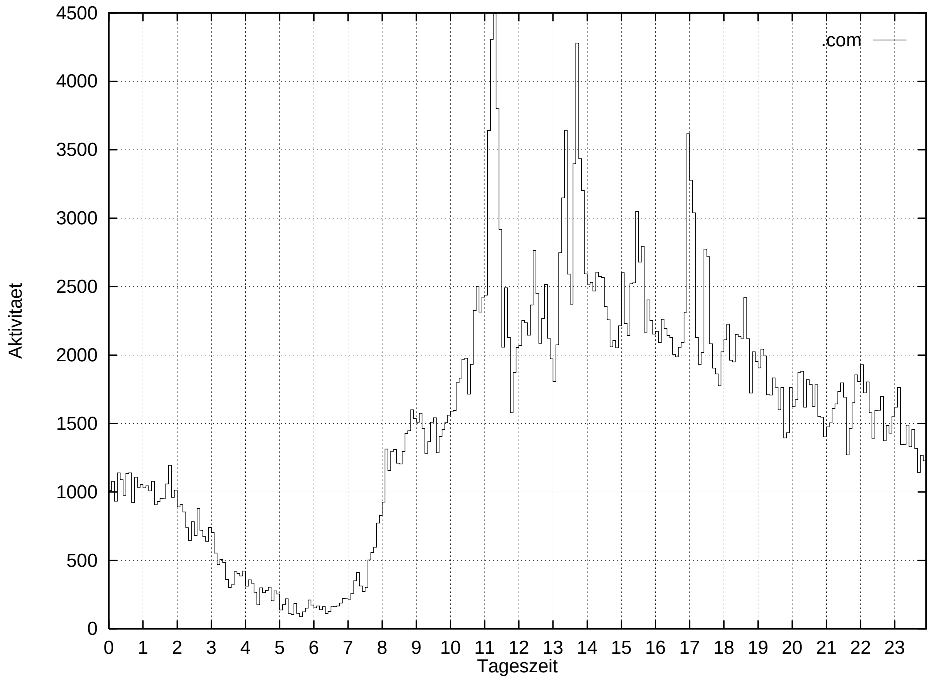


Abbildung 4-6: Aktivitäten-Diagramm zur `com`-Domäne

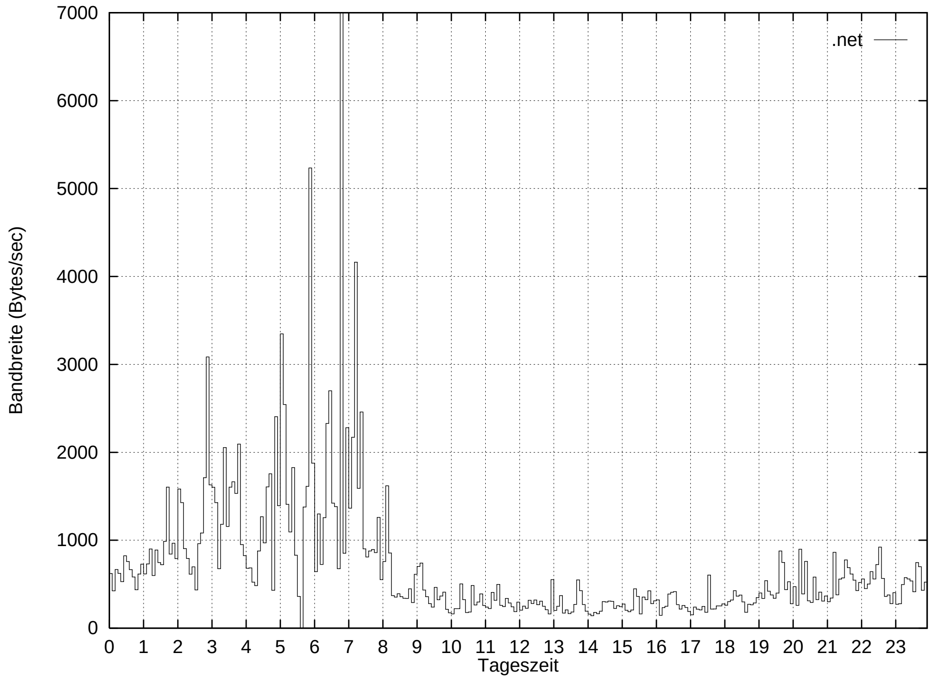


Abbildung 4-7: Gemessene Bandbreite für Server der `net`-Domäne

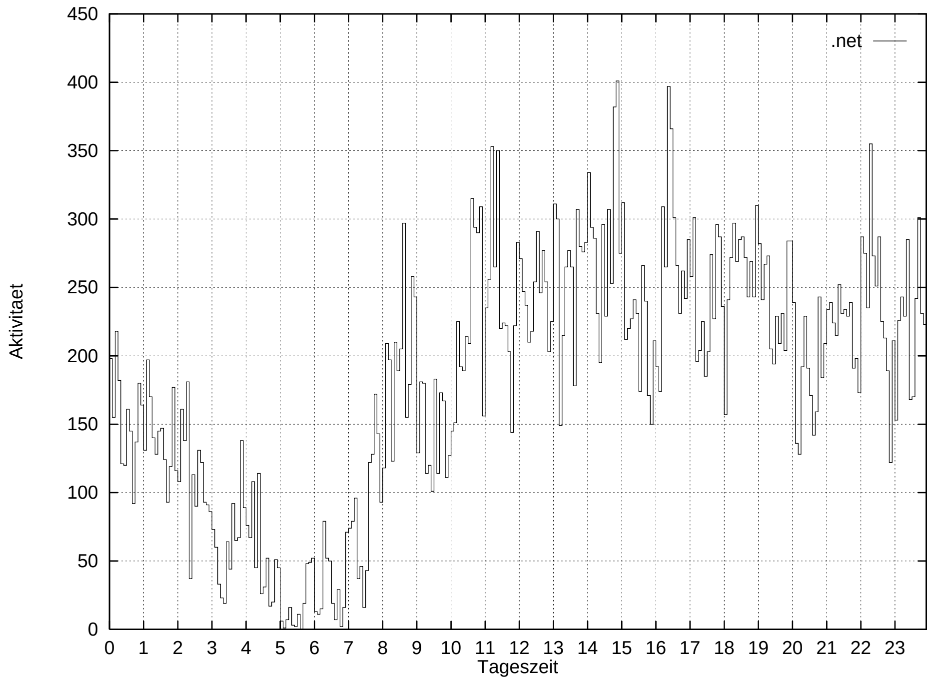


Abbildung 4-8: Aktivitäten-Diagramm zur `net`-Domäne

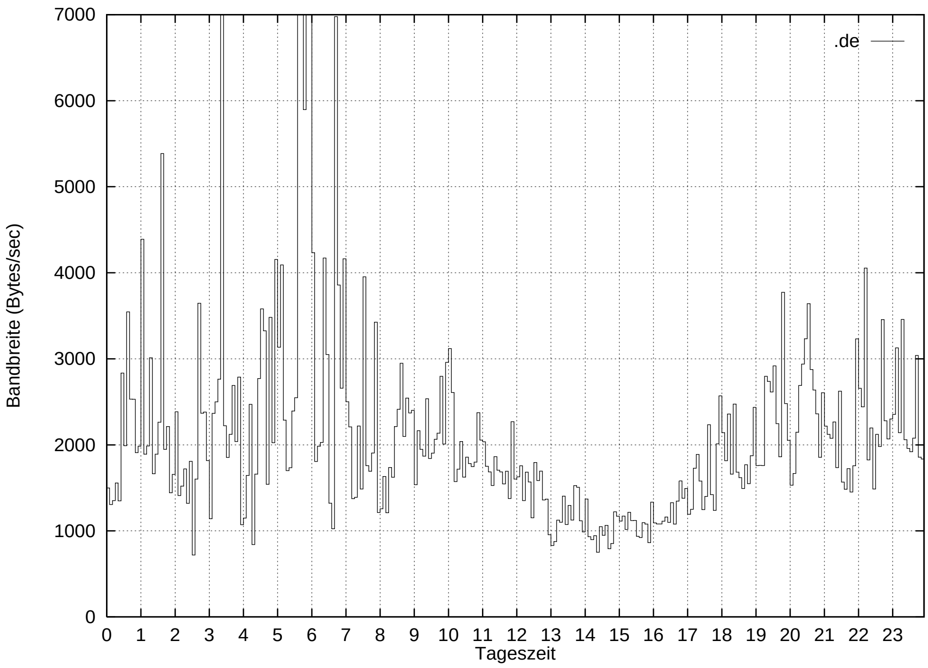


Abbildung 4-9: Gemessene Bandbreite für Server der `de`-Domäne

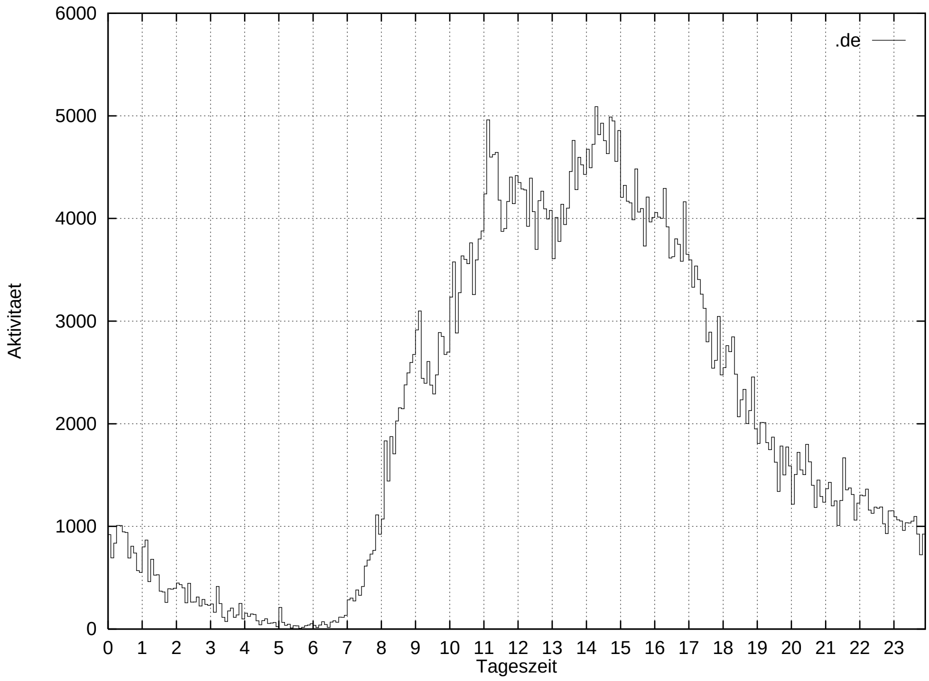


Abbildung 4-10: Aktivitäten-Diagramm zur `de`-Domäne

Die drei eben betrachteten Beispiele haben schon gezeigt, daß durchaus eine Abhängigkeit zwischen der geographischen Lage bestimmter Servergruppen und der vorhandenen Bandbreite besteht. Die nächten vier Beispiele auf Seite Seite 79 bis 81 sollen zeigen, wie sich diese Abhängigkeit bemerkbar macht: In der Messung wurden für die Kontinente folgende Domänen zusammengefaßt:

- Nordamerika: .com .edu .gov .mil .us .ca
- Asien: .jp .tw .kr
- Australien: .au .nz
- Europa: .nl .dk .fr .it .no .fi .se .uk .ru .es .pl

Bei allen vier Kontinenten ist der typische Verlauf wiederzuerkennen. Die lokale Aktivität hat dabei offenbar genauso einen Einfluß auf die Bandbreite wie die Entfernung der Domänen. Europa liegt dabei zusammen mit Nordamerika deutlich vor den weiter entfernten Kontinenten Asien und Australien. Die deutliche Spitze im Nordamerika-Diagramm deutet auf eine sehr leistungsstarke Anbindung hin, die aber tagsüber voll ausgelastet ist. In den Nachtstunden wird sie aber soweit entlastet, daß solche Übertragungsraten möglich werden (obwohl die Aktivität vergleichsweise niedrig ist, haben doch genug Übertragungen zu dem entsprechenden Bandbreiten-Wert geführt, daß er nicht mehr als zufällig angesehen werden kann). Weiterhin ist der leicht verzögerte Anstieg bis ungefähr 3 Uhr eine Hinweis auf die zusätzliche Belastung der WWW-Server durch Anfragen aus Nordamerika selbst.

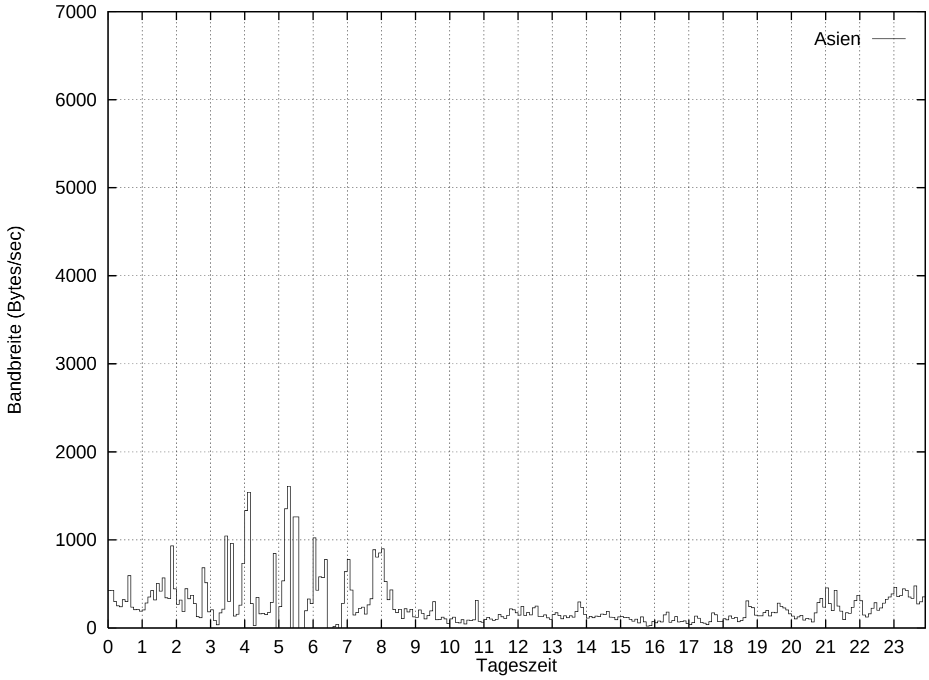


Abbildung 4-11: Bandbreite: Asien

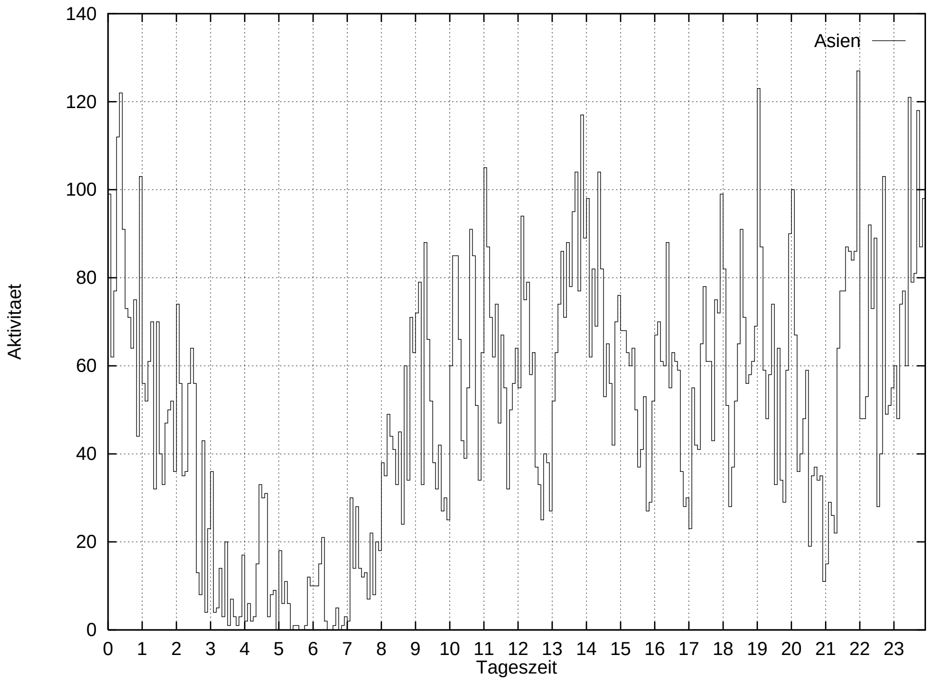


Abbildung 4-12: Aktivitäten-Diagramm: Asien

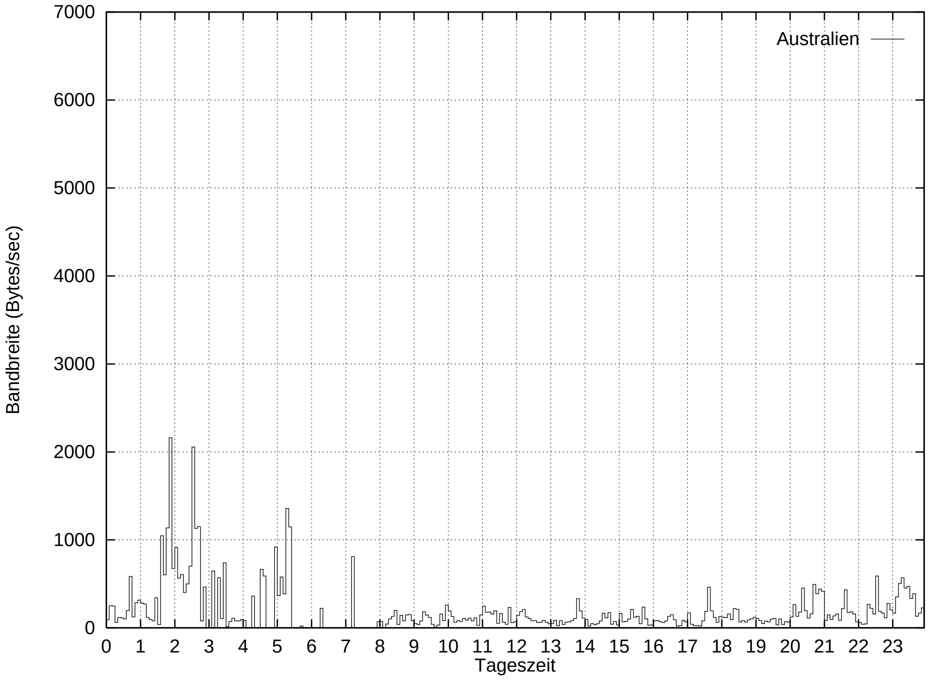


Abbildung 4-13: Bandbreite: Australien

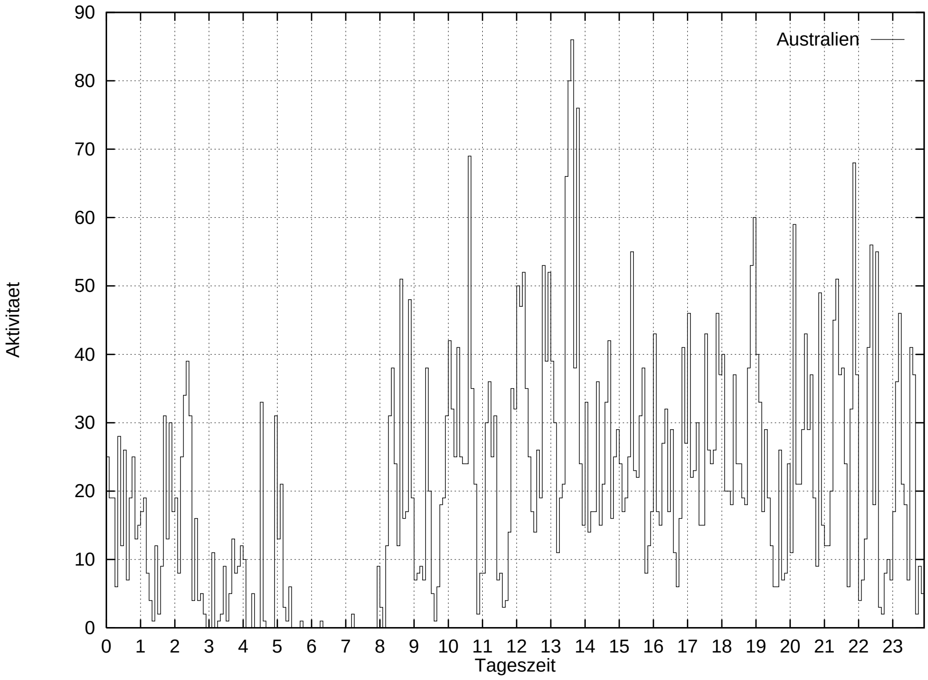


Abbildung 4-14: Aktivitäten-Diagramm: Australien

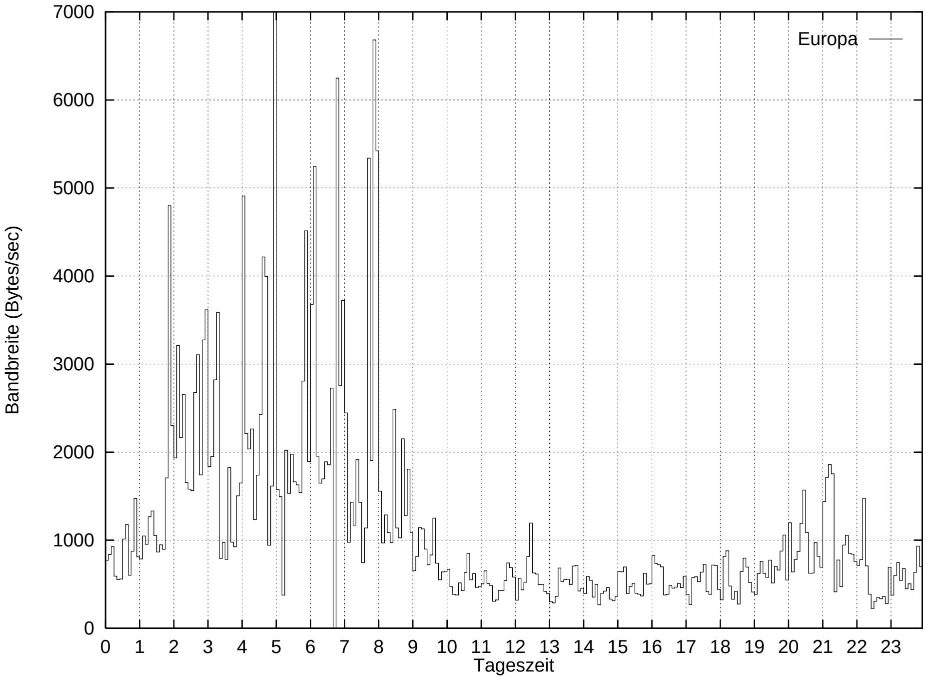


Abbildung 4-15: Bandbreite: Europa

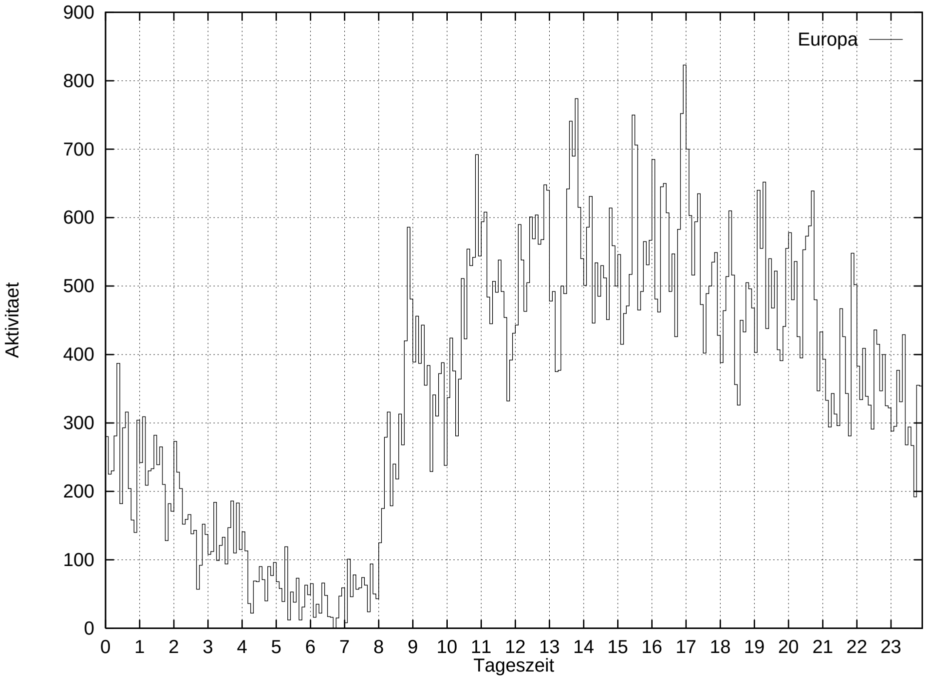


Abbildung 4-16: Aktivitäten-Diagramm: Europa

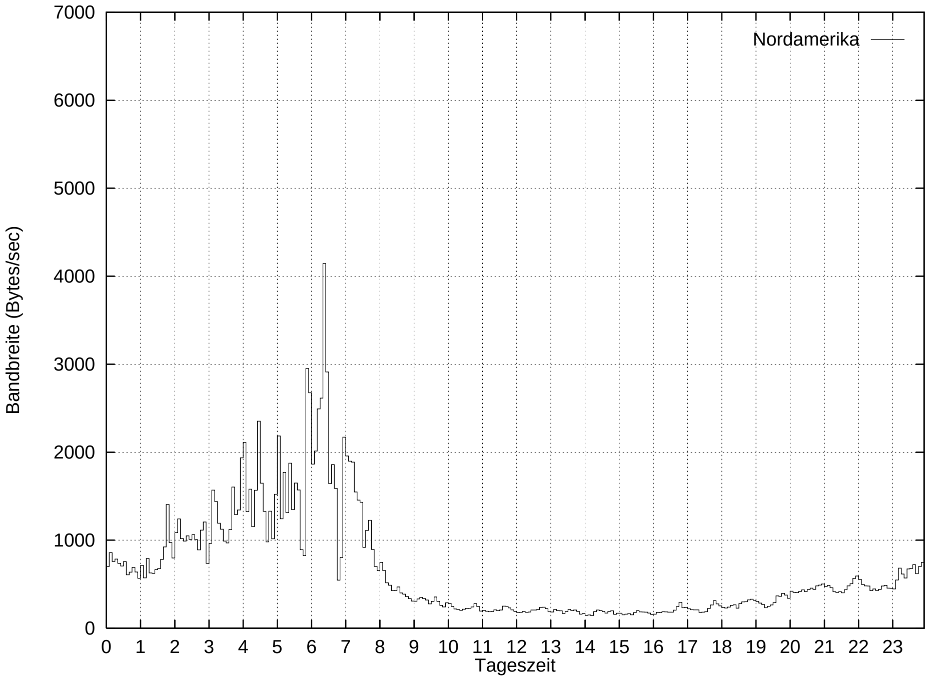


Abbildung 4-17: Bandbreite: Nordamerika

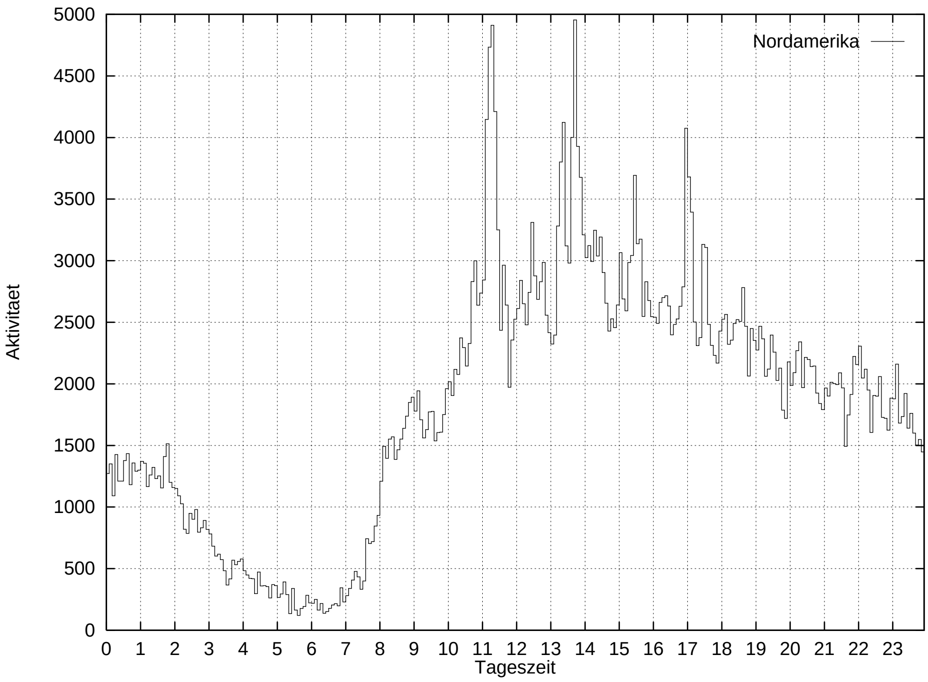


Abbildung 4-18: Aktivitäten-Diagramm: Nordamerika

4.3 Übertragungszeitenschätzung

Die Untersuchungen der Schätzmodelle, die in diesem Kapitel beschrieben werden sollen, sind in vier Teilabschnitte gegliedert. Nach einer allgemeineren Einführung an einem typischen Szenario erfolgt eine detailliertere Analyse der drei Schätzmodelle Average, StraightLine und Simple-StraightLine. Die verschiedenen, bei unterschiedlichem Zeitgranulaten berechneten Übertragungsprofile werden gegenübergestellt, die Qualität der Schätzungen abschließend anhand verschiedener Fehlerstatistiken genauer untersucht und ihre Schwachpunkte aufgezeigt.

Wie schon die Betrachtungen zur Übertragungsbandbreite gezeigt haben, ist die Häufigkeit von Serveranfragen unterschiedlich über den gesamten Tag verteilt. Abb. 4-19

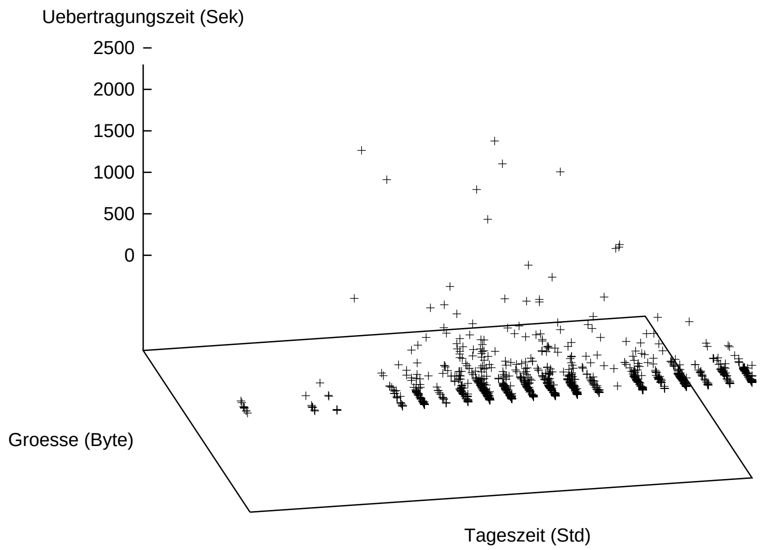
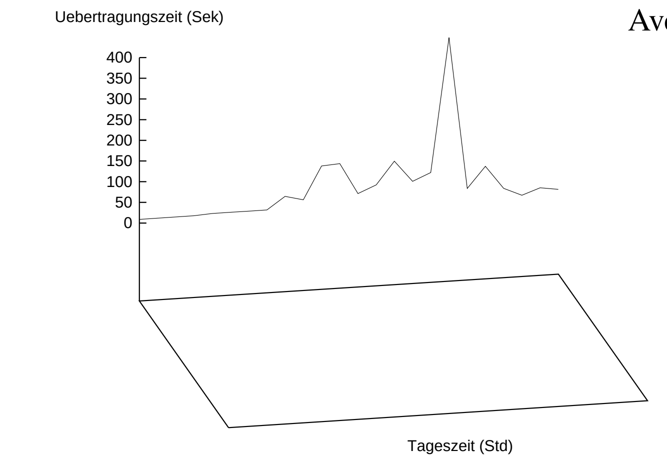
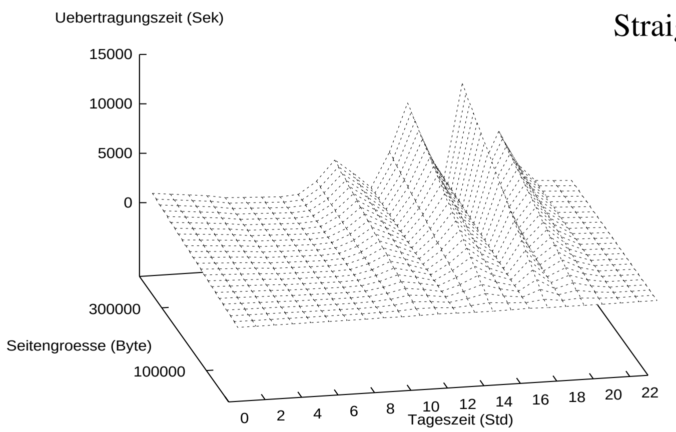


Abbildung 4-19: Meßpunkte für die Übertragungszeiten-Schätzung

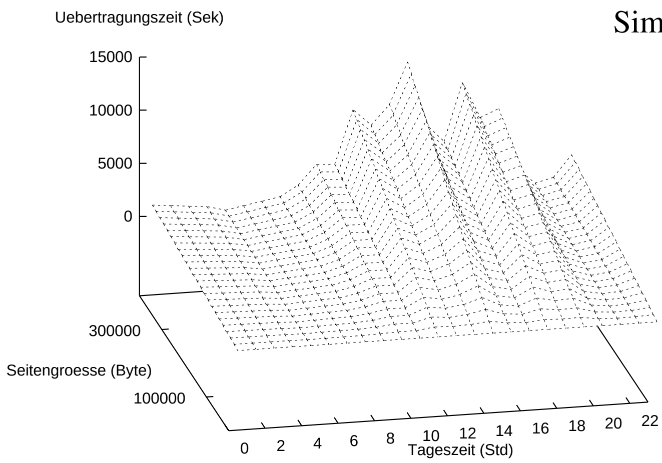
zeigt die Visualisierung einer typischen Meßpunktverteilung. Aus den Logfiles wurden dazu für jede Übertragung Tageszeit, Seitengröße und Übertragungsdauer gespeichert. Jeder Meßpunkt ist als „+”-Symbol in das Diagramm eingezeichnet. An der Verteilung der Punkte ist zu erkennen, daß die Werte in Zeitabschnitte zu je einer Stunde aufgeteilt wurden. Dies ist gleichzeitig auch die größtmögliche Zeitauflösung mit der die Schätzmodelle arbeiten können. Zur Mittags- und Abendzeit stehen aufgrund höherer Aktivität mehr Meßwerte zur Verfügung (zu erkennen an der dichteren Punktwolke). In Zeiten geringer Aktivität kann es aber auch vorkommen, daß innerhalb einer Stunde keine Übertragungen zu dem untersuchten Server stattgefunden haben und deshalb auch keine Meßwerte vorhanden sind. Weiterhin ist zu erkennen, daß kleine Seitengrößen den überwiegenden Teil der Meßpunkte einnehmen und für größere Seiten die Anzahl der Meßpunkte deutlich sinkt. Aus diesen Werten muß nun jedes der drei Schätzverfahren ein Übertragungsprofil berechnen, mit dessen Hilfe dann im weiteren Verlauf Seitenladezeiten geschätzt werden können. Abb. 4-20 zeigt die von den in Abschnitt 4.3 vorgestellten Verfahren berechneten Profile. Die oberste Graphik zeigt das vom Average-Modell generierte Übertragungsprofil. Obwohl es unabhängig von



Average-Modell



StraightLine-Modell



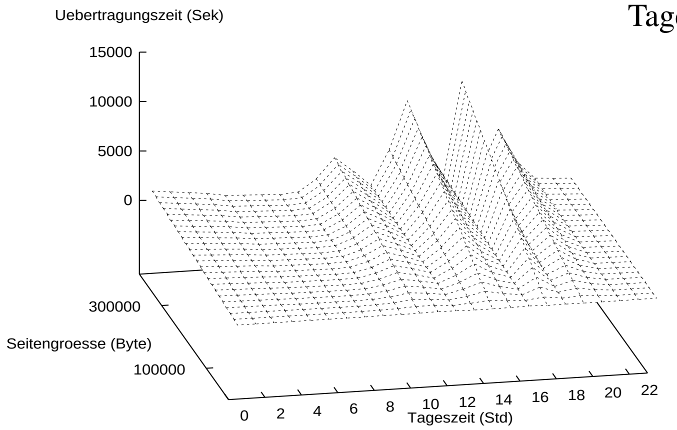
Simple-StraightLine Modell

Abbildung 4-20: Übertragungszeiten-Schätzung durch die drei Modelle der Seitengröße ist, wurde es, um einen besseren Vergleich mit den darunterliegenden Profilen des StraightLine- und des Simple-StraightLine-Modells zu ermöglichen, dreidimensional dargestellt. Die y-Achse, die räumlich gesehen nach hinten verläuft, ist aus diesem Grund auch nicht beschriftet. An dem Kurvenverlauf ist die höhere Netzbelastung im Verlauf des Tages zu erkennen. Genauso, wie die zur Verfügung stehende Bandbreite in diesen Belastungszeiten abnimmt, steigt die durchschnittliche Seitenübertragungszeit. Sie schwankt am Tage zwischen 50 und 150 Sekunden und erreicht in der Nachmittagszeit einen Spitzenwert von ca. 400 Sekunden. Für die Stunden, für die keine Meßwerte vorliegen, wurden die jeweiligen Durchschnittswerte aus benachbarten Durchschnitten interpoliert. Die Annahme, die dieser Maßnahme zugrunde liegt ist, daß sich in den Zeiten niedriger Aktivität die Übertragungsverhältnisse nicht sprunghaft ändern und somit dort benachbarte Meßwerte ebenfalls eine gute Schätzbasis bilden. Natürlich gilt, daß je größer der Bereich fehlender Meßpunkte ist, desto wahrscheinlicher Fehler durch diese Interpolation werden.

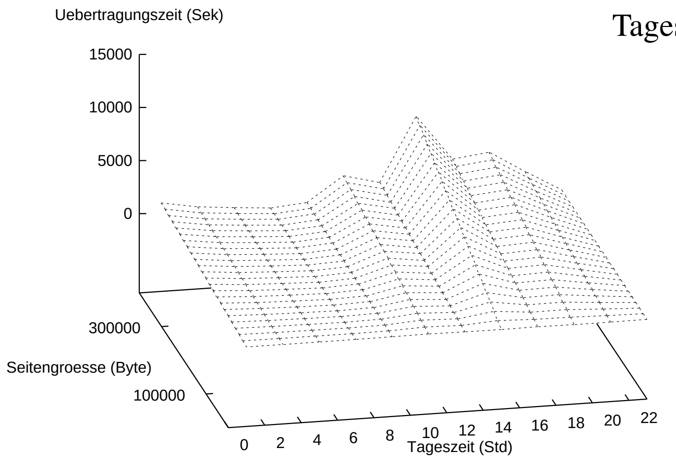
Zum Vergleich sind die Profile des StraightLine- und des Simple-StraightLine-Modells unter dem des Average-Modells abgebildet. Für jede der 24 Stunden wurde jeweils eine Übertragungsgerade berechnet (vgl. Abschitt 2.5 auf Seite 25). CaSimiR hat die Visualisierungsdaten so aufbereitet, daß ein komplettes Profil als dreidimensionales Gitter dargestellt werden kann. Die einzelnen Schätzgeraden sind entlang der y-Achse nach hinten verlaufend zu erkennen. Die horizontal im Gitter verlaufenden Linien dienen einem besseren Vergleich der Schätzgeraden. Durch sie wird die plastische dreidimensionale Darstellung möglich, bei der verdeckte Flächen ausgeblendet werden können. Bei einem Blick auf die z-Achse fällt zunächst die deutlich höhere maximal geschätzte Übertragungszeit auf. Während das Average-Modell höchstens eine Übertragungszeit von 400 Sekunden schätzen kann (für eine 400 KB große Seite wäre das z.B. eine Übertragungsrate von 1KByte/s), schätzt das StraightLine-Modell diese Ladezeit für dieselbe Seite in den Nachmittagsstunden auf ca. 10000 Sekunden, also fast 3 Stunden. Dies entspräche einer Übertragungsrate von 40 Byte/s. Das Simple-StraightLine-Modell schließlich schätzt die Zeit noch etwas höher. Offensichtlich liegen die geschätzten Zeiten je nach Modell weit auseinander und es stellt sich die Frage, welches Verfahren nun im Schnitt die geringsten Fehler bei diesen Schätzungen macht.

Vor der Fehlerbetrachtung aber noch eine Bemerkung: Die hohen Schätzwerte des StraightLine- und Simple-StraightLine-Modells sind gleichzeitig auch ein Hinweis auf die Probleme der Modelle mit Schwankungen bei den gemessenen Übertragungszeiten, insbesondere bei kleinen Seitengrößen. Beide Modelle neigen zu einer Überschätzung der Ladezeit bei großen Seiten, während beim Average-Modell genau das Gegenteil der Fall ist. Eine Maßnahme um diese Schwankungsabhängigkeit zu mildern, ist z.B. die Zeitabschnitte zu vergrößern, für die jeweils eine Schätzgerade berechnet wird. Dadurch werden mehr Meßpunkte in die Berechnung der Schätzgeraden mit einbezogen und Schwankungen einzelner gemssener Übertragungszeiten relativiert. Abb. 4-21 zeigt am Beispiel des StraightLine-Modells, wie sich ein gröberes Zeitgranulat auf das Profil auswirkt. Das schon bekannte Profil mit der 24-Stunden-Auflösung ist als oberste Graphik abgebildet. Eine Halbierung der betrachteten Tagesabschnitte führt zu einer Glättung der Spitzen. Insgesamt wird das Profil ruhiger, die Schwankungen zwischen den einzelnen Schätzgeraden sind nicht mehr so ausgeprägt. Die Reduzierung der Abschnitte auf vier (in der untersten Graphik zu erkennen) führt allerdings zu einem Verlust des charakteristischen Aussehens und die durchaus vorhandenen Abhängigkeiten der Übertragungszeit von dem jeweiligen Zeitpunkt, an dem die Übertragung

Tageszeitauflösung: 24 Werte



Tageszeitauflösung: 12 Werte



Tageszeitauflösung: 4 Werte

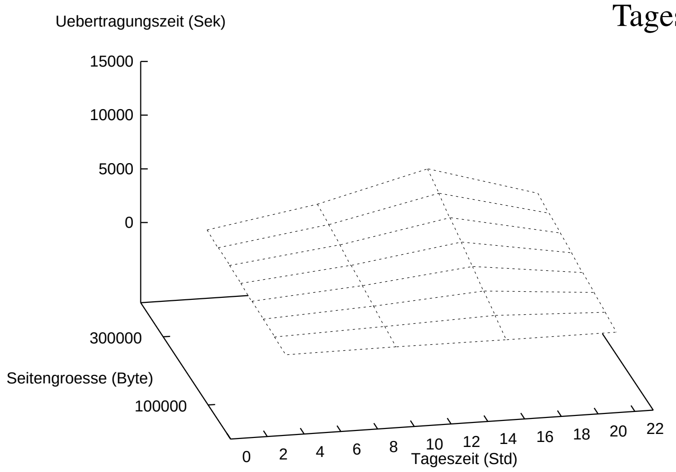


Abbildung 4-21: Übertragungszeiten-Schätzung bei variierenden Zeitauflösungen (StraightLine-Modell)

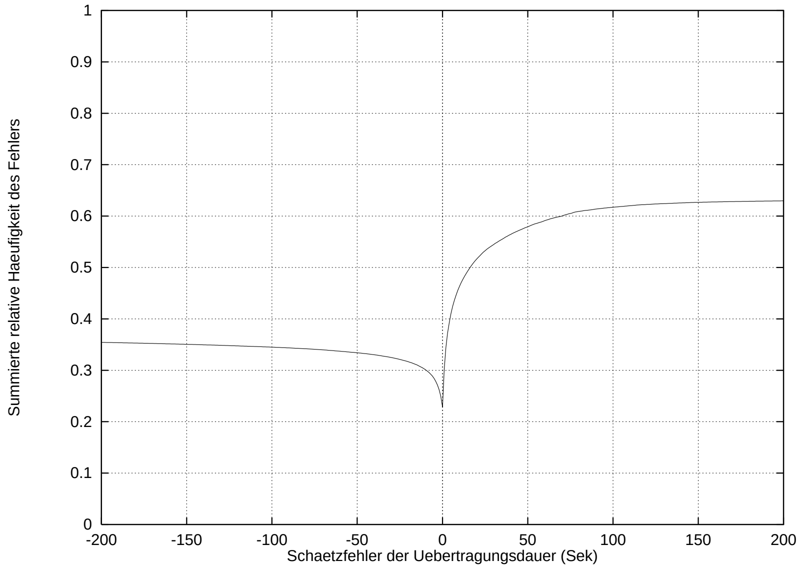
stattfindet, wird kleiner. Die Tageszeitauflösung sollte also nicht zu niedrig liegen und nur bei einer hohen Zahl von Meßwerten entsprechend fein gewählt werden.

Es folgt nun eine genauere Analyse der Schätzgenauigkeit. Zunächst soll sie anhand der Daten gemessen werden, die Grundlage für die Berechnung der Profile waren. Für die vorgestellten Diagramme wurden dazu die Übertragungszeiten aus den Monaten Juni und Juli '96 verwendet. Nach der Berechnung der Profile – es wurde eine Tageszeitauflösung von 24 Werten verwendet – wurde für jede Übertragungszeit ein Schätzfehler ermittelt, der sich aus der Differenz zwischen geschätzter und tatsächlicher Übertragungszeit berechnet. Ist der Fehler positiv, bedeutet das, daß die Übertragungszeit zu hoch geschätzt wurde. Ist der Fehler negativ, wurde sie zu niedrig geschätzt. Die in Abb. 4-22 dargestellten Diagramme stellen die summierte relative Häufigkeit der aufgetretenen Schätzfehler dar und sind wie folgt zu interpretieren: Auf der x-Achse ist der Schätzfehler in Sekunden abgetragen. Auf der y-Achse wird dazu die relative Häufigkeit angegeben, mit der Schätzungen mit *maximal* diesem Fehler gemacht wurden. Für den negativen Teil der x-Achse wird somit die relative Häufigkeit von Schätzungen aufsummiert, die Übertragungszeiten zu gering geschätzt haben. Im positiven Teil wird dies analog für die zu hoch geschätzten Übertragungszeiten angegeben. Ein gutes Schätzverfahren sollte also sowohl in positive als auch in negative x-Achsen-Richtung einen steilen Kurvenanstieg aufweisen, um dann in einem möglichst waagrechten Verlauf überzugehen, d.h. alle vorkommenden Schätzfehler wären bei einem guten Schätzverfahren dicht um den Fehlerwert von Null verteilt. Für den in den Diagrammen gewählten Bereich von -200 bis 200 Sekunden sollten sich die Werte am rechten und linken Rand zu einem Wert addieren, der nahe bei Eins liegt. Je näher die Summe an dieser Zahl liegt, desto geringer ist die Zahl der insgesamt gemachten „groben" Fehlschätzungen. Als letzter Punkt ist eine möglichst gleichmäßige Verteilung von positiven und negativen Schätzfehlern wünschenswert. Besonders bei der Beurteilung von Caching-Heuristiken, bei der die Schätzung von Seitenladezeiten ebenfalls eine Rolle spielt, ist es wichtig, daß Seitenladezeiten weder zu stark unterschätzt (in diesem Fall würden sich die Vorteile von Caching verringern) noch zu stark überschätzt werden (hier würde Caching zu stark übervorteilt).

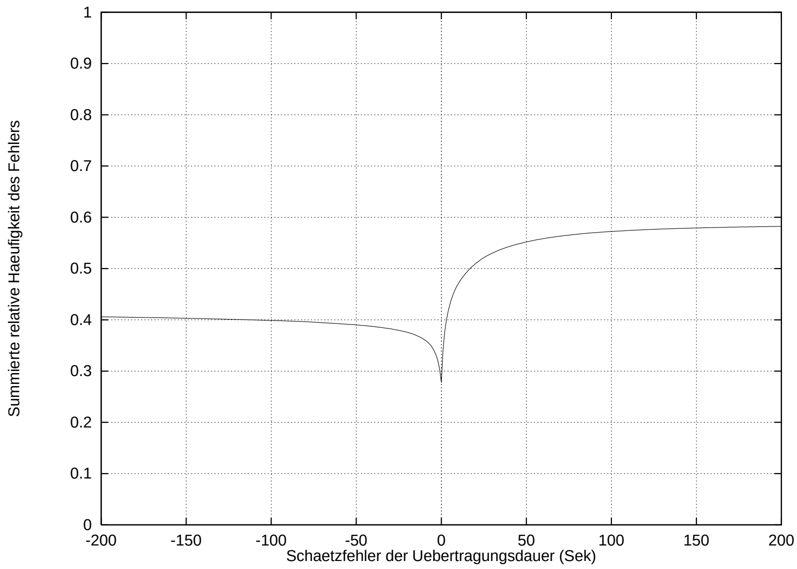
Zu den Diagrammen auf Seite 87: Für das Average-Modell ist hier indirekt die weiter oben schon erwähnte Eigenschaft zu erkennen, Übertragungszeiten für große Seiten zu unterschätzen, dagegen aber Übertragungszeiten für kleine Seiten zu überschätzen. Da kleine Seiten einen hohen Anteil an den insgesamt übertragenen Seiten einnehmen, ist die Kurve auf der negativen x-Achse relativ flach (nur 35,4% aller Schätzfehler liegen im Intervall von -200 bis 0 Sekunden), während 62,9% aller Schätzungen bis zu 200 Sekunden zu hoch waren. Zusammen waren also 98,3% aller Schätzungen mit einer Ungenauigkeit von höchstens +-200 Sekunden zu verbunden. Will man nur ein kleineres Fehlerintervall tolerieren, beispielsweise eines von -50 bis 50 Sekunden, reduziert sich entsprechend die Anzahl der Schätzungen, die als akzeptabel angesehen werden auf 91,3%.

Für das StraightLine-Modell erhöht sich der Anteil der zu niedrig ausgefallenen Schätzungen. Es ergibt sich ein ausgeglicheneres Verhältnis zwischen über- und unterschätzten Zeiten. Insgesamt gesehen liegen die Schätzfehler dichter beieinander, was sich an der geringfügig höheren Summe der Fehlerhäufigkeit im Bereich von -200 bis 200 Sekunden (98,7%) bemerkbar macht. Für das kleinere Fehlerintervall von -50 bis 50 Sekunden ergeben sich 94,1 Prozent.

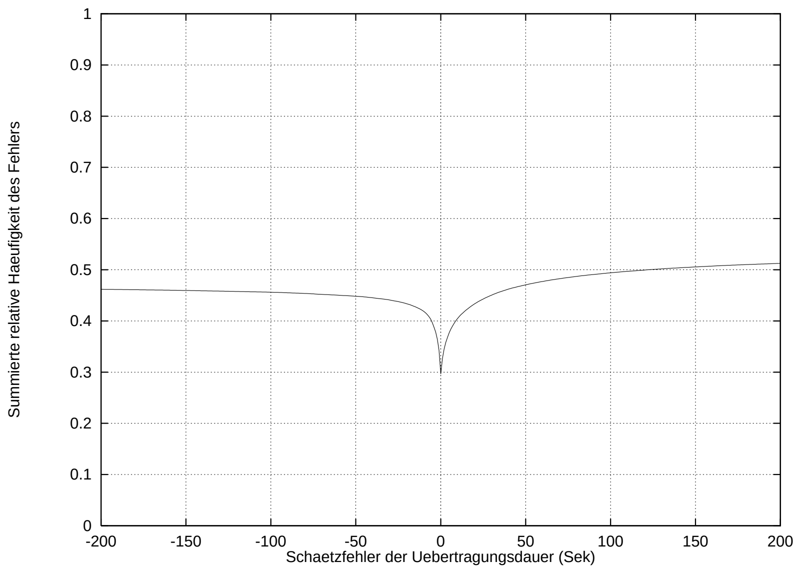
Bemerkenswert beim Simple-StraightLine-Modell ist die gleichmäßige Verteilung von über- und unterschätzten Zeiten. Ungefähr 46% der Schätzungen im negativen



Average-Modell



StraightLine-Modell



Simple-
StraightLine Modell

Abbildung 4-22: Fehlerverteilung bei der Profilerstellung

Bereich von -200 bis 0 Sekunden stehen ca. 51% im Bereich 0-200 Sekunden gegenüber und summieren sich zu insgesamt 97,3%.

Zu den Fehlerintervallen ist noch zu bemerken, daß bei einer langen Seitenladezeit ein größerer Fehler eher hingenommen werden kann, als bei einer Seite, die in wenigen Sekunden geladen ist. Deshalb wird sich die weiter unten folgende Analyse genauer mit der Abhängigkeit der Schätzfehler von der Seitengröße beschäftigen.

Die Graphiken auf Seite 83 geben wieder, wie gut jedes einzelne Verfahren aus vorgegebenen Meßpunkten entsprechende Übertragungsprofile generieren kann. Sie machen jedoch keine Aussage darüber, wie gut mit den erstellten Profilen eine Vorhersage von Übertragungszeiten möglich ist. Zur Beantwortung dieser Frage wurden die aus den Übertragungszeiten der Monate Juni und Juli '96 berechneten Profile an den Übertragungszeiten des Monats August getestet. Wie bei der vorangegangenen Analyse auch, wurde für jede Übertragung die Differenz zwischen tatsächlicher und geschätzter Übertragungszeit gespeichert und daraus die Diagramme zur summierten relativen Fehlerhäufigkeit generiert (zu sehen in Abb. 4-24). Für alle drei Modelle bleibt der charakteristische Kurvenverlauf erhalten. In allen drei Diagrammen sinkt der Anteil der

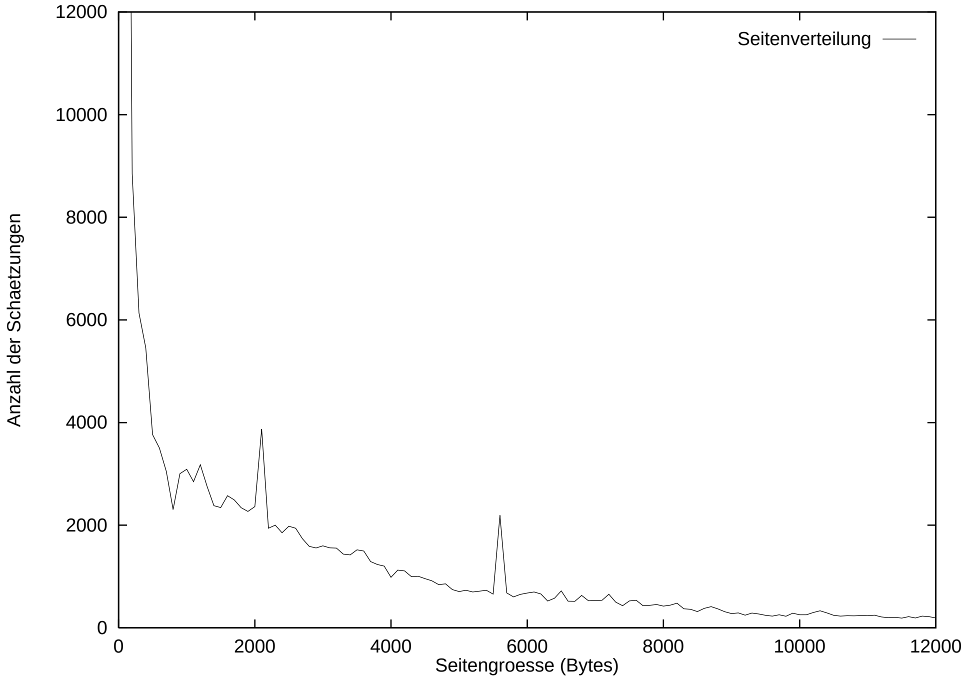
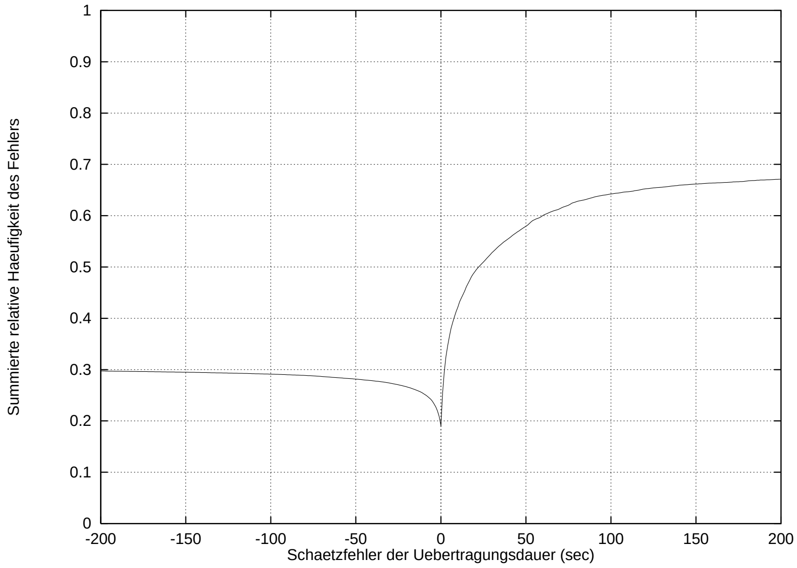
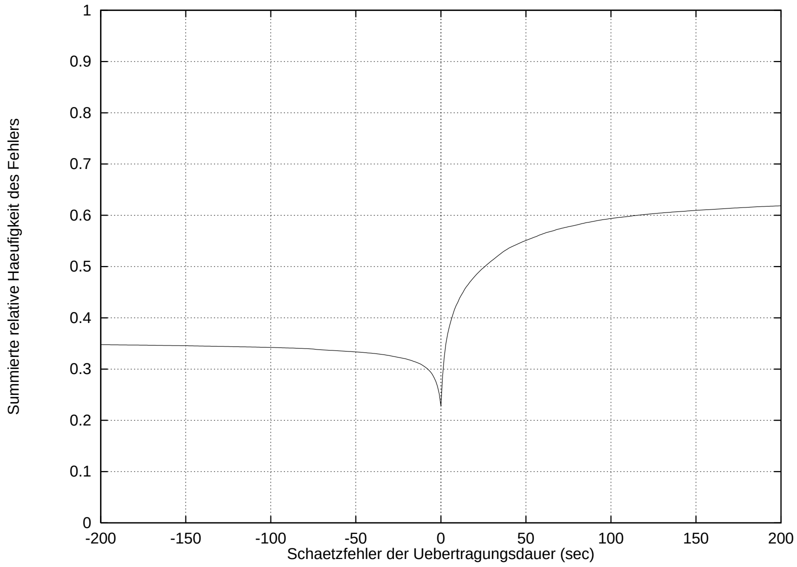


Abbildung 4-23: Verteilung der Seitengrößen

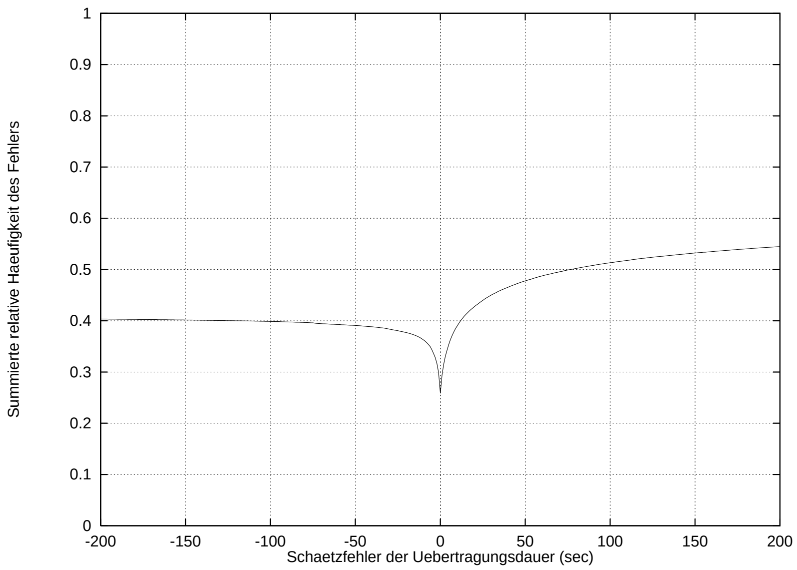
Schätzungen mit einer Abweichung von +-200 Sekunden leicht. Am stärksten ist das Simple-StraightLine-Modell davon betroffen (um 3 Prozentpunkte auf knapp 95%), gefolgt vom StraightLine-Modell (um 2 Prozentpunkte auf 96,5%) und dem Average-Modell (auf knapp 97%). Übereinstimmend ist eine Verschiebung der Anteile vom negativen in den positiven Bereich zu beobachten. Daß selbst beim Average-Modell der überwiegende Teil der Zeitschätzungen zu hoch ausfiel, ist ein Hinweis auf die Verteilung der Seitengrößen (Abb. 4-23). Die Abbildung zeigt, daß der Hauptanteil der Schätzungen auf Seiten mit einer Größe unter 12 KB entfällt und erklärt, warum die z.T. enormen Schätzfehler nur Einzelfälle bleiben. Die Güte eines Schätzverfahrens entscheidet sich somit bei den kleinen Seitengrößen. Der betrachtete Bereich bei der Darstellung des mittleren Schätzfehlers pro Seitengröße wurde für eine genauere Begutachtung deshalb in Abb. 4-26 auf Seiten unter 16 KB eingeschränkt. Das Average-Modell zeigt in diesem Seitenbereich ein gleichmäßig überschätzendes Verhalten,



Average-Modell



StraightLine-Modell



Simple-StraightLine-Modell

Abbildung 4-24: Fehlerverteilung bei der Übertagungszeiten-Schätzung

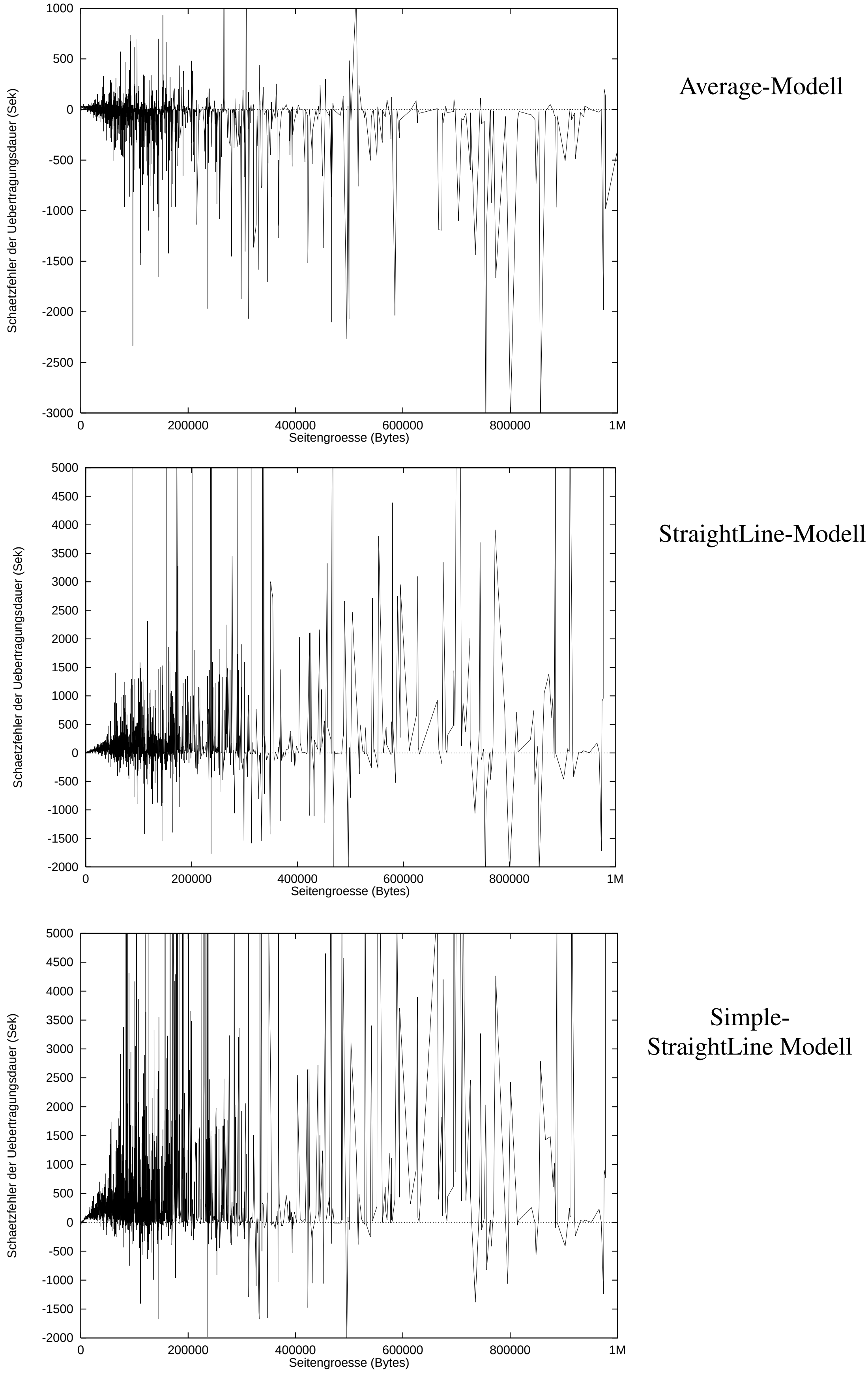


Abbildung 4-25: Schätzfehler in Bezug zur Seitengröße

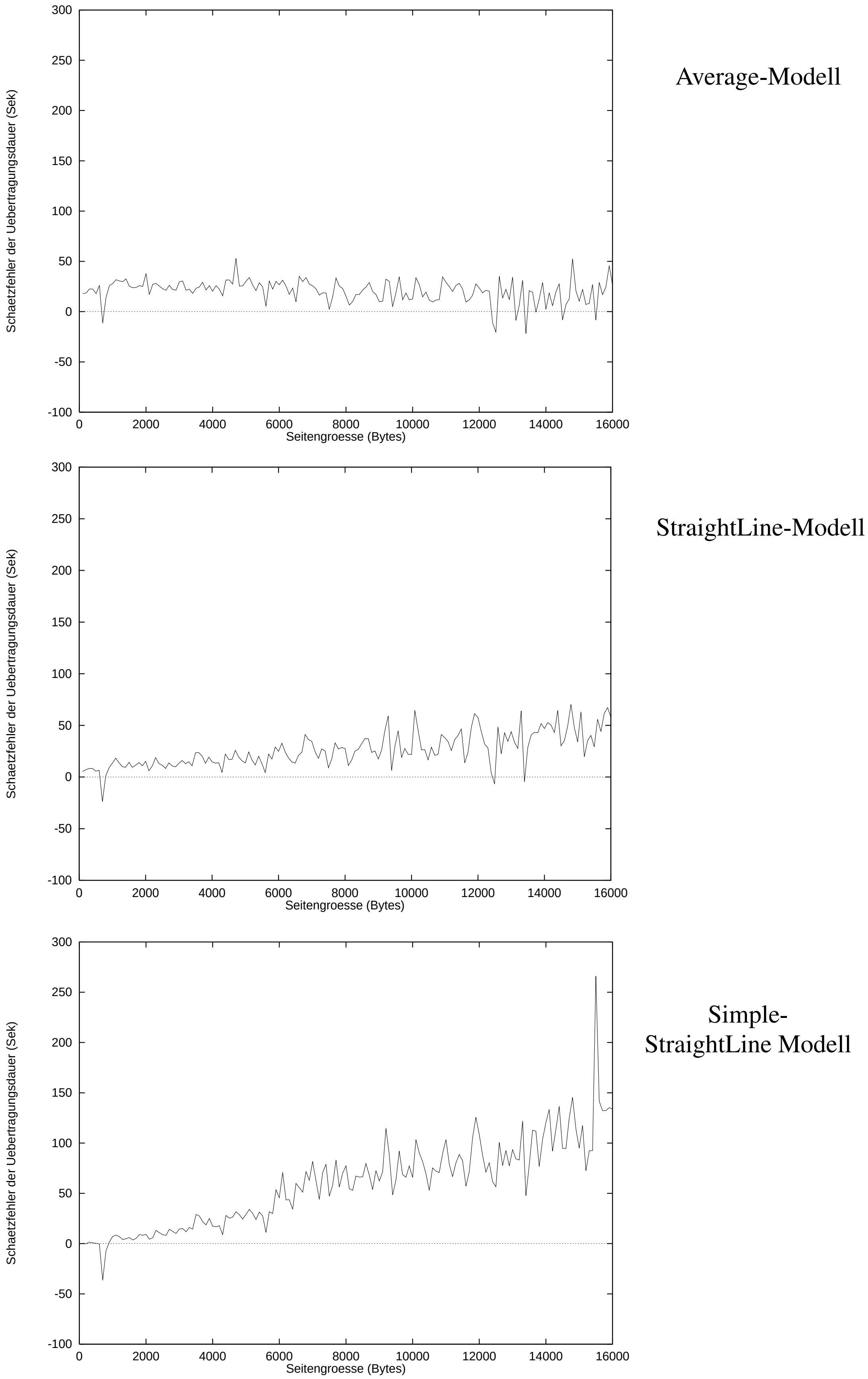


Abbildung 4-26: Schätzfehler in Bezug zur Seitengröße (vergrößert)

ohne dabei besonders große Schwankungen aufzuweisen. Dies erklärt die im Diagramm zur Häufigkeitsverteilung festgestellte Verlagerung hin zu den positiven Schätzfehlern. Zu den großen Seiten hin wird die schon angesprochene Tendenz zur Unterschätzung in diesem Bereich bereits sichtbar.

Für die Modelle StraightLine und Simple-StraightLine erkennt man das bessere Schätzverhalten für kleine Seiten (ungefähr bis 6K). Vor allem das Simple-StraightLine-Modell liegt im weiteren Verlauf mit seinen Schätzungen deutlich über den tatsächlichen Übertragungszeiten. Einzig der ausgeprägte negative Einschnitt im Seitenbereich bis ca. 1K sorgt für ein relativ ausgeglichenes Verhältnis zwischen Anzahl der über- und unterschätzten Ladezeiten.

Bemerkung: Bei der Vorstellung der Verfahren in Kapitel 2 wurde schon angedeutet, daß beim Simple-StraightLine-Verfahren die Hinzunahme von neuen Meßpunkten nur steilere Schätzgeraden verursacht, die dann für die großen Überschätzungen verantwortlich sind. Insbesondere die Schwankungen bei den geradenbestimmenden Punkten, wirken sich stark auf das Schätzverhalten aus. Die Fehlerbetrachtungen bestätigen diese Abhängigkeit und die damit verbundene Überschätzung.

Das Average-Modell bietet trotz seiner Einfachheit akzeptable Schätzleistung. Falls das Überschätzen bei kleinen und Unterschätzen bei großen Seiten in Kauf genommen wird, steht ein Verfahren zur Verfügung, das leicht implementierbar ist und sich aufgrund der leichten Vergleichbarkeit verschiedener Profile gut für den Einsatz z.B. für ein CgR erweiterten Proxy-Cache einsetzen läßt. Wird mehr Genauigkeit bei kleineren Seiten benötigt, ist das StraightLine-Modell ein interessanter Kandidat. Gute Schätzleistungen bei kleinen Seiten stehen dabei Überschätzungen von Ladezeiten großer Seiten gegenüber. Soll die gleichmäßigere Verteilung von über- und unterschätzten Zeiten genutzt werden und kann dabei eine insgesamt größere Abweichung in Kauf genommen werden, ist das Simple-StraightLine-Modell mit in Betracht zu ziehen. Weiterführende Ansätze könnten eine Kombination der Schätzmodelle realisieren, um die jeweiligen Vorteile der einzelnen Verfahren zu verbinden.

4.4 Caching-Strategien

Nach den Auswertungen der Logfiles und den daraus gewonnenen Erkenntnissen, soll nun das Caching in den Mittelpunkt der Betrachtungen gerückt werden. Bevor die einzelnen in Kapitel 2 vorgestellten Verfahren in Abschnitt 4.4.2 gegenüber gestellt werden, widmet sich Abschnitt 4.4.1 zunächst den Grenzen, die den Caching-Verfahren gesetzt sind. Anhand eines Caches ohne Größenbeschränkung, werden die erreichbaren Grenzwerte ermittelt und der Einfluß von konsistenzerhaltenden Maßnahmen auf sie untersucht. Weitere Maßnahmen zur Verbesserung der Cache-Leistung, wie die Beschränkung der maximalen Seitengröße für in den Cache einzulagernde Seiten oder die Einführung eines Bereichs für die Verwaltung replizierter Seiten werden ebenfalls analysiert.

Alle in diesem Kapitel vorgestellten Ergebnisse wurden auf der Basis der aufgezeichneten Anfragen des Monats Mai ‘96 (620745 Einträge) gemacht. Zum Auffüllen des Caches vor jedem Simulationslauf dienten die Anfragen des Vormonats April ‘96 (554286 Einträge). Die zwei Light-Logfiles für diese Monate sind unter der Verwendung von Profilen folgender Zusammensetzung generiert worden: geschätzt aus den Übertagungszeiten des Monats April ‘96 findet ein Cache-Übertragungsprofil[1] (Tageszeitauflösung: 24 Werte) Verwendung. Mit einer Tageszeitauflösung von 12 Werten existieren Profile für die meistreferenzierten Domänen (darunter net, edu, com, org und de), zusammen mit den Profilen aller im Monat April referenzierten Server. Die zusätzliche Integration von Profilen für ganze Server-Domänen soll ermöglichen, daß für Server ohne gespeichertes Übertragungsprofil, aus dem Monat Mai zumindest auf Übertragungsinformationen der entsprechenden Domäne zurückgegriffen werden kann. Um eine gleichmäßige Verteilung von über- und unterschätzten Ladezeiten bei der Logfile-Generierung zu erhalten, wurde das Simple-StraightLine-Modell gewählt.

4.4.1 Grenzen des Cachings

Ein Cache mit unbegrenztem Speichervermögen bildet den Ausgangspunkt für eine Grenzbetrachtung, die zeigen soll, wie gut ein Caching-Verfahren überhaupt werden kann (im simulierten Zeitraum). Dazu werden eine Reihe von Eigenschaften betrachtet, an denen auch der Einfluß von konsistenzerhaltenden Maßnahmen gut zu erkennen ist. Betrachtet werden diese Eigenschaften:

- Byte-Hit-Rate
  Das angegebene Verhältnis, von aus dem Cache geladenen und von den Servern übertragenen Daten, kann als Maß für die durch das Caching-Verfahren erbrachte Netzentlastung betrachtet werden. Zusätzliche Kommunikation, die bei der Konsistenzerhaltung des Caches stattfindet, geht in dieses Bewertungskriterium nicht ein.
- Cache-Hit-Rate
  Wird zum Vergleich zur Byte-Hit-Rate mit in die Messung aufgenommen. Die Frage, inwiefern sich die Byte-Hit-Rate von der Cache-Hit-Rate als klassischem Qualitätsmaß unterscheidet, soll mit dieser Maßnahme geklärt werden.
- Durchschnittliche Übertragungszeit
  Sie gibt an, wie lange im Schnitt ein Seitenzugriff dauert. Ein niedriger Wert ist aus Sicht des Benutzers wünschenswert. Verfahren, die eine niedrige durchschnittliche Übertragungszeit erreichen, eignen sich daher für Bereiche, in denen für Clients ein möglichst günstiges – sprich schnelles – Umfeld geboten werden soll.
- Stale-Rate
  Beurteilt, wie konsistent die aus dem Cache gelieferten Daten mit denen der anbietenden Servern war. Die Konsistenz sollte mit möglichst wenig unnötigen Aufwand erreicht werden. Dieser kann mit folgendem Maß festgehalten werden:
- 304-Reply-Rate
  Beschreibt, inwieweit unnötiger Aufwand bei der oben erwähnten Konsistenthaltung des Caches betrieben wurde. Die 304-Reply-Rate zählt nämlich die Rückfragen zur Seitenaktualität an die Server, die zu keinem Neuladen der Seite führten und daher unnötig waren. Seltene Rückfragen lassen die 304-Reply-Rate sinken, bewirken aber im Gegenzug ein Ansteigen der Stale-Rate.
- Anzahl der von den Servern übertragenen Bytes
  In dieses Maß geht sowohl die Byte-Hit-Rate ein, als auch die Belastung durch die Kommunikation zwecks Erhaltung der Cache-Konsistenz. Verfahren, die hier niedrige Werte erzielen, sind interessant für Einsatzbereiche, in denen es auf eine möglichst effiziente Nutzung von Übertragungskapazitäten ankommt.

Die folgenden Betrachtungen für einen unendlich großen Cache zeigen zugleich die Auswirkungen von konsistenzerhaltenden Maßnahmen auf. Für eine Reihe von TTL-

1. Der Cache wird hier wie ein eigener Server behandelt. So können auch Ladezeiten aus dem Cache geschätzt werden.

Faktoren (0%, 0.5%, 1%, 2%, 5%, 10%, 12%, 15%, 20% 50% und 80%) wurden die o.g. Maße erhoben und graphisch aufbereitet. In die Diagramme sind zusätzlich die Werte eingetragen, die sich für einen unendlich großen Cache *ohne* Konsistenzerhaltung ergeben.

Bemerkung: Da in einem unendlich großen Cache immer ausreichend Platz für die Einlagerung neuer Seiten ist, brauchen nie Seiten verdrängt zu werden. Damit sind die dargestellten Werte unabhängig von den verwendeten Caching-Verfahren.

Für einen unendlich großen Cache ohne konsistenzerhaltenden Maßnahmen (TTL-Fenster unendlich groß) ergeben sich für den Monat Mai ‘96 folgende Werte:

Cache-Hit-Rate:	0.565245
Byte-Hit-Rate:	0.406327
durchschnittl. Zugriffszeit:	13.353 Sekunden
Stale-Rate:	0.281684
304-Reply-Rate:	0[1]
von Servern übertragene Bytes:	3.650.950.731

1. Da keine Get-If-Modified-Requests vom Proxy-Server versendet werden, existieren auch keine entsprechenden Antworten der WWW-Server, insbesondere keine 304-Replys.

Der Verlauf der Byte- und der Cache-Hit-Rate ist auf den Abbildungen 4-27 und 4-28 zu erkennen. Ihren niedrigsten Wert bei einem TTL-Faktor von 0 erreichend, steigen sie schon bei geringer Erhöhung des Faktors steil an. Ab einem TTL-Faktor von ca. 20% flachen beide Kurven merklich ab. Auch eine Verdopplung bzw. Vervierfachung des TTL-Faktors bringt nur noch eine kleine zusätzliche Erhöhung der Hitraten um ca. 1 Prozentpunkt. Während die Cache-Hit-Rate bei strikter Konsistenzerhaltung (TTL-Faktor 0) bei 48.3% liegt, kommt die Byte-Hit-Rate nur auf knapp über 33%. Die Differenz bleibt im weiteren Kurvenverlauf weitestgehend erhalten und ist ein Indiz dafür, daß anteilsmäßig mehr kleinere Seiten an den Cache-Hits beteiligt sind. Für die Byte-Hit-Rate fällt ein im Vergleich zur Cache-Hit-Rate ausgeprägterer Knick ab einem TTL-Faktor von ca. 1% auf. Er ist ein Hinweis darauf, daß größere Seiten offenbar selteneren Änderungen unterworfen sind und daher stärker an den Cache-Treffern beteiligt sind. Sobald die Anforderungen an die Konsistenz aber gelockert werden und die Cache-Hit-Rate über die 50%-Marke steigt, wird die Byte-Hit-Kurve flacher, was auf einen nun steigenden Anteil von kleineren Seiten hindeutet.

Die 52% bei der Cache-Hit-Rate bzw. knapp über 36% bei der Byte-Hit-Rate, die bei einem TTL-Faktor von 80% erreicht werden, liegen immer noch deutlich unter den Werten, die ohne Konsistenzerhaltung erzielt werden können (als durchgezogene horizontale Linie im Diagramm vermerkt). Nur durch die Inkaufnahme einer hohen Stale-Rate (siehe Abb. 4-29) kann dieser Vorteil erreicht werden. Hier zeigt sich, daß bereits die Einführung eines TTL-Faktors ausreicht, um die Zahl der veralteten aus dem Cache gelieferten Seiten deutlich zu reduzieren (bei einem TTL-Faktor von 80% kann in diesem Fall die Stale-Rate um fast ein Drittel gesenkt werden). Mit TTL-Faktoren unter 20% läßt sich die Stale-Rate dann noch einmal deutlicher reduzieren. Dies ist aber, wie ein Blick auf die Zahl der 304-Server-Replys (Abb. 4-30) zeigt, mit einem umgekehrt proportional zur Stale-Rate steigenden (unnötigen) Aufwand verbunden. Aufgrund dieses Aufwands steigt auch die mittlere Seitenladezeit mit kleinen TTL-Faktoren stark an

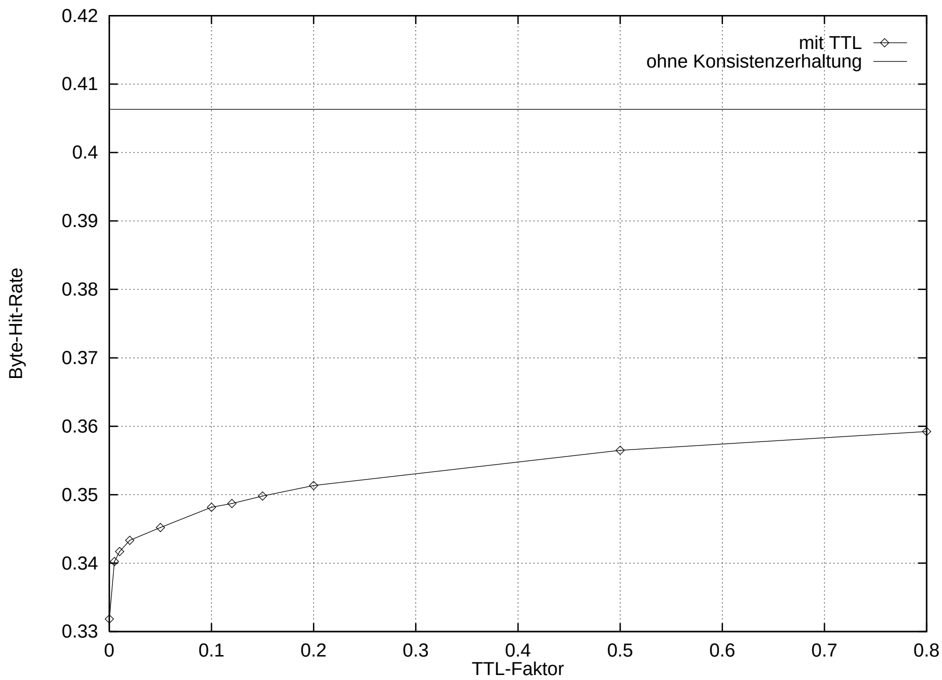


Abbildung 4-27: Byte-Hit-Rate für unendlich großen Cache

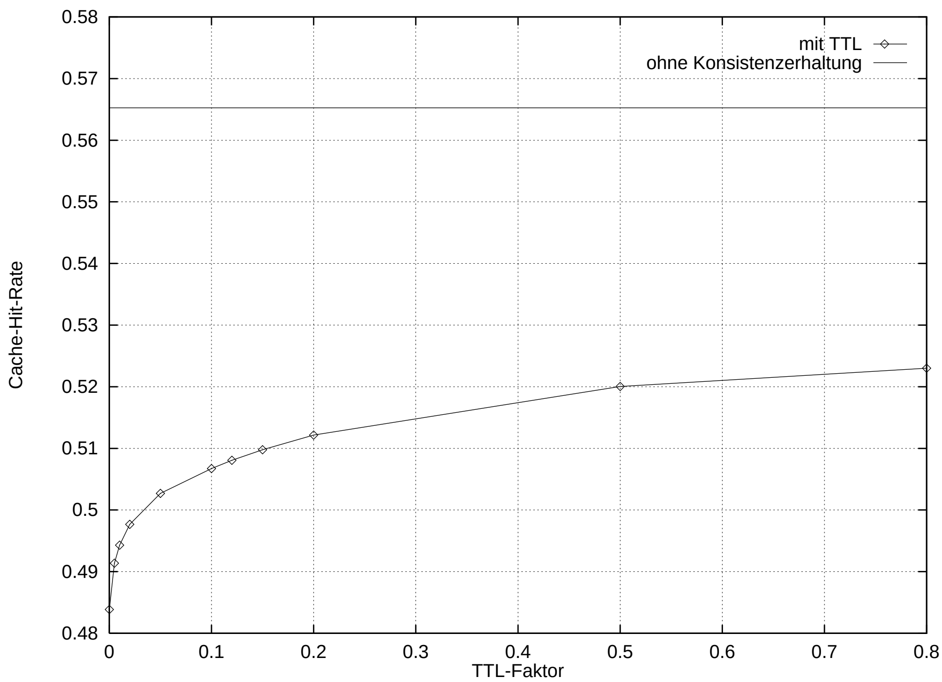


Abbildung 4-28: Cache-Hit-Rate für unendlich großen Cache

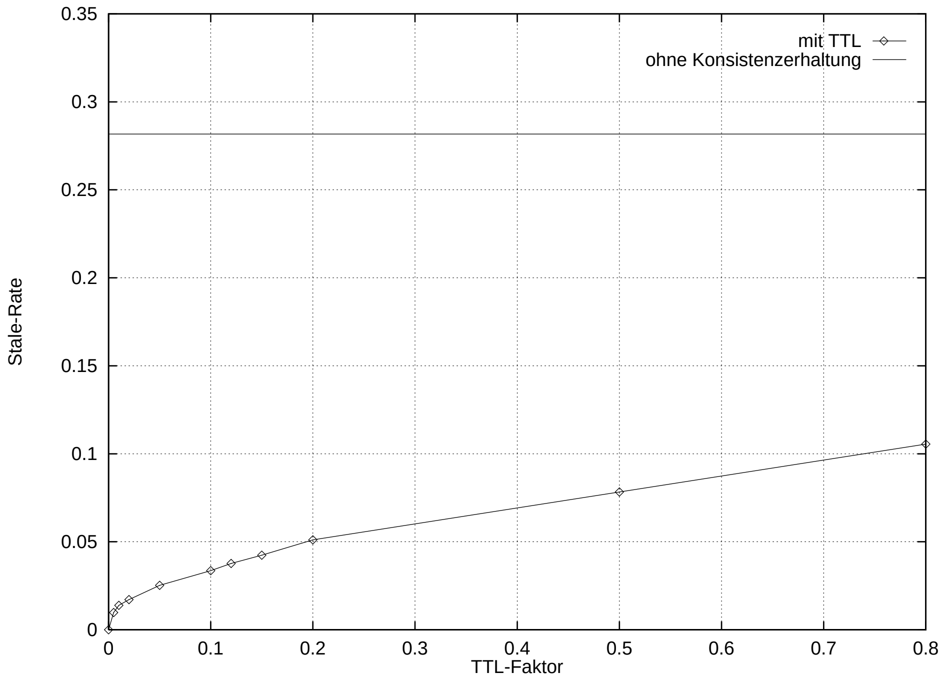


Abbildung 4-29: Stale-Rate für unendlich großen Cache

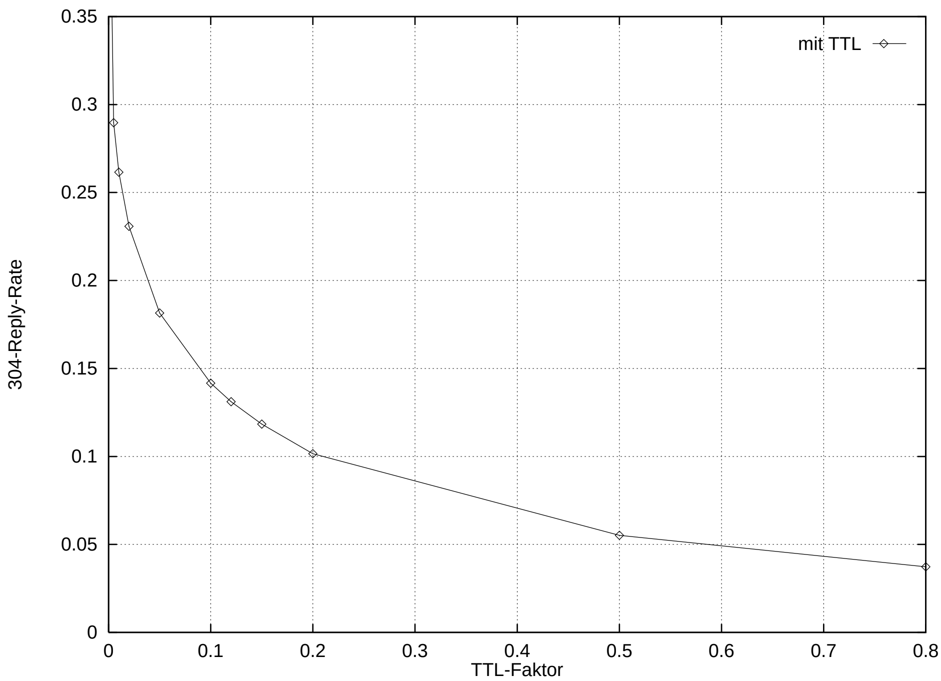


Abbildung 4-30: 304-Reply-Rate für unendlich großen Cache

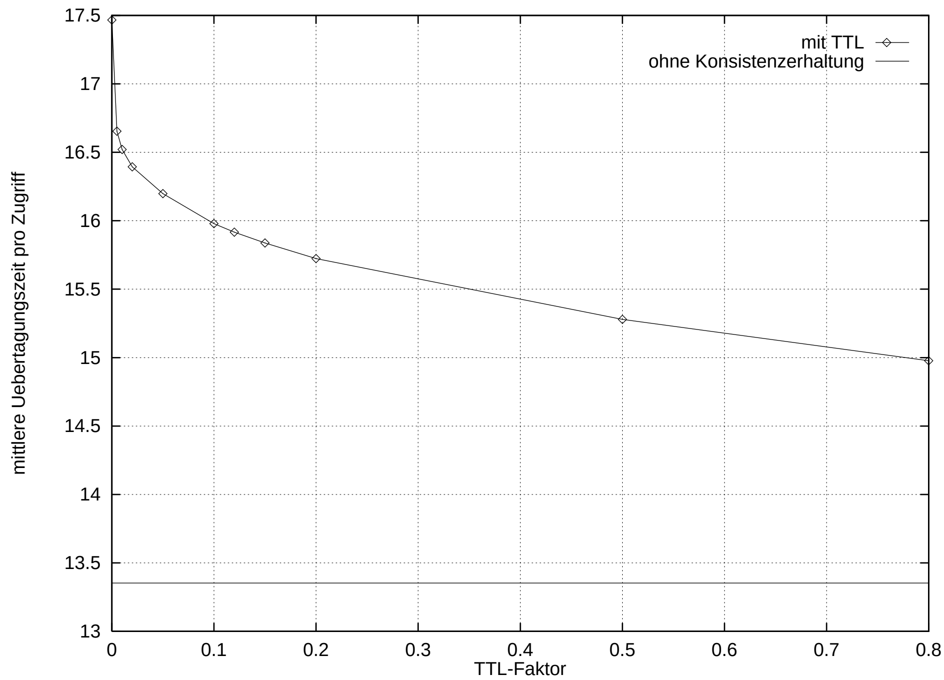


Abbildung 4-31: Mittlere Übertragungszeit für unendlich großen Cache

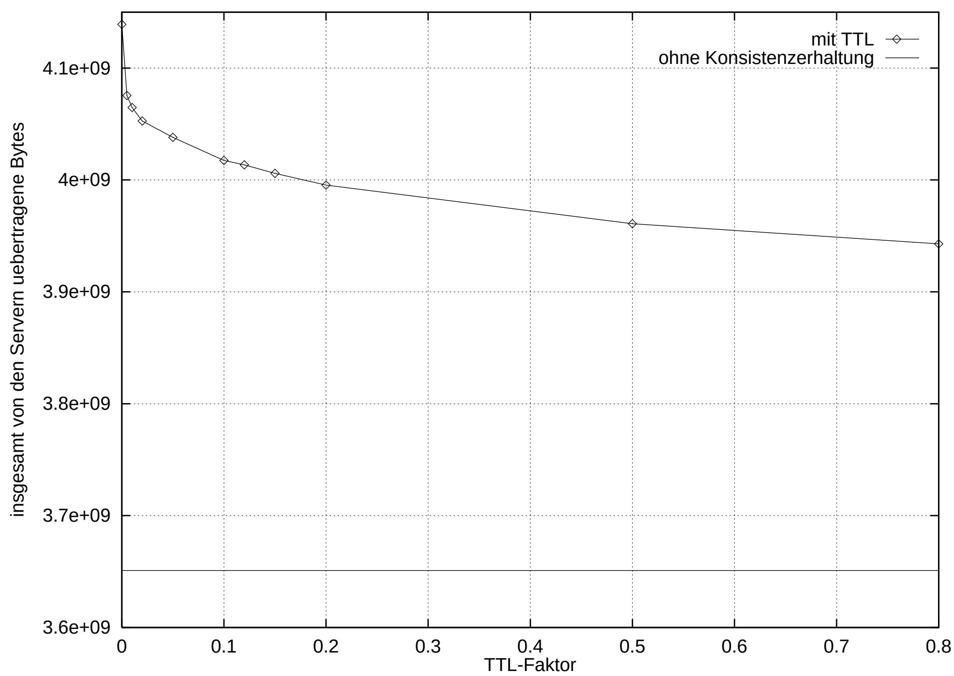


Abbildung 4-32: Anzahl der von den Servern übertragenen Bytes
für unendlich großen Cache

(siehe Abb. 4-31) und die Anzahl der von den Servern übertragenen Bytes nimmt aufgrund der vielen Get-If-Modified-Anfragen entsprechend zu.

Für die Wahl des TTL-Faktors gilt es, einen Kompromiß zu finden, zwischen der Konsistenzerhaltung des Caches und der durch sie verursachten zusätzlichen Kommunikation. Nimmt man eine Stale-Rate von 2-5% in Kauf, stellt ein TTL-Faktor von ungefähr 10% ein akzeptables Verhältnis von Aufwand und Cache-Leistung in Form von niedrigen mittleren Übertragungszeiten und geringer Anzahl von den Servern übertragener Bytes dar.

Bemerkung: Der lineare Anstieg der Stale-Rate mit dem TTL-Faktor ist u.U. beeinflußt durch das teilweise Fehlen der LastModified-Daten. Da für Seiten ohne feststellbares letztes Änderungsdatum ein Defaultalter von einem Monat eingesetzt wird, das sich nach einem weiteren Monat ändert, können gleichmäßig über die Zeit verteilte Änderungen hintereinander referenzierter Seiten auftreten. Durch eine Erhöhung des TTL-Faktors gehen dann immer mehr dieser Änderungen in die Stale-Rate ein. Der Einfluß des Defaultalters auf die Meßwerte sollte in anschließenden Arbeiten noch genauer geklärt werden.

4.4.2 Gegenüberstellung der Caching-Verfahren

Nachdem der letzte Abschnitt gezeigt hat, welche Grenzen selbst bei einem unendlich großen Cache existieren, sollen nun vier Caching-Verfahren TTL[1], Random, LRU und LCP (vgl. Abschitt 2.3 auf Seite 7) auf ihre Leistungsfähigkeit hin untersucht werden. Die Abhängigkeit der Leistung von der Cachegröße ist für die Werte 5 MB, 10 MB, 20 MB, 50 MB, 70 MB, 100 MB, 200 MB, 500 MB und 850 MB untersucht worden. Für jedes der Verfahren wurde bei den Messungen mit einem TTL-Faktor von 10% gearbeitet.

Hinweis: Alle vier Verfahren lagern, im Falle einer Verdrängung zunächst Seiten mit abgelaufenem Expiry-Datum aus, bevor nach der jeweiligen Verdrängungsstrategie noch gültige Seiten aus dem Cache ausgelagert werden. Somit verwendet auch das Random-Verfahren keine reine, zufällige Seitenverdrängung.

Die Cache-Hit- und die Byte-Hit-Rate sind in Abb. 4-33 und 4-34 dargestellt, die mittlere Übertragungszeit pro Zugriff findet sich auf Abb. 4-35 und die insgesamt für jedes Verfahren von den Servern übertragenen Bytes sind der Abb. 4-36 zu entnehmen. Den Abschluß bildet das die jeweiligen Stale-Raten darstellende Diagramm in Abb. 4-37 (Seite 102). Die exakten Meßwerte, die sich für die einzelnen Simulationsläufe ergeben haben, sind im Anhang ab Seite 119 in tabellarischer Form nocheinmal aufgelistet. Zusätzlich zum Vergleich sind die Werte des Caches in die Diagramme eingetragen worden, der zur Zeit der Logfileaufzeichnungen in Betrieb war. Dieser Cache war ein CERN 'httpd'-Cache, mit einem Speichervolumen von 500 MB. Die Verdrängung von Seiten wird durch eine nächtliche Löschung ungültiger Seiten realisiert, d.h. die im Verlaufe des Tages referenzierten Seiten werden ausnahmslos gespeichert (auch über das 500 MB-Limit hinaus). In der Nacht werden dann solange Seiten mit abgelaufenem Expiry-Datum gelöscht, bis das Speicherlimit wieder erreicht ist. Effektiv werden also, je nach Tagesaktivität, 600 MB und mehr des Cachespeichers in Anspruch genommen.

1. Das TTL-Verfahren verdrängt ausschließlich Seiten mit abgelaufenem Expiry-Datum. Kann dadurch nicht genug Platz im Cache geschaffen werden, wird die referenzierte Seite *nicht* eingelagert.

Die aus den Logfiles ermittelten Daten für den CERN-Proxy-Cache lauten wie folgt:

Cache-Hit-Rate:	0.213039
Byte-Hit-Rate:	0.166358
durchschnittl. Zugriffszeit:	22.012 Sekunden
304-Reply-Rate:	0.0688979
Stale-Rate:	0
von Servern übertragene Bytes:	4.992.987.253

Bemerkung: Die Informationen über die Änderungszeiten der referenzierten Seiten wurden über die im Cache gespeicherten Informationen ermittelt. Da keine Möglichkeit besteht zu überprüfen, ob eine ohne Rückfrage beim Server aus den Cache geladene Seite auch wirklich aktuell war, geht der Simulator von der optimistischen Annahme aus, daß zu jeder Seite bei ihrem Ladezeitpunkt das aktuelle letzte Änderungsdatum gespeichert war. Somit ergibt sich für den CERN-Cache eine Stale-Rate von 0%.

Für alle Verfahren steigen Cache-Hit- und Byte-Hit-Rate stetig mit zunehmender Cachegröße an. Immer mehr Seiten können in den Cache aufgenommen werden, und immer höher ist die Wahrscheinlichkeit, daß eine referenzierte Seite auch beim zweiten Zugriff noch im Cache gelagert ist ohne zwischenzeitlich durch andere Seiten verdrängt worden zu sein. Je größer der Cache ist, desto größer ist auch das Zeitintervall, aus dem in diesem Zeitraum referenzierte Seiten im Cache gehalten werden. Dieses Zeitintervall bestimmt, wie weit der zweite Referenzierungszeitpunkt einer Seite hinter dem ersten liegen darf, um sie weiterhin vor einer Verdrängung zu bewahren. Die im Vergleich zum LRU-Verfahren regelmäßig niedriger liegende Cache- bzw. Byte-Hit-Rate des LCP-Verfahrens ist in diesem Zusammenhang auf das durch den kleineren Pufferbereich bedingte kleinere Zeitintervall zurückzuführen. Das LCP-Verfahren hält zum erstenmal referenzierte Seiten im Bewährungspuffer, der nur einen Teil des Cachespeichers einnimmt. Somit werden Seiten bei ausbleibender oder nicht schnell genug nachfolgender zweiter Referenz schneller wieder verdrängt, als das beim LRU-Verfahren der Fall ist. Trotzdem, wie die mittlere Übertragungszeit (Abb. 4-35) zeigt, sind die Gewinne der im privilegierten Bereich gespeicherten Seiten groß genug, um eine insgesamt kürzere durchschnittliche Ladezeit zu erreichen. Die bessere Qualität der im Speicher gehaltenen Seiten gibt also hier den Ausschlag. Erst bei Cachegrößen über 500 MB setzt sich das LRU-Verfahren durch. Es können hier so viele Seiten in den Cache aufgenommen werden, daß wirklich nur lange nicht mehr referenzierte Seiten aus dem Cache verdrängt werden. Ein Auslagerungsentscheidung, bei der zwischen Seiten gewählt werden muß, die den letzten Referenzierungszeitpunkt betreffend als fast gleich angesehen werden können und bei denen das LCP-Verfahren dann die Seiten mit dem wenigsten zu erwartenden Gewinn auslagert, tritt bei diesen Speicherdimensionen immer seltener auf. Letztenendes ist es dann nur noch entscheidend, daß das Zeitintervall, aus dem die Seiten im Cache gehalten werden, so groß wird, daß selbst lange Zeiträume zwischen erstem und zweitem Referenzierungszeitpunkt einer Seite ohne Auslagerung überbrückt werden können.

Im Zusammenhang mit der höheren Byte-Hit-Rate des LRU-Verfahrens ist auch die Anzahl der insgesamt von den Servern geladenen Bytes (zu erkennen auf Abb. 4-36) niedriger. Für eine Cachegröße von 850 MB reduziert sich das verursachte Datenaufkommen im Vergleich zum LCP-Verfahren um über 6,5%. LRU ist also das Verfahren, das in einer Umgebung zum Einsatz kommen würde, in der es auf eine effiziente Nut-

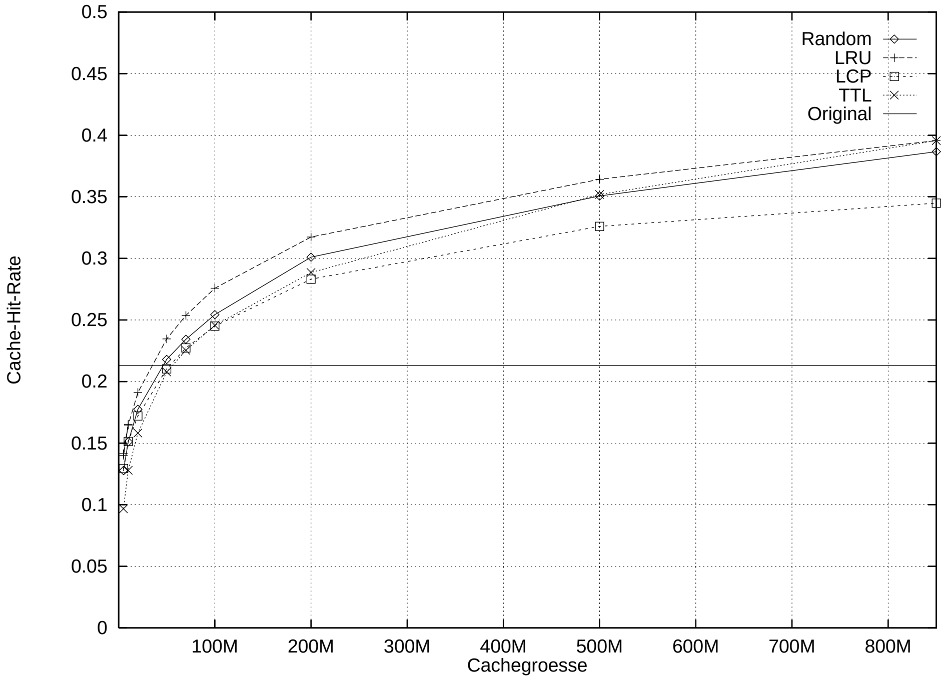


Abbildung 4-33: Cache-Hit-Rate

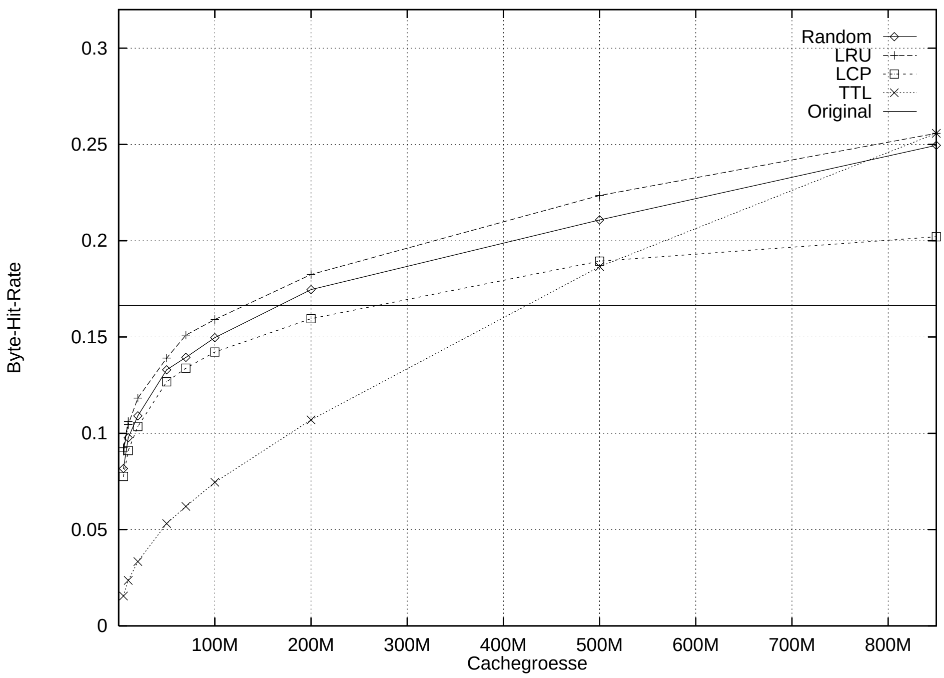


Abbildung 4-34: Byte-Hit-Rate

zung der vorhandenen Bandbreite ankommt. Das LCP-Verfahren hingegen wäre für eine Umgebung geeignet, in der wenig Speicher-Ressourcen zur Verfügung stehen und insgesamt eine möglichst niedrige Ladezeit erreicht werden soll. Dem CERN 'httpd'-Cache sind sowohl LRU als auch LCP überlegen. Beim LRU-Verfahren beispielsweise liegt die mittlere Zugriffszeit bei einer Cachegröße von 500 MB ca. 6.8% niedriger und die Anzahl der von den Servern übertragenen Bytes liegt um 8.1% niedriger.

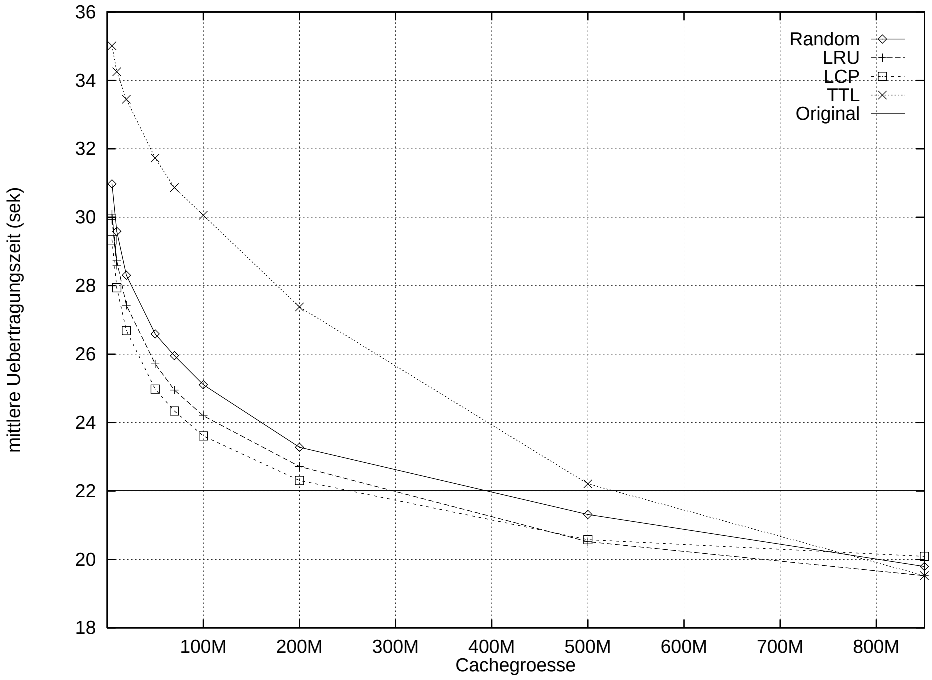


Abbildung 4-35: Mittlere Übertragungszeit

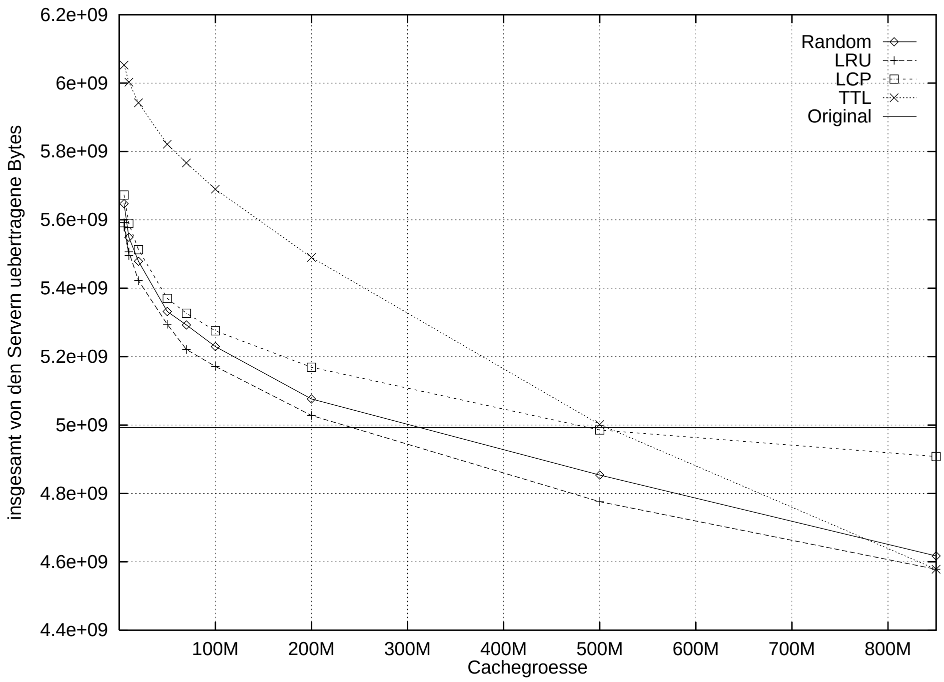


Abbildung 4-36: Insgesamt von den Servern übertragene Bytes

Bemerkenswert ist die gute Performanz des Random-Verfahrens. Liegt es bei kleinen Cachegrößen noch hinter LRU und LCP, sind bei steigendem Speichervolumen zunehmend bessere Werte und damit eine Annäherung an das LRU-Verfahren zu erkennen. In diesem Größenbereich gelingt es dem Random-Verfahren sogar, das LCP-Verfahren bei der mittleren Übertragungszeit zu unterbieten. Interessanterweise ist offensichtlich die Menge der Seiten, die ohne größere Verluste ausgelagert werden können, anteilsmäßig so groß, daß bei einer zufälligen Auswahl einer zu verdrängenden Seite in vielen Fällen keine „wichtigen" Seiten verdrängt werden. Hinzu kommt, daß, je mehr Seiten in dem Cache gespeichert sind, die Wahrscheinlichkeit höher ist, Seiten mit abgelaufenem Expiry-Datum anzutreffen. Da diese Seiten zuerst ausgelagert werden, ist in selteneren Fällen wirklich eine Zufallsauswahl notwendig. Das getestete TTL-Verfahren gibt hier Aufschluß über den tatsächlichen Einfluß des TTL-Faktors: Sobald die Cachegröße eine kritische Grenze von ungefähr 20 MB überschreitet, steigt die Cache-Hit-Rate deutlich an und erreicht mit den anderen Verfahren vergleichbare Werte. Für eine Cachegröße von 850 MB kann das TTL-Verfahren sogar zum LRU-Verfahren aufschließen – mit diesem TTL-Faktor und dieser Cachegröße ist offensichtlich der Einfluß der Verdrängungsstrategie auf die Cacheleistung sehr gering. Weitere Arbeiten müssen zeigen, wie sich ein größerer TTL-Faktor auf die Performanz der Verfahren auswirkt und ob das TTL-Verfahren dann immer noch so gute Resultate liefert.

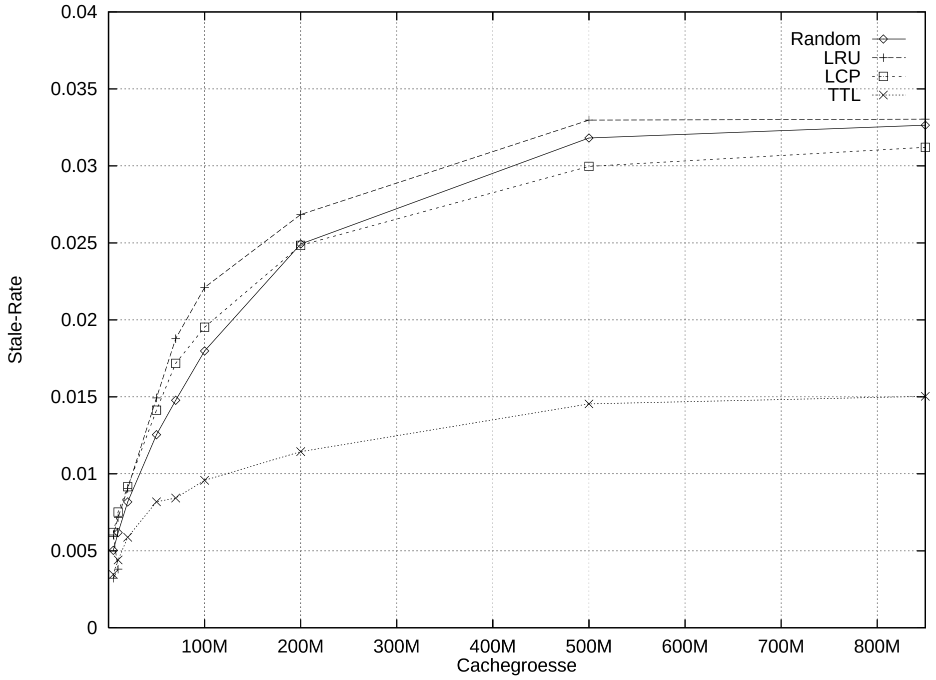


Abbildung 4-37: Stale-Rate

Die Abhängigkeit der Stale-Rate in Abb. 4-37 von der Cachegröße, erfordert ebenfalls noch eine Erklärung: Für das LRU-Verfahren ist stetiges Ansteigen der Stale-Rate bis zu einer Cachegröße von 500 MB zu beobachten; bei dem Übergang zu 850 MB fällt sie jedoch wieder ab. Obwohl also mehr Seiten im Cache gespeichert waren, die potentiell für Zugriffe verantwortlich sein können bei denen veraltete Seiten aus dem Cache geladen werden, sinkt die Zahl der Stale-Zugriffe. Eine mögliche Erklärung dafür sind unterschiedliche Einlagerungszeitpunkte der Seiten und die damit zusammenhängende unterschiedliche Größe des jeweiligen TTL-Zeitfensters. In dem kleineren Cache könnten auf diese Weise durch zwischenzeitliche Auslagerung von Seiten die Zeitfenster bei der Wiedereinlagerung zufälligerweise über Änderungszeitpunkte der Seiten hinweg gelegt worden sein, während dieselben Seiten im größeren Cache erst nach dem Ablauf ihres Zeitfensters und damit evtl. nach der Änderung der Seite neu geladen werden. Der Einfluß, den das verwendete Defaultalter bei Seiten ohne ermittelbares LastModified-Datum hat, muß noch genauer geklärt werden. Möglicherweise ist das beobachtete Kurvenverhalten auf diese Datengrundlage zurückzuführen.

Bemerkung: Die Leistung des LCP-Verfahrens ist stark durch das Größenverhältnis von Puffer und privilegiertem Bereich bestimmt. Wird der Pufferbereich zu klein gewählt, fällt die Anzahl der Cache-Hits entsprechend geringer aus. Wird aber der Puffer zu groß gewählt, werden u.U. Seiten trotz ihres angesammelten Gewinns wieder zu schnell (weil nicht in den privilegierten Bereich aufgenommen) verdrängt. Testläufe haben gezeigt, daß ein entsprechend der Cachegröße angepaßtes Verhältnis gute Ergebnisse bringt. Dabei wird bei einer Cachegröße von 10 MB 28% des Speichers für den privilegierten Bereich gewählt, während bei einer Größe von 850 MB mit ca. 11% gearbeitet wird. Genauere Untersuchungen über ein günstiges Verhältnis zwischen Puffer und privilegiertem Bereich sind jedoch noch notwendig.

4.4.3 Verbesserungsmöglichkeiten

Um die Leistung der Caching-Verfahren zu verbessern, wurden zwei mögliche Ansätze in Betracht gezogen: Die erste getestete Maßnahme umfaßte eine Begrenzung der maximalen Seitengröße für Seiten, die in den Cache eingelagert werden dürfen. Die oft mit dem Einlagern von großen Seiten verbundene Auslagerung von vielen kleinen Seiten soll auf diese Weise vermieden werden. So können mehr Seiten im Cachespeicher gehalten und die Wahrscheinlichkeit für einen Cache-Treffer erhöht werden. Die zweite Maßnahme wurde für das LCP-Verfahren konzipiert. Sie sieht die Einführung einer unteren Gewinnschranke für in den Cache einzulagernde Seiten vor, d.h. nur Seiten mit genügend guten Gewinnaussichten werden in den Cache aufgenommen, die anderen dagegen nicht. Damit soll der störende Einfluß von Seiten abgeschwächt werden, die ohnehin mit fast gleicher Geschwindigkeit von den jeweils anbietenden Servern übertragen werden können und für die sich das Caching aus diesem Grund nicht lohnt. Aus Rechenzeitgründen wurde die Meßreihe nicht mehr in diese Arbeit aufgenommen, sie bleibt Gegenstand zukünfiger Aktivitäten.

Beschränkung der maximalen Seitengröße

Die Abbildungen 4-38 bis 4-41 dokumentieren die Auswirkungen einer Beschränkung der maximalen Größe für in den Cache einzulagernde Seiten auf das LRU-Verfahren. Wie erwartet steigt die Cache-Hit-Rate mit kleiner werdender Seitengrenze deutlich an (bei 500 MB ergibt sich im Vergleich zum LRU-Verfahren ohne Seitengrößenbeschränkung eine Erhöhung um 12,5%). Die Byte-Hit-Rate zeigt aber einen entgegengesetzten Verlauf: Mit immer strikteren Begrenzungen fällt auch sie weiter ab, genauso wie die Gesamtzahl der von den WWW-Servern übertragenen Bytes steigt. Obwohl also mehr Seiten aus dem Cache geladen werden konnten, ist der Anteil der aus dem Cache geladenen Daten gesunken. Die Anteile der großen und nun ausgesperrten Seiten konnten also nicht durch die kleineren egalisiert werden.

Für die mittlere Seitenübertragungszeit hat die Begrenzung der Seitengröße einen positiven Effekt. Bei 850 MB Cachegröße und einer Beschränkung auf maximal 5 MB große Seiten kann z.B. eine Reduzierung um 3% erreicht werden. Dagegen verschlechtert sich die Byte-Hit-Rate und es müssen mehr Daten von den Servern übertragen werden. Es fällt weiterhin auf, daß für eine Cachegröße von 500 MB eine Beschränkung auf 100 KB große Seiten die beste Wahl ist, um eine möglichst niedrige mittlere Seitenladezeit zu erreichen, während für einen größeren Cache eine Erhöhung der Seitenobergrenze auf 5 MB die mittlere Übertragungszeit am weitesten absinken läßt.

Analog stellt sich das Bild beim LCP-Verfahren dar (Abbildungen Abb. 4-42 bis Abb. 4-45 auf den Seiten 106 und 107). Gewinne bei der mittleren Übertragungszeit von ca. 6% sind bei einer Seitenbegrenzung auf 1 MB zu erreichen, die Byte-Hit-Rate läßt sich geringfügig von 20,2% auf 20,4% bei einem 850 MB-Cache steigern und die von den Servern übertragene Datenmenge nimmt für kleinere Cachegrößen durchweg zu – nur bei 850 MB fällt sie leicht ab (minus 0,5%).

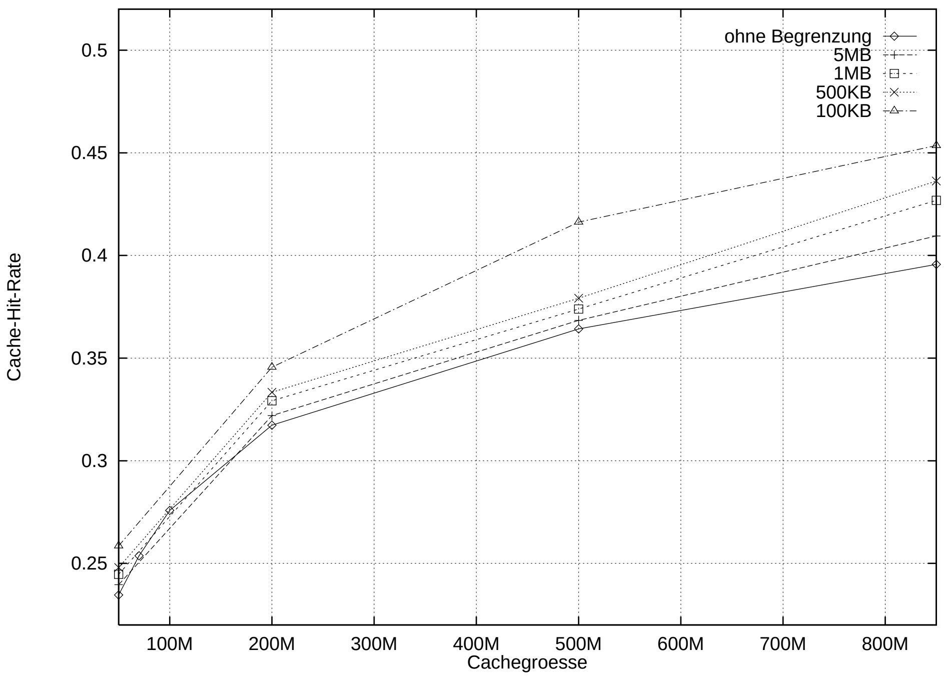


Abbildung 4-38: Cache-Hit-Rate bei beschränkter Seitengröße (LRU)

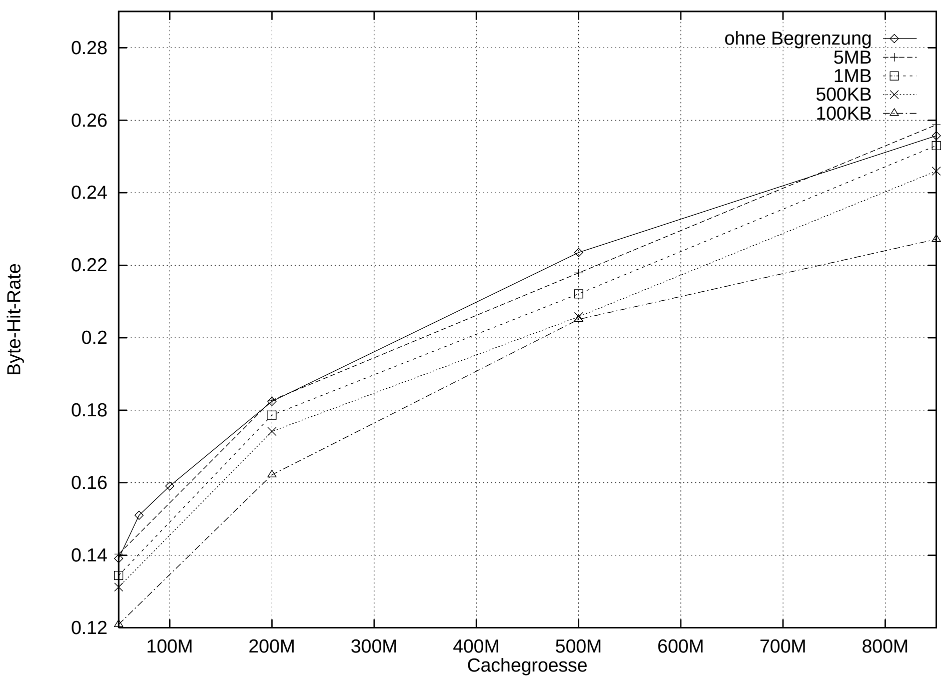


Abbildung 4-39: Byte-Hit-Rate bei beschränkter Seitengröße (LRU)

Insgesamt gesehen ist die Maßnahme, nur Seiten bis zu einer bestimmten Größe in den Cache einzulagern ein wirksames und äußerst einfach zu realisierendes Mittel, um die Cacheleistung bezüglich der durchschnittlichen Seitenladezeiten zu verbessern. Diese Verbesserung geht auf Kosten des nun steigenden Datenaufkommens von den Servern zum Cache und der sinkenden Byte-Hit-Rate.

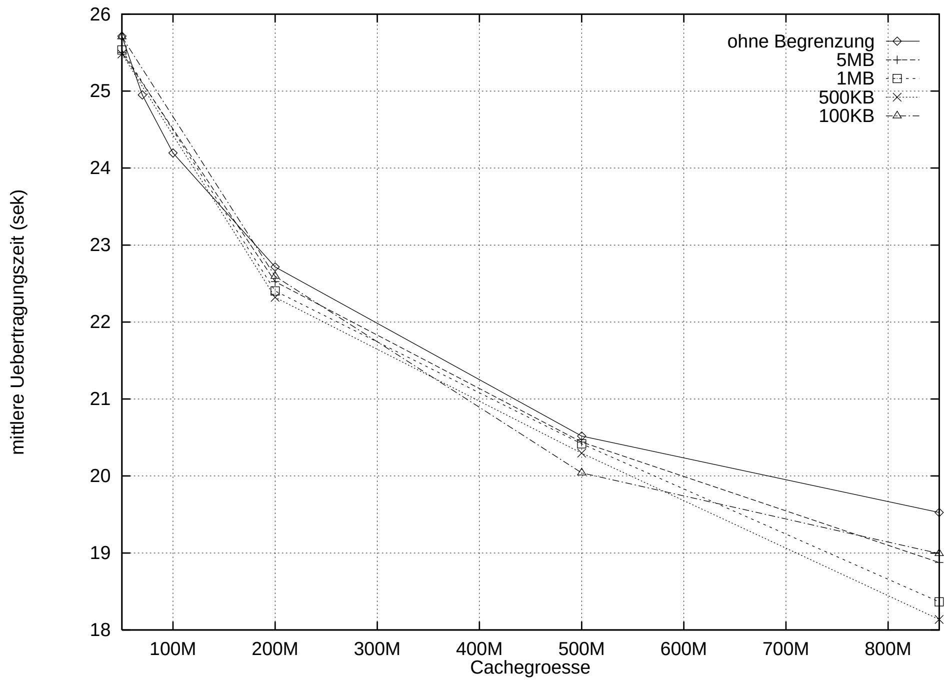


Abbildung 4-40: Mittlere Übertragungszeit bei beschränkter Seitengröße (LRU)

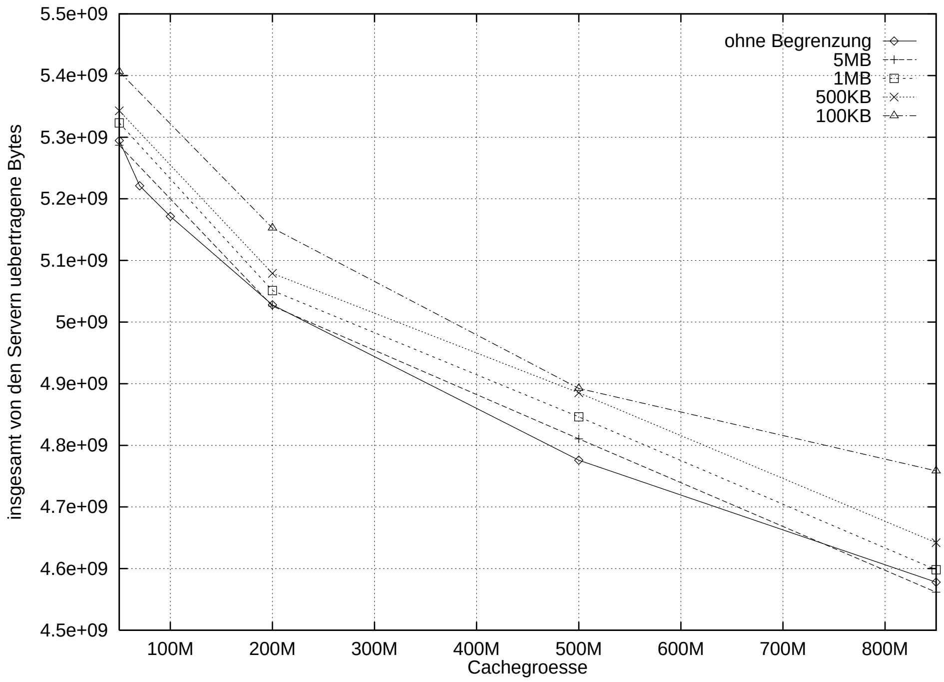


Abbildung 4-41: Insgesamt von Servern übertragene Bytes bei beschränkter Seitengröße (LRU)

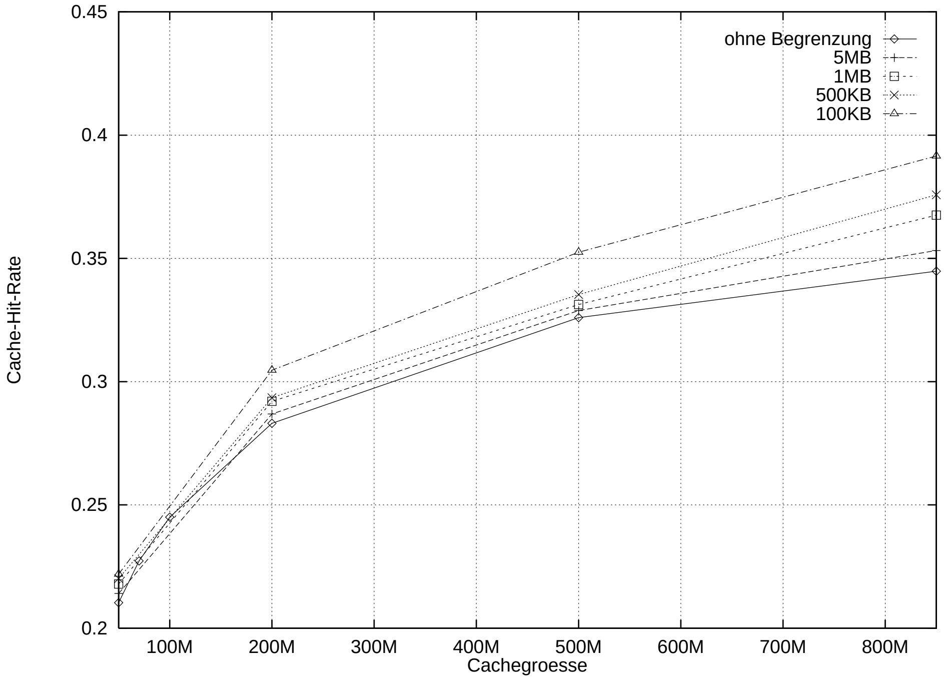


Abbildung 4-42: Cache-Hit-Rate bei beschränkter Seitengröße (LCP)

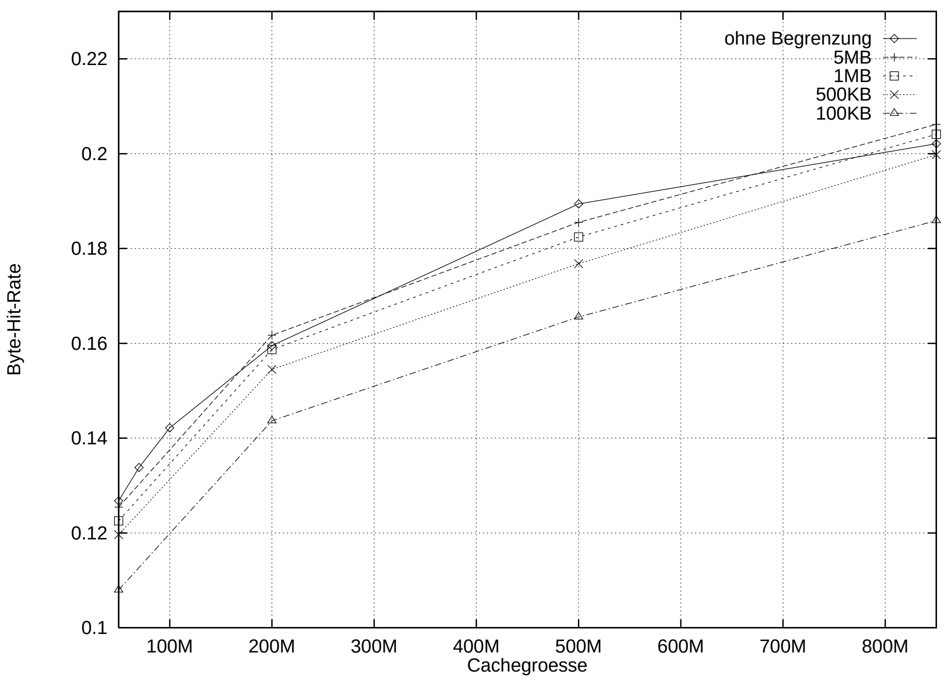


Abbildung 4-43: Byte-Hit-Rate bei beschränkter Seitengröße (LCP)

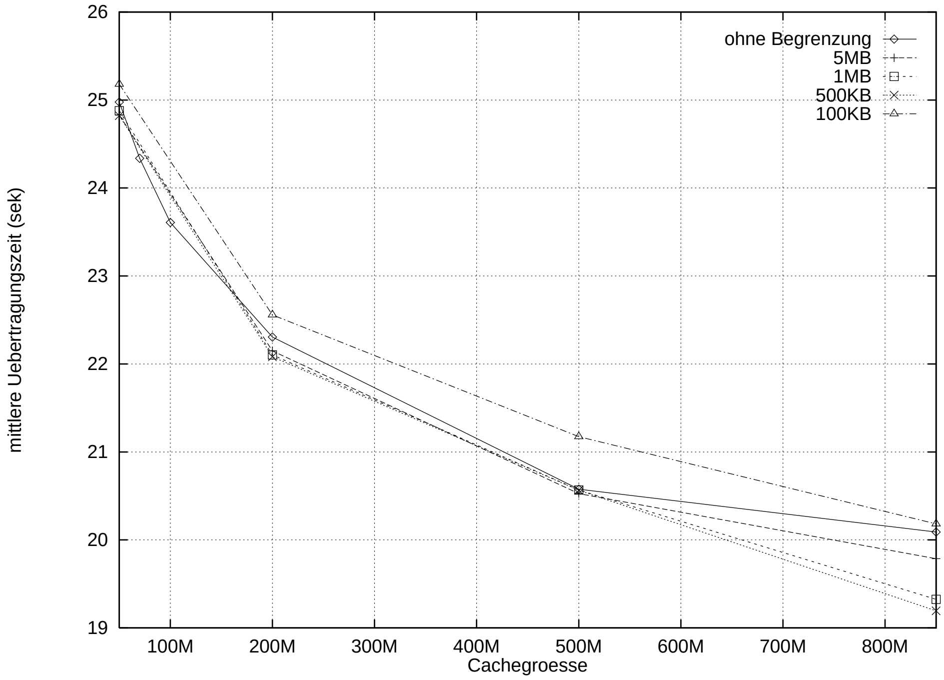


Abbildung 4-44: Mittlere Übertragungszeit bei beschränkter Seitengröße (LCP)

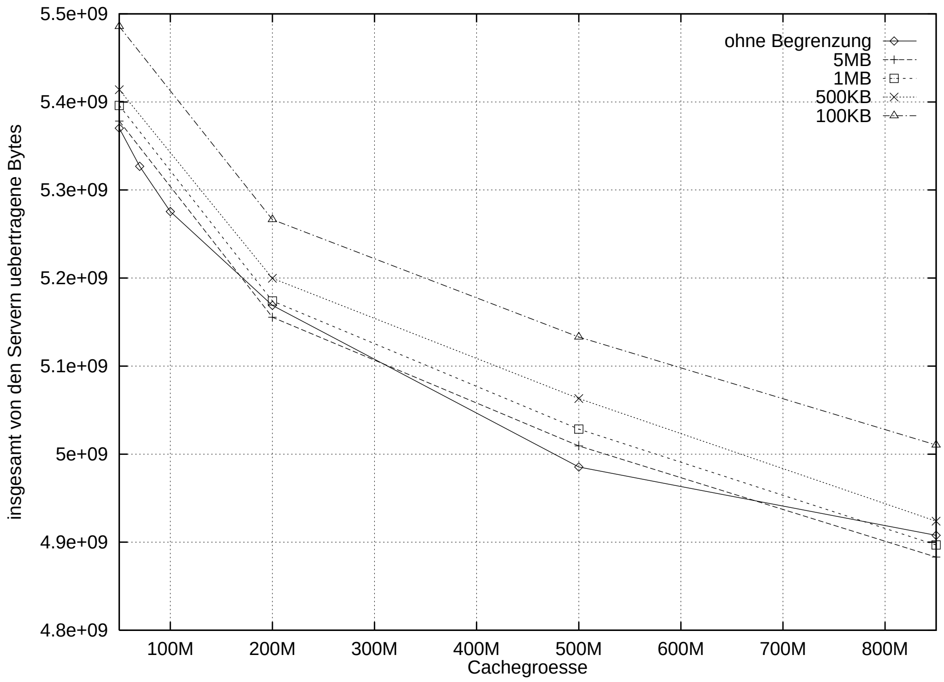


Abbildung 4-45: Insgesamt von Servern übertragene Bytes bei beschränkter Seitengröße (LCP)

4.4.4 Verbesserung durch Replikation

Durch die Einführung eines replizierten Bereichs, in dem häufig verwendete oder besonders wichtige Seiten gehalten werden, soll die Cacheleistung weiter erhöht werden (zusätzlich zu den in Abschnitt 2.4 erwähnten Vorteilen des CgR-Konzepts).

Die Testkonfiguration, mit der die Verbesserungsmöglichkeiten sowohl für das LRU- als auch das LCP-Verfahren untersucht wurden, umfaßte vier verschiedene Größenaufteilungen des zur Verfügung stehenden Speichers in replizierten Bereich und einen den einzelnen Caching-Verfahren zur Verfügung stehenden Cachebereich. Für das Auffüllen des replizierten Bereichs wurde eine Liste der meistreferenzierten WWW-Seiten des Monats April ‘96 verwendet (die Cachesimulation arbeitete mit Zugriffen des Monats Mai ‘96). Graphisch dargestellt sind, wie auch schon bei den vorangegangenen Analysen, die vier Bewertungsmaße Cache- und Byte-Hit-Rate, mittlere Übertragungszeit und insgesamt von den Servern übertragene Bytes (Abbildungen 4-46 bis 4-49 für das LRU-Verfahren und Abbildungen 4-50 bis 4-53 ab Seite 112 für das LCP-Verfahren). Die exakten Werte der Testreihen können den Tabellen des Anhangs ab Seite 119 entnommen werden.

Am stärksten wird beim LRU-Verfahren die Byte-Hit-Rate von der Replikation beeinflußt. Verbesserungen um bis zu 9,2% sind bei einer Gesamtgröße von 500 MB (davon 40% repliziert) erreichbar. Übereinstimmend für alle vier dargestellten Maße ist ab einer Cachegröße von ca. 300 MB eine günstigere Cacheleistung sowohl in Bezug auf mittlere Übertragungszeit, als auch in Bezug auf das Datenaufkommen von den Servern zu verzeichnen. Verbesserungen um ca. 1,5% bei den durchschnittlichen Seitenladezeiten und eine 3,8%ige Reduzierung der übertragenen Bytes von den WWW-Servern sind bei einer Cachegröße von 500 MB und einem Replikationsfaktor von 40% zu vermerken. Mit zunehmender Cachegröße kristallisiert sich der 5%ige Replikationsanteil als am vorteilhaftesten heraus. Daß eine Einrichtung eines replizierten Bereichs mit 40% des zur Verfügung stehenden Speichers nicht die optimalen Verbesserungen bringt, ist ein Hinweis darauf, daß bei einer zu starken Verkleinerung des dem LRU-Verfahren verfügbaren Speicheranteils die dadurch erlittenen Verluste auch nicht mehr über die Seiten des replizierten Bereichs abgefangen werden können. Eine genaue Erklärung für das Abflachen der Kurven für die Replikationsanteile von 10 und 40 Prozent konnte im Rahmen dieser Arbeit nicht gefunden werden. Dieser Punkt muß deshalb Gegenstand einer nachfolgenden genaueren Untersuchung sein.

Für das LCP-Verfahren ist die im Vergleich zu LRU deutlichere Verbesserung zu erkennen, wenngleich das LRU-Verfahren seine führende Position, was Byte-Hit-Rate und von den Servern übertragene Bytes nicht verliert. Die Replikation bewirkt ein Ansteigen der Cache- bzw. Byte-Hit-Rate um ca. 10,3 % bei einer Cachegröße von 850 MB und einer Replikationsrate von 40%, die mittlere Übertragungszeit kann ab 500 MB deutlich verbessert werden (Reduzierung um 5% bei einer Replikationsrate von 40%). Bei gleicher Replikationsrate und 850 MB Cachespeicher, ist für das Aufkommen von WWW-Servern geladener Daten eine Reduzierung um 2,7% zu vermerken.

Im Gegensatz zum LRU-Verfahren stellt die Reservierung von 40% des Speichers für die Replikation in Verbindung mit der Eigenschaft des LCP-Verfahrens, besser mit Speichergrößen unter 500 MB zu arbeiten, die günstigste von den getesteten LCP-Aufteilungen dar, allerdings ohne dabei besser als das LRU-Verfahren zu werden.

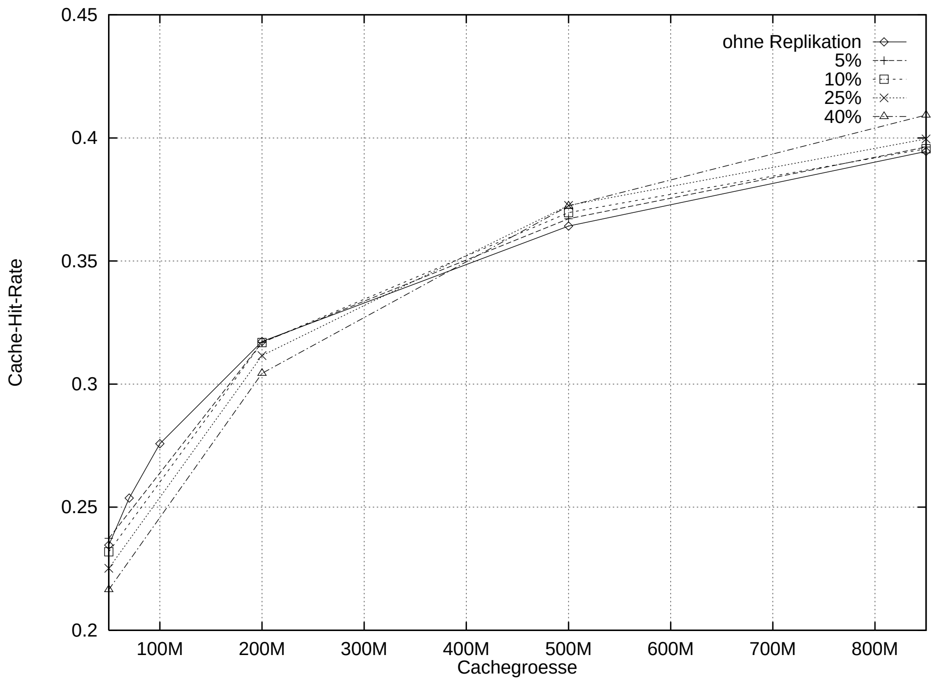


Abbildung 4-46: Cache-Hit-Rate in Verbindung mit Replikation (LRU)

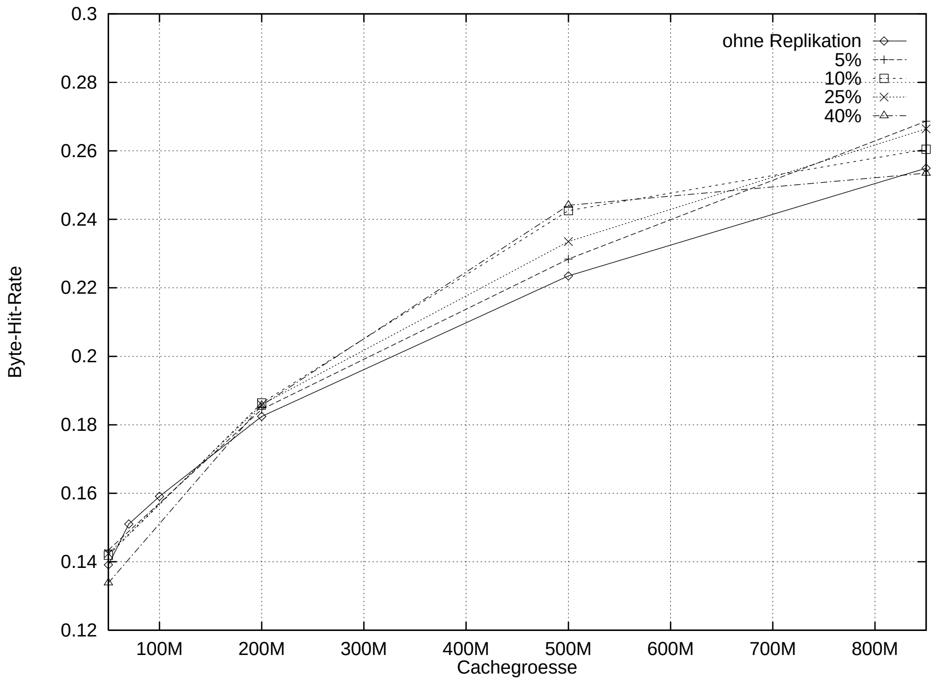


Abbildung 4-47: Byte-Hit-Rate in Verbindung mit Replikation (LRU)

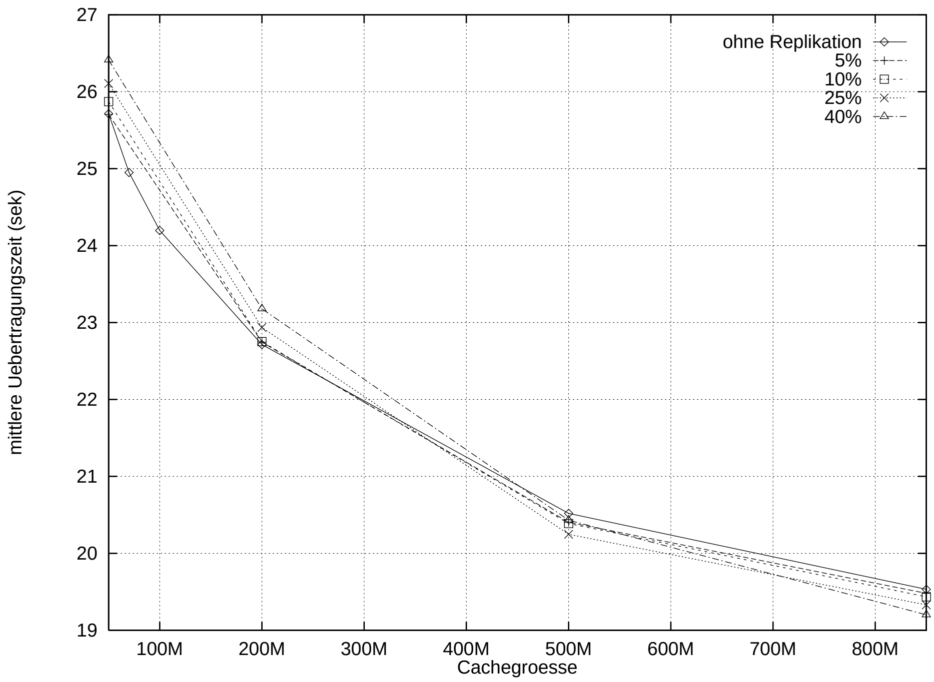


Abbildung 4-48: Mittlere Übertragungszeit in Verbindung mit Replikation (LRU)

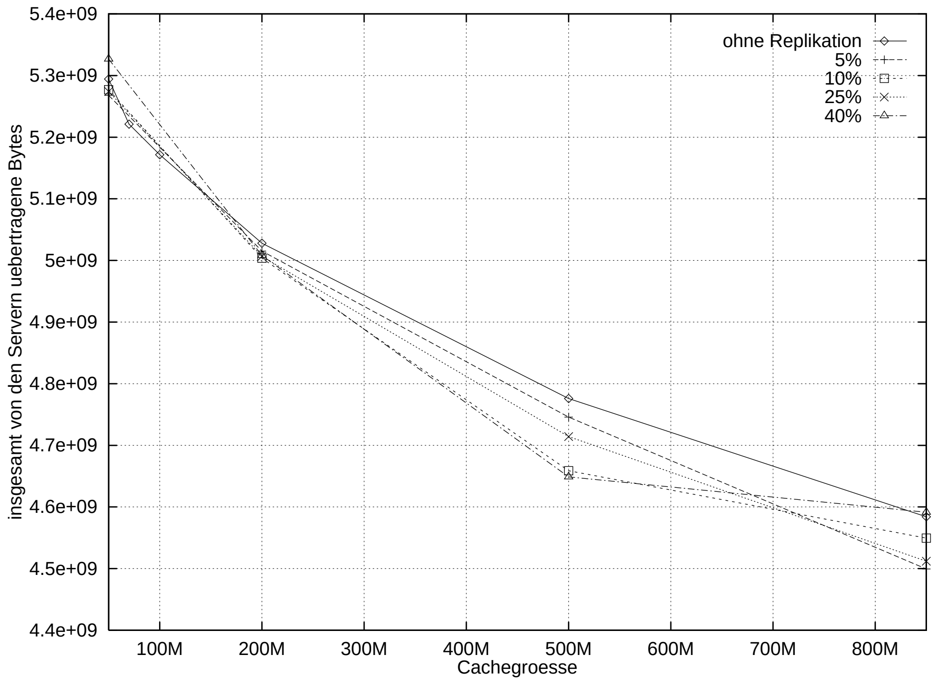


Abbildung 4-49: Insgesamt von Servern übertragene Bytes in Verbindung mit Replikation (LRU)

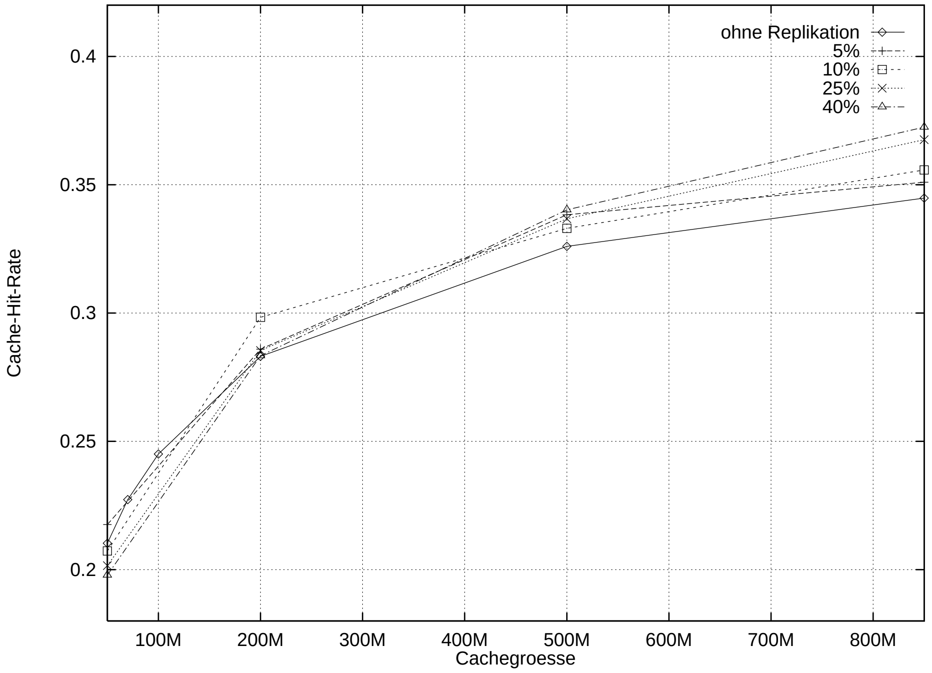


Abbildung 4-50: Cache-Hit-Rate in Verbindung mit Replikation (LCP)

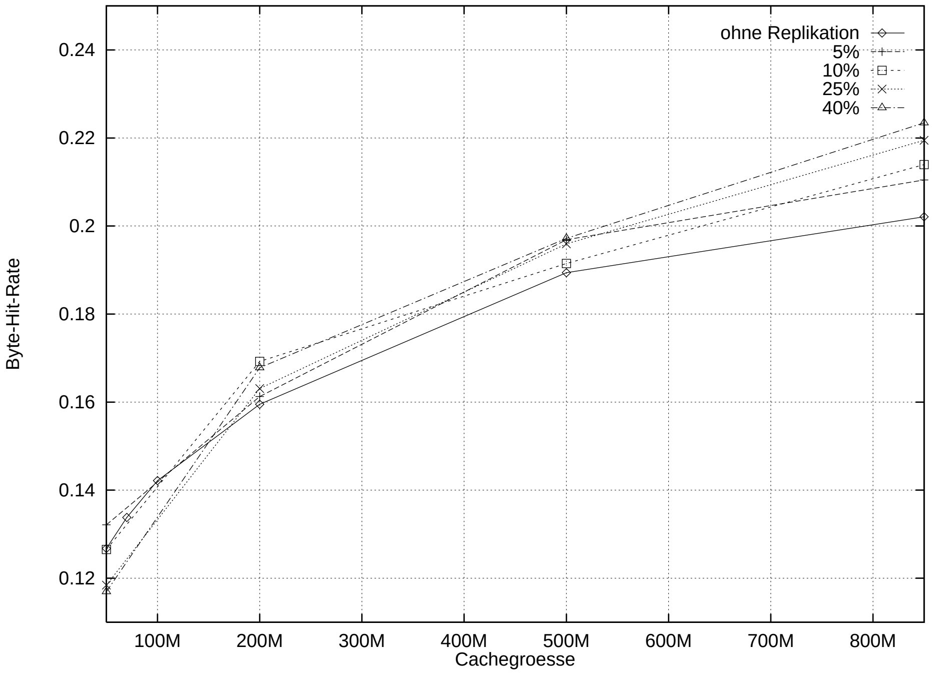


Abbildung 4-51: Byte-Hit-Rate in Verbindung mit Replikation (LCP)

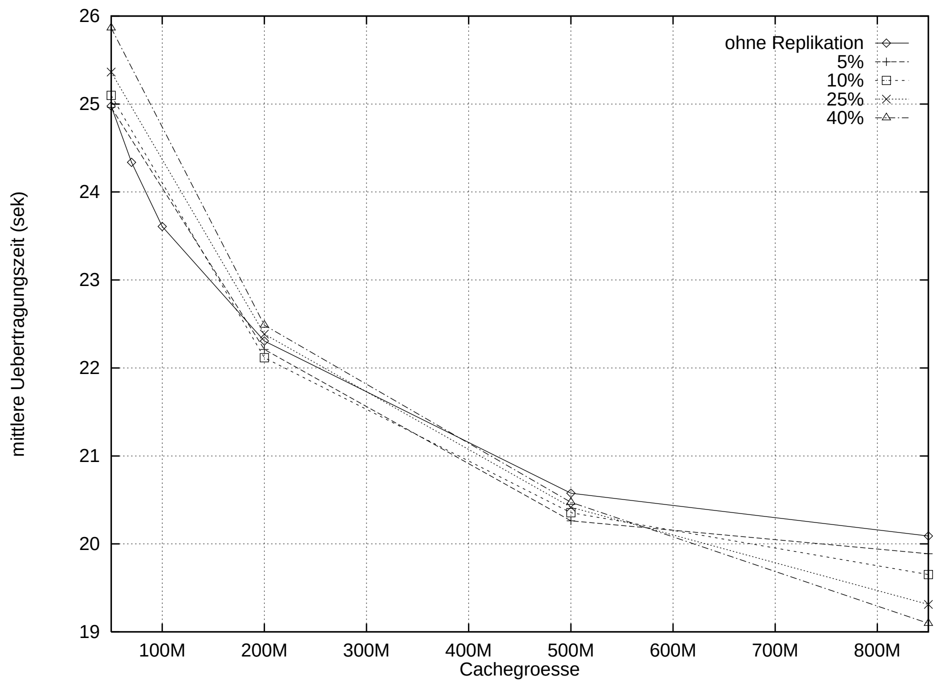


Abbildung 4-52: Mittlere Übertragungszeit in Verbindung mit Replikation (LCP)

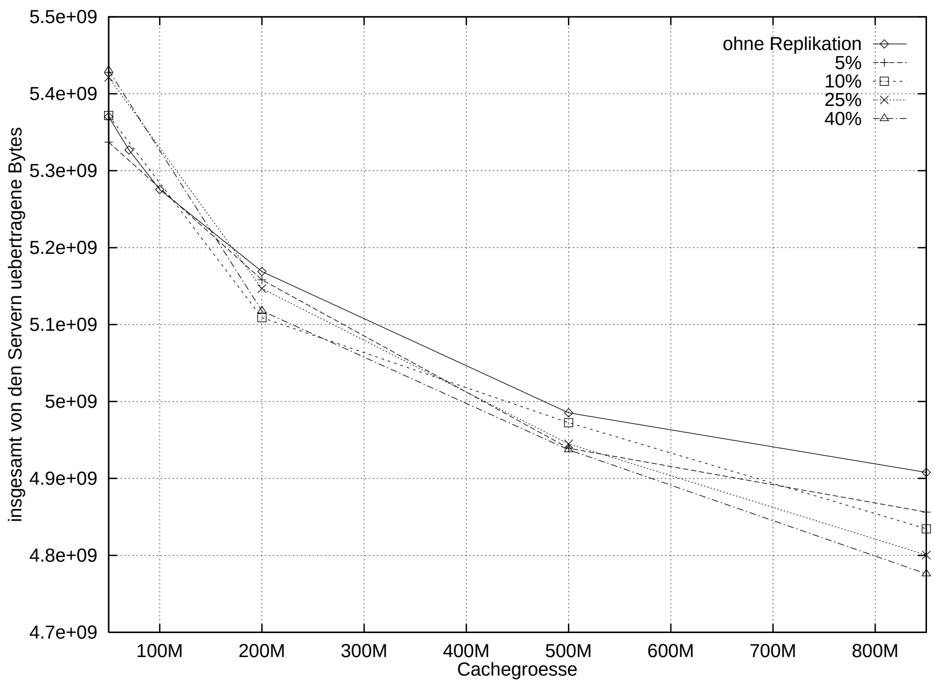


Abbildung 4-53: Insgesamt von Servern übertragene Bytes in Verbindung mit Replikation (LCP)

5. Zusammenfassung und Ausblick

Mit den an der Universität Kaiserslautern aufgezeichneten Logfiles, die das Zugriffsverhalten auf die WWW-Server in aller Welt festhalten, steht eine ausgezeichnete Basis für ein breites Spektrum von Analysen zur Verfügung. Mit dem Analyse- und Cachesimulationstool CaSimiR wird ein Schritt zur besseren Auswertung der Daten gemacht. Für diese Arbeit wurden u.a. von der Analysekomponente von CaSimiR die Informationen über die Verbindungen zu anderen WWW-Servern aufbereitet. Untersuchungen über die tageszeitabhängige Veränderung von zur Verfügung stehender Bandbreite und zur Vorhersage von Seitenübertragungszeiten konnten auf der Basis dieser Daten durchgeführt werden. Übereinstimmend für alle Übertragungen von WWW-Servern hat sich eine ausgeprägte Abhängigkeit der gemessenen Bandbreite von der jeweiligen Tageszeit, an der die Übertragungen stattfanden, gezeigt. Während die Hauptaktivität bei Zugriffen auf WWW-Server in die Zeit von 8 bis 21 Uhr fällt und in diesem Zeitraum ein spürbarer Rückgang der zur Verfügung stehenden Bandbreite zu vermerken ist, zeichnen sich die Nacht- und frühen Morgenstunden durch die verbesserte Verbindungsqualität mit höherer Bandbreite und niedrigerer Zugriffsaktivität aus. Die Vorteile, vor allem bei den häufig referenzierten Servern der `com`-Domäne, macht die Nacht und den frühen Morgen zu einem idealen Zeitraum, um beispielsweise replizierte Seiten zu aktualisieren. So könnte gleichzeitig die hohe Bandbreite genutzt und die Belastung in die weniger frequentierten Nachtstunden verlagert werden. Weitere Beobachtungen haben gezeigt, daß auch eine Abhängigkeit der Bandbreite von Entfernung und Anbindung der WWW-Server existiert. Deutliche Unterschiede beispielsweise sind zwischen Servern aus Deutschland und Nordamerika gegenüber denen aus Asien oder Australien zu erkennen.

Der zweite die Übertragungszeiten betreffende Aspekt beschäftigte sich mit der Schätzung von Ladezeiten für WWW-Seiten. Die drei vorgestellten Schätzmodelle Average, StraightLine und Simple-StraightLine wurden auf ihre Genauigkeit hin mit folgendem Ergebnis getestet: Das Average-Modell hat die Tendenz, Übertragungszeiten für kleine Seiten zu überschätzen, dagegen aber die von großen Seiten zu unterschätzen. Insgesamt ist eine geringere Störanfälligkeit gegenüber zufälligen Schwankungen von Meßwerten bei der Übertragungsprofil-Generierung zu erkennen. Durch das StraightLine-Modell können Übertragungszeiten für kleinere Seiten präziser vorhergesagt werden. Im Gegenzug sorgt die höhere Störanfälligkeit gegenüber Schwankungen der Meßwerte zu einer häufigeren Überschätzung von Ladezeiten großer Seiten. Die speicheraufwendigere Neuberechnung der für die Schätzung benötigten Übertragungsprofile begünstigt zudem nicht den Einsatz in einem beispielsweise CgR-verbesserten Proxy-Cache, bei dem Übertragungsprofile schritthaltend aktualisiert werden sollten. Aufgrund der einfacheren, ressourcensparenderen Berechnung der Übertragungsprofile bietet sich das Simple-StraightLine-Modell an. Ungefähr gleiche Exaktheit bei der Schätzung von Übertragungszeiten für kleinere Seiten und eine ausgeglichenere Verteilung von positiven und negativen Schätzfehlern steht dabei eine insgesamt größere Schätzungenauigkeit bei größeren Seiten gegenüber. Für das Average-, StraightLine bzw. das Simple-StraightLine-Modell lagen 96,8%, 96,5% bzw. 94,7% aller Schätzfehler in einem Intervall von -200 bis 200 Sekunden. Wird das Intervall auf -50 bis 50 Sekunden verkleinert, sind noch 86%, 88% bzw. 86,7% der geschätzten Zeiten innerhalb des Toleranzbereichs.

Der abschließende Teil der Arbeit beschäftigte sich mit dem Caching im World-Wide Web. Hier zeigte sich zunächst, daß eine Cachebewertung nach klassischen Maßstäben (Cache-Hit-Rate) nicht mehr ausreichend ist. An die Stelle der Cache-Hit-Rate treten Maße wie Byte-Hit-Rate, mittlere Übertragungszeit und von den Servern übertragenes Datenvolumen. Mit den simulierten Anfragen des Monats Mai ‘96 wurden dann die Caching-Verfahren TTL, Random, LCP und LRU auf diese Leistungsmerkmale hin untersucht. Prinzipiell gilt für alle Verfahren, daß sich bei wachsendem Cache auch die Leistungswerte verbessern. Lediglich Stale- und 304-Reply-Rate verschlechtern sich mit größer werdendem Cachevolumen. Im Testfeld, insbesondere bei einer Cachegröße von über 500 MB, setzte sich das LRU-Verfahren an die Spitze. Bei der Byte-Hit-Rate sowie der Anzahl der von den WWW-Servern übertragenen Bytes liegt das LRU-Verfahren bei allen getesteten Cachegrößen vor den anderen Verfahren. Lediglich bei der erreichten mittleren Übertragungszeit liegt das LCP-Verfahren mit bis zu 2,8% vor ihm. Die Verfahren TTL und Random[1] erreichten für die drei o.g. Bewertungsmaße die zweitbesten Werte, gefolgt vom LCP-Verfahren, das seine Stärken bei der mittleren Übertragungszeit und kleineren Cachegrößen hat. Die gute Performanz des TTL-Verfahrens, die nur bei einem Cachevolumen von 850 MB erreicht wurde, dokumentiert, daß bei genügend großem Cachevolumen offenbar so viele ungültige Seiten mit abgelaufenem Expiry-Datum im Cache gelagert sind, daß eine Verdrängungsstrategie in immer selteneren Fällen über eine Auslagerung noch gültiger Seiten entscheiden muß.

Zur Verbesserung der Bewertungsmaße wurde eine Beschränkung der maximalen Größe von einzulagernden Seiten untersucht. Die Beschränkung hat sich als wirksame, einfach zu realisierende Maßnahme erwiesen, um die mittlere Übertragungszeit zu verbessern. Eine Reduzierung um 3% für das LRU-Verfahren (850 MB Cachegröße, 5 MB Seitenbegrenzung) und ca. 6% für das LCP-Verfahren (850 MB, 1 MB Seitenbegrenzung) kann beobachtet werden. Insbesondere bei Cachegrößen unter 850 MB und einer strikten Beschränkung der Seitengröße (100 KB) verschlechtern sich die Werte für Byte-Hit-Rate und die von den Servern übertragene Byteanzahl um bis zu 13,6% bzw. 3,8% für das LRU-Verfahren und um 8,4% bzw. 2% für das LCP-Verfahren.

Wird ein Teil des Cachespeichers für die Replikation von Seiten[2] verwendet, ergeben sich durchweg Verbesserungen für Byte-Hit-Rate, mittlere Übertragungszeit und für das von den Servern übertragene Datenvolumen. Für das LRU-Verfahren ergeben sich bei einem 500 MB-Cache und einem 40%igen Replikationsanteil folgende maximale Verbesserungen: Byte-Hit-Rate +9,2%, mittlere Übertragungszeit: -3,8% und Anzahl der übertragenen Bytes: -1,5%. Es zeigte sich, daß für einen 500 MB-Cache eine 40%-Replikationsrate optimal ist, während für einen 850 MB-Cache dagegen ein Replikationsfaktor von 5% die beste Wahl darstellte.

Das ebenfalls untersuchte LCP-Verfahren kam auf folgende Werteverbesserungen durch die Einrichtung eines replizierten Bereiches, der 40% des Cachespeichers ein-

1. Eine zufällige Verdrängung wird hier nur dann vorgenommen, falls sich keine Seiten mit abgelaufenem Expiry-Datum mehr im Cache befinden. Gleiches gilt für die Verdrängung mit der LRU-Strategie.

2. Der replizierte Bereich wurde mit den meistreferenzierten Seiten des Monats April ‘96 gefüllt (die Simulation selbst wurde für den Monat Mai ‘96 durchgeführt).

nimmt: Byte-Hit-Rate: +10,3 % bei einer Cachegröße von 850 MB, mittlere Übertragungszeit: -5% für einen 500 MB-Cache, und für einen 850 MB großen Cachespeicher ist für das Aufkommen der von den WWW-Servern geladenen Daten eine Reduzierung um 2,7% zu vermerken. Die Replikationsrate von 40% erweist sich beim LCP-Verfahren als die günstigste.

Weiteren Arbeiten bleibt es überlassen noch ausstehende Fragen zu beantworten und weitere Erkenntnisse aus diesem Themengebiet zu erbringen. Die folgenden Punkte sollen mögliche weitere Ansatzpunkte dazu aufzeigen:

- Für die Analyse der Bandbreiten könnte eine Gruppierung nach ihrer Entfernung (beispielsweise gemessen in Hops) differenziertere Aussagen über die Entfernungsabhängigkeit der zur Verfügung stehenden Bandbreite ermöglichen.
- Mit Hilfe von Zugriffsinformationen aus anderen Monaten oder anderen Betriebsumgebungen, z.B. eines Backbones, sollten die gemachten Erkenntnisse überprüft und sichergestellt werden, inwieweit die dargestellten Kurvenverläufe charakteristische Eigenschaften des WWW-Cachings wiedergeben oder aber nur auf spezifische Zugriffsmuster des hier simulierten Monats oder Servers zurückzuführen sind.
- Der Einfluß des verwendeten Defaultalters für WWW-Seiten ohne ermittelbares Last-Modified-Datum auf die Meßdaten muß genauer untersucht werden. In diesem Zusammenhang sollten auch die Auswirkungen der Übertragungszeitenschätzung mit berücksichtigt werden. Eventuell könnten neue, aus den bisherigen drei Schätzmodellen aufgebaute Verfahren die quantitative Aussagekraft der Simulation verbessern.
- Ob Verbesserungsmöglichkeiten mit einer unteren Schranke für Gewinne beim Einlagern von Seiten in den Cache zu erreichen sind, sollte für alle Verfahren in weiteren Meßreihen untersucht werden. Der TTL-Faktor sollte gleichfalls Gegenstand weiterer Untersuchungen sein. Möglicherweise sind durch individuell auf einzelne Verfahren abgestimmte Faktoren noch deutliche Gewinne zu erzielen. Die Suche nach einem günsigen, evtl. cachegrößen-abhängigen Verhältnis von Puffer und privilegiertem Bereich des LCP-Verfahrens ist ebenfalls im Zusammenhang mit den Verbesserungsanstrengungen zu sehen.
- Die im Verlauf der Testreihen aufgetretenen unerwarteten Kurvenverläufe (abnehmende Stale-Rate des LRU-Verfahrens bei Vergrößerung des Cachespeichers; Ausfall der durch Replikation erbrachten Verbesserungen beim LRU-Verfahren bei einer Cachegröße von 850 MB und den Replikationsraten 10% und 40%) erfordern eine genauere Untersuchung.
- In Verbindung mit der Replikation müssen weitere Heuristiken zur Bestimmung der zu replizierenden Seiten gefunden werden. Möglicherweise müssen dazu Informationen aus mehren vorangegangenen Monaten mit einbezogen werden oder z.B. Strukturinformationen von HTML-Dokumenten zur besseren Vorhersagbarkeit von Zugriffen mit in die Auswahl einfließen.
- Die Erweiterung von CaSimiR für die Simulation einer frei konfigurierbaren CgR-Konstellation mit Primary-, Replicated-, WLIS- und Proxy-Servern könnte Aufschlüsse über Verbesserungen für u.U. mehrere beteiligte Proxy-Caches bringen und Hinweise auf günstige Orte zur Einrichtung von replizierten Servern geben.

6. Literaturverzeichnis

[BAE96] M. Baentsch, G. Molter, P. Sturm: *Introducing Application-level Replication and Naming into today´s Web;* Proceedings International World-Wide Web Conference, Mai 96, Paris

[BEL66] L.A. Belady: *A Study of Replacement Algorithms for a Virtual Storage Computer*, IBM Systems Journal, Vol. 5, No. 2, 78-101 (1966)

[BER96] T. Berners-Lee, R. Fielding, H. Frystyk, J.C. Mogul, J. Gettys: *Hypertext Transfer Protocol – HTTP/1.1*; Online-Dokument, August 1996 verfügbar über "http://www.w3.org/pub/WWW/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-07.ps"

[CAT95] L.D. Catledge, J.E. Pitkow: *Characterising Browsing Strategies in the World-Wide Web*; Proceedings Third International World-Wide Web Conference, April 1995, Darmstadt

[CHA95] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, K.J. Worrel: *A Hierarchical Internet Object Cache;* Technical Report CU-CS-766-95. Department of Computer Science, Universiy of Colorado, Boulder, Colorado, März 1995

[GWE96] J.S. Gwertzman, M. Seltzer: *WWW Cache Consistency*; Proceedings of the 1996 Usenix Technical Conference, San Diego, Kalifornien, Januar 1996

[DEN70] P.J. Denning: *Virtual Memory*, ACM-Computing Surveys, Vol. 2, No. 3, 153-189 (1970)

[LUO94] A. Luotonen, K. Altis: *World-Wide Web Proxies*; Proceedings First International Conference on the Worl-Wide Web, Genf, Schweiz, Mai 1994

[NEH93] J. Nehmer: *Bertiebsorganisation von Rechnersystemen – eine Einführung am Beispiel des Client/Server-Modells*; Skript zur Vorlesung Betriebssysteme, Oktober 1993

[STR92] B. Stroustrup: *Die Programmiersprache C++*; Addison-Wesley, 2.Auflage, 1992

[W3C] *Names and Adresses, URIs, URLs, URNs, URCs*; Internet-Draft, verfügbar über "http://www.w3.org/pub/WWW/Adressing"

[WIL93] T. Williams, C. Kelley: *GNUPLOT, An Interactive Plotting Program*; GNU-Projekt, "http://www.gnu.org/"

[GZIP] *gzip-Komprimierungstool*; GNU-Projekt, "http://www.gnu.org/"

7. Angang

A. Meßwerte der Cacheauswertung

Es folgt eine tabellarische Auflistung der durchgeführten Messreihen:

- Random-Verfahren (TTL-Faktor: 10%): Seite 120.
- TTL-Verfahren (TTL-Faktor: 10%): Seite 121.
- LRU-Verfahren (TTL-Faktor: 10%): Seite 122.
- LCP-Verfahren (TTL-Faktor: 10%): Seite 123.
- LRU mit Replikation (TTL-Faktor: 10%): Seite 124.
- LRU mit Begrenzung der Seitengröße (TTL-Faktor: 10%): Seite 125.
- LCP mit Replikation (TTL-Faktor: 10%): Seite 126.
- LCP mit Begrenzung der Seitengröße: Seite 127.

Bewertungsmaße für das Random-Verfahren (TTL-Faktor: 10%)

Cachegröße	mittl. Zugriffszeit	Cache-Hit-Rate	Byte-Hit-Rate	Stale-Rate	304-Reply-Rate	von Servern übertr. Bytes
5000000	30.975	0.127726	0.0816901	0.00503266	0.000367301	5647413247
10000000	29.5896	0.15103	0.0976955	0.00618128	0.000594447	5548997809
20000000	28.3039	0.17745	0.109073	0.00817405	0.000771653	5479042991
50000000	26.5895	0.217961	0.133004	0.0125446	0.00101652	5331883711
70000000	25.9536	0.234265	0.139402	0.0147742	0.00117117	5292546180
100000000	25.1046	0.254362	0.149648	0.0179832	0.00114056	5229538004
200000000	23.2776	0.301006	0.174599	0.0249281	0.00193477	5076141753
500000000	21.3125	0.350666	0.210818	0.0318166	0.00602824	4853656270
850000000	19.7927	0.386595	0.249506	0.0326479	0.0259269	4616970997
1000000000	19.0709	0.400303	0.263676	0.0325303	0.0419077	4530824054

Bewertungsmaße für das TTL-Verfahren (TTL-Faktor: 10%)

Cachegröße	mittl. Zugriffszeit	Cache-Hit-Rate	Byte-Hit-Rate	Stale-Rate	304-Reply-Rate	von Servern übertr. Bytes
5000000	36.5653	0.00910036	0.00110421	0.00111318	0	6142974898
10000000	36.6385	0.00466375	0.000409229	0.000478457	0	6147248890
20000000	33.8025	0.1545	0.0310131	0.0156264	2.73864e-05	5959044224
50000000	32.2134	0.204554	0.050698	0.0197537	0.000112768	5837991930
70000000	31.2742	0.223211	0.0605115	0.0212905	0.000154653	5777643388
100000000	30.3843	0.243857	0.0728618	0.0224811	0.000243256	5701697617
200000000	27.5146	0.288922	0.107187	0.0265552	0.000860257	5490642539
500000000	22.4249	0.350857	0.184396	0.0321549	0.0055079	5016115025
850000000	19.5307	0.39449	0.254971	0.032163	0.0364046	4584015707

Bewertungsmaße für das LRU-Verfahren (TTL-Faktor: 10%)

Cachegröße	mittl.Übertragungs-zeit	Cache-Hit-Rate	Byte-Hit-Rate	Stale-Rate	304-Reply-Rate	von Servern übertr. Bytes
5000000	30.0924	0.140156	0.0907073	0.00602663	0.00053323	5591970036
10000000	28.7278	0.1647	0.104576	0.00715753	0.000679828	5506687289
20000000	27.4282	0.19112	0.118285	0.00905364	0.000839314	5422395603
50000000	25.7149	0.234608	0.139098	0.0149321	0.000960137	5294402305
70000000	24.9499	0.253724	0.151026	0.0187823	0.00119373	5221067797
100000000	24.1967	0.275793	0.159086	0.0221089	0.00148531	5171517417
200000000	22.7149	0.317293	0.182436	0.0268452	0.00251794	5027984673
500000000	20.5191	0.364208	0.223502	0.0329781	0.0105245	4775935099
850000000	19.5311	0.394532	0.254951	0.0318553	0.0364063	4584137981
1000000000	18.6129	0.412759	0.272349	0.0320276	0.055319	4478316545

Bewertungsmaße für das LCP-Verfahren (TTL-Faktor: 10%)

Cachegröße	mittl. Zugriffszeit	Cache-Hit-Rate	Byte-Hit-Rate	Stale-Rate	304-Reply-Rate	von Servern übertr. Bytes
5000000	29.3381	0.129349	0.077596	0.00619095	0.00039952	5672593033
10000000	27.9387	0.15136	0.091072	0.00751677	0.000547729	5589728216
20000000	26.6867	0.172029	0.103547	0.00916318	0.000563839	5513009592
50000000	24.9767	0.210318	0.126771	0.0141362	0.00073299	5370198823
70000000	24.3373	0.22728	0.133807	0.0171794	0.000985912	5326947037
100000000	23.608	0.24506	0.142172	0.0195217	0.00112607	5275509676
200000000	22.3066	0.283124	0.159514	0.0248411	0.00170601	5168896954
500000000	20.5769	0.325976	0.18941	0.0299737	0.00654375	4985342795
850000000	20.0895	0.344802	0.202092	0.0312093	0.0137045	4907799921
1000000000	19.6544	0.357359	0.213869	0.0311642	0.0252986	4836092706

Bewertungsmaße: LRU mit Replikation (TTL-Faktor: 10%)

Cachegröße	Replikationsrate	mittl.Übertragungszeit	Cache-Hit-Rate	Byte-Hit-Rate	Stale-Rate	304-Reply-Rate	von Servern übertr. Bytes
50000000	0.05	25.7059	0.237339	0.143412	0.0125543	0.0008667	5267870280
200000000	0.05	22.7451	0.316913	0.184526	0.026328	0.00215225	5015110264
500000000	0.05	20.4037	0.367181	0.228405	0.0329056	0.00945316	4745715722
850000000	0.05	19.4806	0.396277	0.26868	0.0317731	0.0316152	4499410051
50000000	0.1	25.8727	0.231908	0.14191	0.0134129	0.000915029	5277110743
200000000	0.1	22.7524	0.316863	0.186367	0.0261621	0.00200082	5003776998
500000000	0.1	20.3872	0.369664	0.242557	0.032664	0.0100992	4658724518
850000000	0.1	19.4349	0.395618	0.26046	0.031786	0.0239583	4549485686
50000000	0.25	26.1077	0.2252	0.142384	0.0124157	0.000847369	5274187941
200000000	0.25	22.9365	0.311587	0.185907	0.0250119	0.00192188	5006600939
500000000	0.25	20.2466	0.372609	0.233493	0.0311207	0.00736695	4714297345
850000000	0.25	19.329	0.399599	0.266452	0.0305601	0.0143714	4512037289
50000000	0.4	26.4144	0.216495	0.133843	0.0111318	0.000784541	5326708452
200000000	0.4	23.1779	0.304362	0.185661	0.0224118	0.00179301	5008105916
500000000	0.4	20.4308	0.372367	0.244106	0.0290619	0.00509388	4648888818
850000000	0.4	19.2018	0.409263	0.25349	0.0173292	0.00872339	4591404099

Bewertungsmaße: LRU mit Begrenzung der Seitengröße (TTL-Faktor: 10%)

Cachegröße	Seiten-obergrenze	mittl.Über-tragungszeit	Cache-Hit-Rate	Byte-Hit-Rate	Stale-Rate	304-Reply-Rate	von Servern übertr. Bytes
50000000	5000000	25.5018	0.239607	0.140317	0.0154443	0.000668551	5286892991
200000000	5000000	22.5249	0.321965	0.182755	0.0279551	0.00191383	5025981374
500000000	5000000	20.4403	0.36838	0.21787	0.0326237	0.0111318	4810608317
850000000	5000000	18.8766	0.409508	0.258745	0.0326575	0.0518297	4561760198
50000000	1000000	25.5334	0.244654	0.13442	0.0168926	0.000451071	5323142782
200000000	1000000	22.4037	0.329212	0.178659	0.0290586	0.00223119	5051195751
500000000	1000000	20.4183	0.37393	0.212123	0.0325786	0.0149288	4846184161
850000000	1000000	18.3648	0.426811	0.253019	0.0319165	0.0696421	4598081780
50000000	500000	25.4822	0.248016	0.13123	0.0171359	0.000407575	5342757902
200000000	500000	22.3213	0.333367	0.174147	0.0301299	0.0025711	5078963968
500000000	500000	20.2973	0.37913	0.205825	0.0332165	0.0193542	4885192713
850000000	500000	18.1374	0.436268	0.245959	0.0322822	0.078314	4642034422
50000000	100000	25.7056	0.258551	0.120909	0.0186212	0.00051551	5406236956
200000000	100000	22.5936	0.345585	0.162183	0.0316104	0.00539996	5152714742
500000000	100000	20.0415	0.41624	0.205073	0.0324852	0.060073	4892343540
850000000	100000	18.9933	0.453571	0.227179	0.0327784	0.0961619	4758634421

Bewertungsmaße: LCP mit Replikation (TTL-Faktor: 10%)

Cachegröße	Replikationsrate	mittl.Übertragungszeit	Cache-Hit-Rate	Byte-Hit-Rate	Stale-Rate	304-Reply-Rate	von Servern übertr. Bytes
50000000	0.05	24.9611	0.217574	0.13214	0.0130182	0.00068305	5337176946
200000000	0.05	22.2099	0.285908	0.161261	0.0234057	0.00160936	5158149290
500000000	0.05	20.2627	0.338281	0.196886	0.024817	0.00582043	4939321934
850000000	0.05	19.889	0.350996	0.210456	0.0291247	0.0117101	4856239799
50000000	0.1	25.0983	0.207274	0.126519	0.0140815	0.000702382	5371748204
200000000	0.1	22.1166	0.298321	0.169239	0.02063	0.00150948	5109081424
500000000	0.1	20.355	0.333022	0.191481	0.0282048	0.00554978	4972544454
850000000	0.1	19.6518	0.355753	0.213951	0.027438	0.00997833	4834634907
50000000	0.25	25.3632	0.201569	0.118439	0.0115426	0.000695938	5421439415
200000000	0.25	22.3855	0.285418	0.163096	0.0210038	0.00143859	5146852013
500000000	0.25	20.4167	0.336726	0.195946	0.0241371	0.00446077	4945020476
850000000	0.25	19.313	0.367576	0.219486	0.0250022	0.00784219	4800464064
50000000	0.4	25.8656	0.197888	0.116949	0.0105986	0.000608946	5430594487
200000000	0.4	22.4858	0.283384	0.167808	0.0183763	0.00123722	5117860637
500000000	0.4	20.4739	0.340246	0.197188	0.0212116	0.00287074	4937286022
850000000	0.4	19.0969	0.372444	0.22344	0.0209442	0.00495534	4775972559

Bewertungsmaße: LCP mit Begrenzung der Seitengröße (TTL-Faktor: 10%)

Cachegröße	Seiten-obergrenze	mittl.Über-tragungszeit	Cache-Hit-Rate	Byte-Hit-Rate	Stale-Rate	304-Reply-Rate	von Servern übertr. Bytes
50000000	5000000	24.8279	0.214072	0.12546	0.0149256	0.000571893	5378253235
200000000	5000000	22.1478	0.286879	0.16171	0.025384	0.0016303	5155389521
500000000	5000000	20.5258	0.328836	0.185519	0.0303168	0.00648414	5009268478
850000000	5000000	19.7866	0.353221	0.206188	0.0311384	0.0217062	4883107586
50000000	1000000	24.8784	0.217916	0.122562	0.0156038	0.000579948	5396075515
200000000	1000000	22.1029	0.292036	0.158687	0.0259446	0.0016013	5173978357
500000000	1000000	20.5674	0.331276	0.182427	0.0304811	0.00804678	5028384674
850000000	1000000	19.3238	0.367526	0.204091	0.0309274	0.0364127	4896914281
50000000	500000	24.8226	0.220164	0.119656	0.0164673	0.000373744	5413933359
200000000	500000	22.0809	0.293518	0.154502	0.0264811	0.00183328	5199727754
500000000	500000	20.5637	0.335341	0.176782	0.0306245	0.00939838	5063179502
850000000	500000	19.1937	0.3758	0.199794	0.0309	0.0450185	4923870764
50000000	100000	25.1803	0.221919	0.107966	0.0169844	0.000346358	5485823507
200000000	100000	22.5572	0.304634	0.143674	0.0281629	0.00330248	5266406450
500000000	100000	21.1737	0.352515	0.16557	0.0310047	0.0229241	5132969188
850000000	100000	20.1827	0.39153	0.185888	0.0308613	0.0613456	5010404324