Küfr-98

Darmstadt University of Technology

Möglichkeiten für den Einsatz von Lastverteilungsstrategien verteilter Systeme in der Videoverteilung

Christine Küfner, Carsten Griwodz

11 June 1999

Technical Report TR-KOM-1998-07

Industrial Process and System Communications (KOM)

Department of Electrical Engineering & Information Technology
Merckstraße 25 • D-64283 Darmstadt • Germany

Phone: +49 6151 166150
Fax: +49 6151 166152
Email: info@KOM.tu-darmstadt.de
URL: http://www.kom.e-technik.tu-darmstadt.de/

Möglichkeiten für den Einsatz von Lastverteilungsstrategien verteilter Systeme in der Videoverteilung

Christine Küfner, Carsten Griwodz[1]
1
Industrial Process and System Communications
Dept. of Electrical Engineering & Information Technology
Darmstadt University of Technology
Merckstr. 25 • D-64283 Darmstadt • Germany

{griff}@kom.tu-darmstadt.de

Übersicht

Videoverteilungssysteme bieten gegenüber konventionellen Videotheken viele Vorteile. Vor allem stellt ein solches System jedem Nutzer die Möglichkeit bereit, eine Film zu jeder Zeit und ohne das Haus zu verlassen zu bestellen und anzusehen. Um diesen Vorteil zu wahren, muß eine VoD-System eine Anfrage in einem vernünftigen Zeitraum erfüllen können. Der Faktor Zeit wird vor allem durch die Architektur eines Video-on-Demand-Systems beeinflußt. Da sich solche Systeme zur Zeit noch im Versuchsstadium befinden, gibt es noch kein allgemeingültiges Modell. Auf jeden Fall müssen zwischen einer Zentralbibliothek und den Kunden mehrere Server geschaltet sein, damit ein Film nicht direkt zum Kunden übertragen werden muß, sondern innerhalb dieser Server zwischengespeichert werden kann. Ein solcher Server kann in seinem Cache nicht nur einen, sondern gleich mehrere Filme speichern, die er dann nicht nur einem Kunden, sondern allen in der Nähe liegenden Nutzern sehr schnell zur Verfügung stellen kann. Leider ist die Kapazität eines Caches begrenzt, d.h. es muß genau überlegt werden, welche Filme man in einem Cache speichert und welche man für neue Videos herausnehmen soll. Diese Entscheidung wird mittels sogenannter Ersetzungsalgorithmen getroffen, von denen heutzutage eine Vielzahl in der Literatur und der Praxis zu finden sind. Der Hauptzweck dieser Studienarbeit besteht nun in der Dokumentation vorhandener Caching-Strategien und ihre Bewertung in Hinblick auf ihre Einsetzbarkeit in Videoverteilungssystemen.

Zunächst wird dazu in Kapitel 2 erklärt was Caching eigentlich ist und warum man Ersetzungsalgorithmen verwendet. Ein mögliches Videoverteilungsszstemmodell, Übertragungsmöglichkeiten von Videos und die Erläuterung der Haupteinflußfaktoren einer „optimalen" Caching-Strategie für VoD-Systeme werden im Kapitel 3 vorgestellt. Anschließend werden Caching-Algorithmen aus drei verschiedenen Systemen erläutert und in Hinblick auf ihre Eignung in Video-on-Demand-Systemen bewertet. Dabei behandelt das Kapitel 4 Paging, das Kapitel 5 Caching zwischen Hauptspeicher und Platte und das Kapitel 6 Caching im World Wide Web. Die Möglichkeit einer Kooperation der Server innerhalb eines Videoverteilungssystems, sowie vorhandene Cooperative Caching Strategien werden im Kapitel 7 aufgezeigt. Alle Algorithmen die in dieser Arbeit beschrieben wurden, werden dann im Kapitel 8 mittels einer Tabelle verglichen. Ein Ausblick auf mögliche anschließende Arbeiten soll im Kapitel 9 vorgenommen werden.

Keywords: Caching, Web Caching, Paging, Cooperative Caching.

1 Caching

1.1 Cache Speicher

Unter einem Cache Speicher versteht man einen High-Speed-Speicher, der zum Zwischenspeichern von Daten dient, die der Prozessor häufig benötigt. Der Prozessor kann auf im Cache abgelegte Daten wesentlich rascher zugreifen als auf Daten im Hauptspeicher. Werden die angeforderten Daten im Cache gefunden, wird der Speicherzugriff als Cache-Hit (Cache-Treffer) bezeichnet. Sind sie nicht vorhanden, liegt ein Cache-Miss (Cache-Fehlertreffer) vor.

Man unterscheidet zwei Cache-Level:

Den als Primären oder Level 1-Cache bezeichneten internen Cache, der sich in der Regel im Prozessor befindet und den als Sekundären oder Level 2-Cache bezeichneten externen Speicher der sich zwischen CPU und dem Hauptspeicher befindet und aus sehr schneller Bausteinen sogenannten SRAMs besteht.

Wenn man also von einem Cache spricht meint man im allgemeinem einen Level 2-Cache, der mit relativ geringen Kosten die Leistung eines Rechners steigern kann. Mit Hilfe eines Caches kann der DRAM-Hauptspeicher nahezu mit SRAM-Geschwindigkeit arbeiten. Die damit verbundenen Kosten sind jedoch wesentlich geringer als die Kosten für einen Hauptspeicher, der vollständig mit SRAMs aufgebaut ist.

Die Leistung eines Cache-Systems läßt sich z.B. mit der Trefferquote darstellen. Die Trefferquote ist der Prozentsatz an Zugriffen, die ein Treffer (Hit) sind. Sie wird von der Größe und Organisation des Caches, dem Ersetzungsalgorithmus und dem gerade laufenden Programm bestimmt. Ein effektives Cache-System sollten Dateien so strukturiert halten, daß die Trefferquote möglichst hoch ist.

1.2 Ersetzungsstrategien

Das Hauptproblem beim Cache besteht ist seinem begrenzten Speicherplatz. Da bei jeder Anfrage nach einer Seite, einem Block, einer Datei usw. Kopien im Cache abgelegt werden, nimmt der freie Speicherplatz im Laufe der Zeit immer mehr ab. Es muß daher in gewissen Zeitabständen immer wieder freier Platz im Cache geschaffen werden, um zu gewährleisten, daß auch weiterhin Kopien von erhalten Seiten gespeichert werden können. Die Schwierigkeit besteht nun in der Auswahl der Seiten, die aus dem Cache entfernt werden sollen.

Zur Lösung dieses Auswahlproblems gibt es nun unterschiedliche Ansätze. Bei all diesen Ansätzen wird hierbei für jede der im Cache befindlichen Seite ein bestimmter Wert ermittelt. Anschließend werden dann diejenigen Seiten aus dem Cache gelöscht, die - je nach Strategie - die kleinsten bzw. größten Werte aufweisen. Die Art und Weise der Ermittlung dieser Werte hängt wiederum von der Auswahl und der Gewichtung bestimmter zu berücksichtigender Parameter ab. In der Regel ist man darum bemüht, diejenigen Parameter zu finden, mit deren Hilfe sich Aussagen über die Wahrscheinlichkeit einer erneuten Anfrage nach einer Seite ableiten lassen, so daß die Speicherung einer Seite nur noch bei einer entsprechend hohen Anfragewahrscheinlichkeit erfolgt. Die Ansätze bezeichnet man allgemein als Ersetzungs- oder Löschalgorithmen, im Englischen als Replacement Algorithms. [[DOM98]]

1.3 Eingrenzung der realisierbaren Ersetzungsstrategien

Allgemein lassen sich zwei Typen von Strategien unterscheiden: Eine Strategie heißt realisierbar, wenn die Ersetzungsentscheidung zum Zeitpunkt t keine Kenntnisse über das zukünftige Referenzverhalten des Prozesses oder den Nutzers erfordert. Eine Strategie heißt nicht-realisierbar, wenn sie Kenntnisse über zukünftige Referenzen benötigt. Nicht-realisierbare Strategien sind nur von theoretischem Interesse. Ihr Wert liegt darin, daß sie eine obere Schranke für die mit realisierbare Strategien erreichbaren Trefferquoten festlegen. [[EFF81]]

Theoretische Überlegungen führen zu einem Optimalitätsprinzip, welches eben zu diesen nicht-realisierbaren Strategien zählt.

1.3.1 OPT (Optimal Algorithm)

Beschreibung:

Die optimale Ersetzungsstrategie wählt den zu ersetzenden Block dahingegen aus, daß sie untersucht welcher Block erst in der am weitesten entfernten Zukunft benötigt wird. Dieser wird dann durch einen neuen Block ersetzt. Diese Strategie ist allerdings nur dann optimal, wenn alle Seiten gleich groß sind.

Sobald des Größenunterschiede gibt, müssen noch anderer Faktoren berücksichtigt werden, um eine optimale Strategie definieren zu können. Ein weiteres und entscheidendes Problem liegt aber in der praktischen Durchführbarkeit dieser Strategie. Um sie zu verwirklichen müßte man in die Zukunft sehen können, was in der Regel leider nicht möglich ist. [[RL96]][[MGS+70]]

Parameter:

- Zeit, bis ein Block wieder aufgerufen wird

Bewertung:

Dieser Algorithmus gehört, wie oben schon beschrieben, zu den nicht-realisierbaren Strategien. Es werden schon hellseherische Fähigkeiten benötigt, um ihn anwenden zu können. Dies ist jedoch nicht das einzige Problem. Der Algorithmus ist nur dann optimal, wenn alle Seiten die gleiche Größe haben, was bei Filmen, die vollständig übertragen[1] werden natürlich nicht der Fall ist. Weiterhin ignoriert er vollständig die unterschiedlichen Speicherkosten[2], die jeder Film hat.

So einfach, wie sich eine obere Schranke definieren läßt, so leicht kommt man auch zu einer unteren Schranke. Diese wird durch das Random Verfahren entwickelt.

1.3.2 RAND (Random Algorithm)

Beschreibung:

Man geht hier davon aus, daß alle Blöcke im Puffer den selben Erwartungswert für eine Anfrage des Prozesses oder des Nutzers haben. Von den r Blöcken die sich zu einer bestimmten Zeit t im Speicher befinden, wird deshalb der zu ersetzende Block nach dem Zufallsprinzip ausgewählt. Dabei wird eine Nummer m zwischen 1 und r generiert. Der der Nummer m zugeordnete Block wird dann durch den neuen Block ersetzt.

Dies ist der am einfachsten ausführbare Algorithmus und liefert hier die untere Leistungsgrenze.[[CO76]][[RL96]]

Der RAND Algorithmus tritt auch unter folgenden Bezeichnungen auf: **ARB** (Arbitrary Replacement Algorithm), **RR** (Random Replacement Algorithm) und **RANDOM** (Random Replacement Algorithm)

Parameter:

- r = Anzahl der Seiten
- m = generierte Zufallsnummer

Bewertung:

Der RAND Algorithmus berücksichtigt überhaupt keine Faktoren bei der Auswahl des zu ersetzenden Filmes. Er stellt zwar damit die einfachste aber gleichzeitig auch die schlechteste Strategie dar.

Das Ziel ist nun so nahe wie möglich an der optimalen Strategie heranzukommen. Das beliebteste und auch einer der einfachsten Verfahren ist der LRU.

---

1. zu den Übertragungsmöglichkeiten von Filmen siehe auch Kapitel 3.3
2. unterschiedliche Lade- und Wartezeiten, die zu unterschiedlichen Kosten führen

1.3.3 LRU (Least Recently Used Algorithm)

Beschreibung:

Unter dieser Methode versteht man, daß derjenige Block zuerst gelöscht wird, dessen Verwendung am weitesten zurück liegt. Dabei geht man von der allgemeinen Beobachtung aus, daß Blöcke die lange nicht mehr aufgerufen wurden auch in nächster Zeit nicht mehr aufgerufen werden.

Dieser Algorithmus ist sehr einfach einzusetzen und liefert im allgemeinem auch gute Ergebnisse. Aufgrund dessen hat sich diese Strategie zu der am häufigsten verwendeten Methode entwickelt. Jeder neue Algorithmus hat sich in seiner Effizienz somit am LRU zu messen. [[TAN90]]

Parameter:

- Vergangene Zeit seit der letzten Anfrage

Bewertung:

In Videoverteilungssystemen ist der Zeitpunkt der letzten Anfrage kein sehr aussagekräftiger Einflußfaktor. Um die Popularität eines Filmes zu bestimmen ist es viel entscheidender, wie oft ein Video angefordert wurde. Allerdings stellt der Einsatz der LRU-Strategie als Sekundäralgorithmus eine gute Wahl dar.

Die nun folgende Abbildung 1 gilt nur unter der Annahme gleich großer Blöcke. Vor allem die obere Grenze, die durch den OPT Algorithmus gebildet wird, ist nur unter dieser Annahme als richtig einzuordnen.

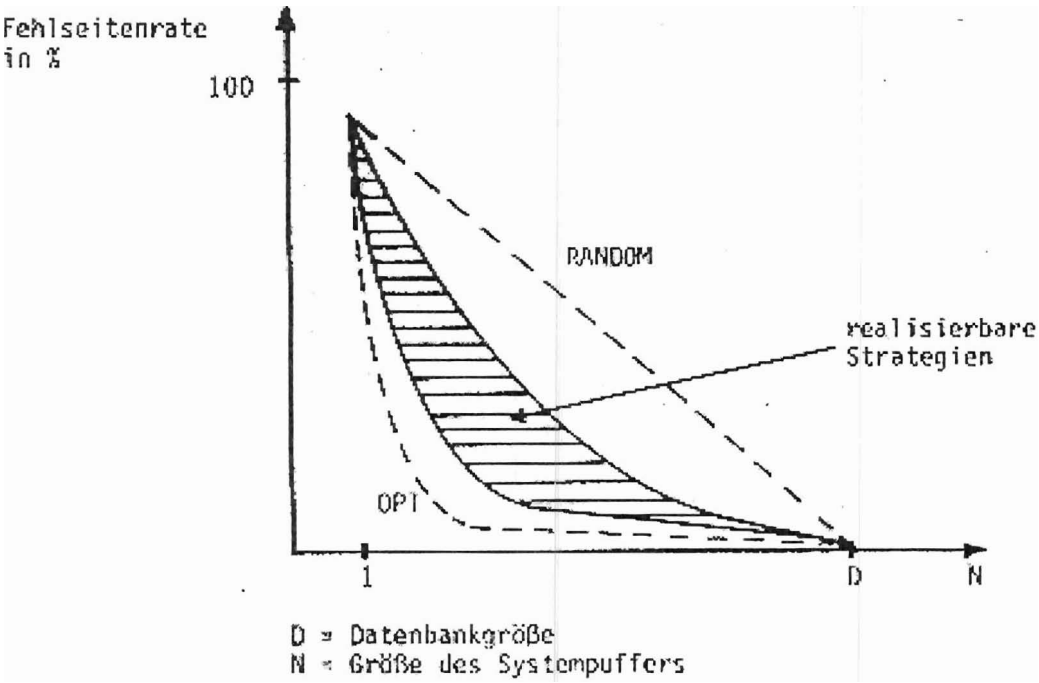


Abb. 1: Eingrenzung der realisierbaren Strategien durch RANDOM und OPT [[EFF81]]

2 Videoverteilungssysteme

2.1 Arten von Videoinformationssystemen

Es lassen sich verschiedene Arten von Videoinformationssystemen unterscheiden, wobei aufgrund der bitweisen Speichermöglichkeit nur digitale Videoinformationssysteme interessant sind:

1. *Pay-per-Channel: Es gibt einen Kanal und keine Auswahlmöglichkeit der Filme. Dies entspricht der heutigen Form von Fernsehkanälen.*
2. *Pay-per-View: Entspricht in der Funktionsweise Pay-per-Channel, allerdings werden nur die tatsächlich gesehenen Filme erfaßt und abgerechnet.*
3. *Near-Video-on-Demand: Hier wird der gleiche Film über mehrere Kanäle zeitversetzt angeboten. Indem der Zuschauer die Kanäle wechselt, kann er im Film vor- und zurückspringen bzw. den Startzeitpunkt des Filmes innerhalb gewisser Grenzen selber bestimmen.*
4. *Video-on-Demand: Alle Filme können zu jedem Zeitpunkt abgerufen werden. Der Kunde kann sein „Programm" ganz individuell zusammenstellen.*
5. *Interactive-Video-on-Demand: Die Funktionsweise entspricht dem Video-on-Demand-Prinzip, aber der Kunde verfügt nun über Möglichkeiten des Eingriffes. Wie bei einem Videogerät läßt sich der Film vor- und zurückspulen, anhalten und wieder starten.*

[[GBW97]]

Die in dieser Arbeit vorgenommenen Bewertungen der einzelnen Algorithmen basieren immer auf *Video-on-Demand-Systemen*. Es besteht also keine Möglichkeit der Interaktivität.

2.2 Modelle von Videoverteilungssystemen

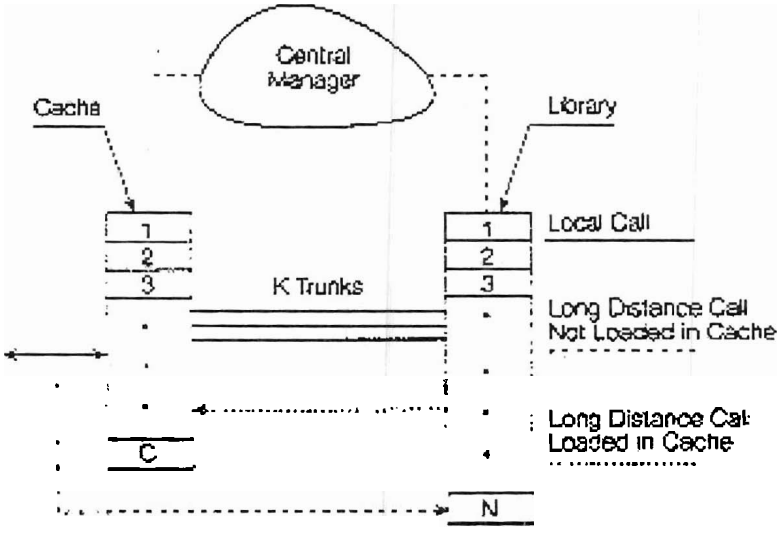


Abb. 2: Ein einfaches VoD-Service Modell [[BM96]]

Die Abbildung 2 stellt ein einfaches Modell eines Videoverteilungssystems dar. Es besteht aus einem einzelnen Cachespeicher verbunden mit einer Video-Bibliothek. Jeder Speicher enthält eine bestimmte Anzahl von Speicherplätzen, die jeweils ein Programm enthalten. Die Größe des Caches C ist dabei um einiges kleiner als die Größe der Bibliothek N, wodurch die verfügbare Anzahl von Programmen für

den Benutzer begrenzt wird. Der Cache und die Bibliothek kommunizieren über K Fernverbindungskabeln[3] (evtl. K = unendlich), wobei jedes Kabel eine Verbindung ermöglicht.

Wählt nun der User einen Videofilm aus, prüft der Servicekontroller zunächst ob das betreffende Programm im Cache enthalten ist. Ist das der Fall, so wird eine Verbindung zwischen Nutzer und dem Cache aufgebaut. Ansonsten muß eine Verbindung zwischen dem Nachfrager und der Bibliothek hergestellt werden, was natürlich viel länger dauert. Der Videofilm der direkt von der Zentrale abgerufen wird kann nun in den Cache gespeichert werden, wenn er z.B. als „populär" eingestuft wird. In einem solchen Fall muß dem Programm Speicherplatz im Cache eingeräumt werden, evtl. auch auf Kosten früherer gespeicherter Filme die zur Zeit nicht frequentiert werden. Zur Auswahl des zu ersetzenden Programmes verwendet man einen Ersetzungsalgorithmus.

Allgemein lassen sich zwei Grundstrategien unterscheiden:

1. *Always Overwrite: AO*

   Jeder Film der von der Zentrale heruntergeladen wird, wird auch im Cache gespeichert. Diese Technik hat den großen Nachteil, daß nun jedes Video unabhängig seiner Popularität im Cache gespeichert wird. Daher werden unpopulärere Filme auf Kosten von populäreren ständig geladen und wieder überschrieben. Dies führt wie man anhand der Abbildung 3 erkennen kann zu schlechten Leistungswerten.

2. *2. Conditional Overwrite: CO*

   Diese Strategien berücksichtigen die Referenzwahrscheinlichkeit des angeforderten Videofilmes, sowie der im Cache enthaltenen Programme. Im Fall, daß das zu ladene Programm populärer ist, als der unpopulärste Filmes im Cache, wird das neue Programm in den Cache geladen. Im anderen Fall übermittelt man das angeforderte Video ausschließlich zum User. [[BM97]][[BM96]]

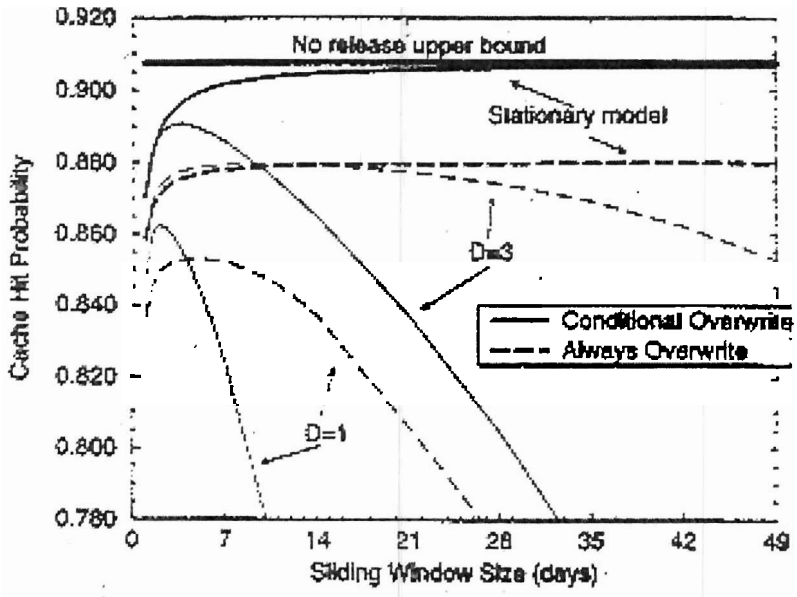


Abb. 3: Cache-Hit Wahrscheinlichkeit gegenüber variierender Fenstergröße (window-size) [[BM97]]

In dieser Abbildung wird zwischen stationärer und nicht-stationärer Anfrageverteilung unterschieden. Unter stationärem Verhalten versteht man eine konstante Wahrscheinlichkeit einer erneuten Anfrage,

3. trunks

für jeden Film der im Cache gespeichert ist. Dies ist natürlich illusorisch, da das Anfragemuster der Kunden natürlich ständig variiert. In der Realität muß somit mit einer nicht-stationären Anfrageverteilung gerechnet werden. Dabei soll der Parameter D die durchschnittliche Zugangszeit eines neuen Filmes (gemessen in Tagen) darstellen.[[BM97]]

Leider wird bei den in dieser Arbeit aufgeführten Algorithmen meistens nach dem Prinzip des Always Overwrite vorgegangen. Nur in wenigen Ausnahmen werden die neu angefragten Filme mit dem schon im Cache befindlichen Videos verglichen (Conditional Overwrite)[4].

Natürlich besteht ein reales Videoverteilungssystem nicht nur aus einem Cache und einer Bibliothek. Vielmehr muß man sich ein hierarchisch verteiltes System mit einer Baumstruktur vorstellen. Bleibt man bei diesem Bild eines Baumes, so liegt an dessen Spitze (oder Krone) der Hauptserver bzw. die Bibliothek. An den Ästen verteilt befinden sich die verschiedenen Servern und die Blätter stellen die Endsysteme, also die Nutzer in Form von Computern oder Fernsehgeräten dar.

Ein mögliches Modell wird in der Abbildung 4 dargestellt. Es handelt sich hierbei um die eben beschriebene Baumstruktur, die verschiedene User mit dem Zentralserver über Zwischenknoten verbindet. Haben diese Zwischenknoten die Fähigkeit Daten zu speichern oder zu cachen, so spricht man von einer Server Hierarchie.

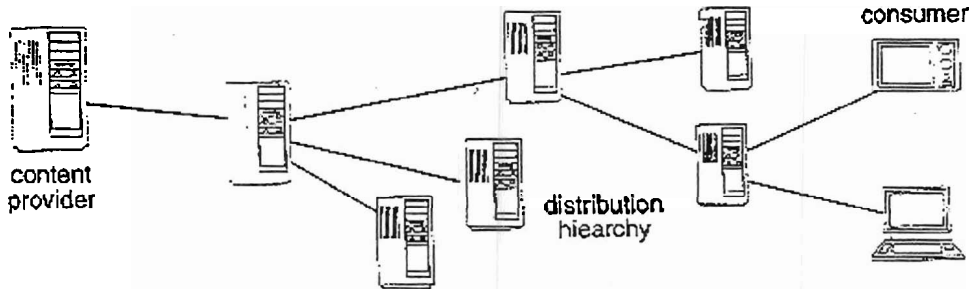


Abb. 4: Hierarchisches Video-on-Demand System [[GBW97]]

Videofilme können nun über diese Baumstruktur an den User geschickt werden. Dabei muß nicht immer der Zentralserver bei einer Anfrage mit eingeschaltet werden, da sich viele Filme auch in der näheren Umgebung des Nutzers, nämlich in den kleineren Servern befinden. Caching-Strategien die auch mit den benachbarten Servern kommunizieren sind hier also von Vorteil[5]. [[GBW97]]

2.3 Übertragungsmöglichkeiten von Videos

Grundsätzlich lassen sich Videofilme auf zwei Arten übertragen:

1. *Ein Film wird in Blöcken zerlegt übertragen.*
2. *Ein Film wird vollständig übertragen.*

Im folgenden werden nun beide Fälle betrachtet:

Wird der Film in Blöcken zerlegt, so sind diese alle gleich groß[6]. Man hat es hier also mit kleinen, konstanten Einheiten zu tun. Von jedem Video können nun mehrere Blöcke in einem Cache gespeichert werden. Dies ist vor allem dann vom Vorteil, wenn die Kapazität des vorhandene Caches sehr klein ist. Denn durch die Zerlegung können nun mehr Filme für den Kunden zur Verfügung gestellt werden.

Bei der Anwendung von Replacement Algorithmen ist zu beachten, daß man bei der Auswahl des zu ersetzenden Blockes nicht wahllos vorgehen kann. Statt dessen muß hier eine bestimmte Reihenfolge

4. siehe dazu Tabelle in Kapitel 8.1
5. siehe dazu Cooperative Caching in Kapitel 7
6. Es besteht auch die Möglichkeit den Film in gleich große Zeiteinheiten zu übertragen.

eingehalten werden, da die einzelnen Blöcke in einem fest vorgegebenen Zyklus ablaufen müssen. Dadurch wird die Anwendung von Algorithmen stark eingeschränkt. Interessant ist hier eher die Speicherplatzzuweisung[7] für die einzelnen Filme, d.h. es muß festgelegt werden, wieviele seiner Blöcke ein Film in den Cache laden darf.

Diese Zerlegung von Filmen hat jedoch einen entscheidenden Nachteil. Durch die Begrenzung der speicherbaren Blöcke wird nun nie der vollständige Film im Puffer vorhanden sein. Da aber beim Ablauf eines Filmes jeder Block nur einmal benötigt wird, wird er bei Gebrauch sofort entfernt, um Platz für einen neuen Block zu schaffen. Dies führt zu einem sehr hohen Datenverkehr, hohen Kosten, Problemen in der rechtzeitigen Speicherung der nächsten Blöcke (Gefahr der Unterbrechung der Filmübertragung) usw. Die Speicherung von einzelnen Blöcken eines Filmes scheint somit kaum sinnvoll und unwirtschaftlich.

Trotzdem sind mögliche Einsatzbereiche vorstellbar. Ist die Entfernung zwischen Server und Nutzer recht kurz und der Cache des Nutzer sehr klein (was in der Regel der Fall sein wird), so könnte man immer nur eine kleine Anzahl von Blöcken in den Cache des Users laden. Dadurch müssen die Übertragungsleitungen nicht ständig offen sein und der User spart Leitungsbenutzungsgebühren. Es muß natürlich sichergestellt werden, daß die einzelnen Blöcke immer rechtzeitig zur Verfügung stehen (kurze und sichere Verbindungen). Die Frage welche Blöcke entfernt werden müssen stellt sich hier allerdings nicht. Es wird auf jeden Fall nach der FIFO-Strategie[8] vorgegangen. Auch hat man es hier immer nur mit einem einzelnen Film zu tun, so daß man keine Algorithmen zur Speicherplatzzuweisung benötigt.

Dasselbe ließe sich noch für einen sehr kleine Server denken, der direkt mehrer User beliefert. Die Leitungen zwischen den Nutzern und dem Server müssen dabei wiederum recht kurz und sicher sein, um Übertragungsprobleme zu vermeiden. Hier sind vor allem Algorithmen zur Speicherplatzzuweisung gefragt, um dem jeweiligen Film die notwendige Anzahl an Speicherrahmen zur Verfügung zu stellen.

Werden die Videos als vollständige Einheiten übertragen, so benötigt jeder einzelne Film sehr viel Speicherplatz im Cache. Aufgrund dessen können nur wenige Filme auf einmal im Cache gespeichert werden. Trotzdem scheint dies die sinnvollere Methode zu sein, vor allem bei Servern, die sich in der Hierarchie weiter oben befinden. Außerdem kann man davon ausgehen, daß die Kapazität von Caches in den nächsten Jahren noch ansteigen wird.

2.4 Der „optimale" Algorithmus für Videoverteilungs-systeme

Die Frage ist nun, welche Faktoren in einem „optimalen" Algorithmus für Videoverteilungssysteme berücksichtigt werden sollten. Grundsätzlich kann man schon sagen, daß der optimale Algorithmus wohl nicht realisierbar ist. Würde man alle notwendigen Parameter bis aufs kleinste berücksichtigen, so entstände ein viel zu komplexer und aufwendiger Algorithmus. Bei der Entwicklung einer Strategie sollte auf jeden Fall das Kosten/Nutzenverhältnis nicht aus den Augen gelassen werden. Des weiteren ist die Situation in Video-on-Demand-Systemen nicht konstant, sondern unterliegt vielen Änderungen. Je nach Größe eines Caches, Tageszeit, Kundenklientel usw. wird sich eine andere Strategie als die bessere erweisen. Günstig wäre natürlich ein Algorithmus der solche Änderungen erkennt und entsprechend reagieren kann, indem er sich anpaßt.

Grundsätzlich kann man zunächst einmal in Primär- und Sekundärstrategie unterscheiden. Wie der Name schon sagt, wird mit der Primärstrategie der primäre Wert eines Filmes ermittelt. In den wohl seltenen Fällen, daß mehrere Filme den gleichen Wert aufweisen, muß auf die Sekundärstrategie zurückgegriffen werden, um eine Entscheidung fällen zu können. Es lassen sich natürlich auch noch weitere untergeordnete Strategien bilden, deren Wahrscheinlichkeit zum Einsatz jedoch sehr gering ist. Denn je komplexer die Primärstrategie gestaltet ist, desto unwahrscheinlicher ist eine Wertgleichheit mehrere Videos.

---

7. siehe Kapitel 4.5: Konventionelle lokale Seitenersetzungsalgorithmen mit variablen Partitionen
8. zur FIFO-Strategie siehe Seite 33

Entscheidend für die Entscheidung, ob ein Film im Cache gehalten werden soll ist seine Popularität. Doch wie läßt sich diese Popularität messen? Das einfachste Mittel stellt wohl die Ermittlung der *gesamten Anfragen an einen Film* dar. Dabei besteht aber das Problem, daß Referenzen die vor mehreren Wochen stattgefunden haben genauso berücksichtigt werden, wie die Anfragen der letzten Woche. Dies verfälscht jedoch die aktuelle Popularität eines Filmes. Um dieser Schwierigkeit aus dem Weg zu gehen, kann man entweder die Anfragen unterschiedlich gewichten (*Alterung*) oder nur die Referenzen berücksichtigen, die in einem bestimmten *Intervall* angefallen sind. Durch die Einführung eines Intervalls läßt sich das aktuelle Anfragemuster besser ermitteln. Zu überlegen ist jedoch welche Größe ein Intervall haben sollte. Denkbar wäre hier eine 7-Tage Periode.

Doch nicht nur die Popularität eines Filmes muß berücksichtigt werden. Jeder Film *verursacht Kosten*, die auch einen entscheidenden Einfluß auf die Ersetzungsentscheidung haben. Ein Film der zwar weniger populär ist als ein anderer, dessen Wiederbeschaffung aber viel höhere Kosten verursacht wird man lieber im Cache halten und statt dessen den populäreren Film vorübergehend entfernen. Was gehört aber nun zu den Kosten, die ein Video verursacht? Hier spielen viele Faktoren eine Rolle, wie z.B. Gebühren für den Film (Kirch-Gruppe), Transportkosten über gemietete Leitungen (*Ladezeit* und *Bandbreitenbedarf*), benutzten Speicherraum im Puffer (*Größe des Videos*) usw. Aufgrund der nur „geringfügigen" Unterschiede in den Größen der Videos[9] spielen bei den Kosten die Ladezeit und der Bandbreitenbedarf die größte Rolle.

Die Größe eines Filmes läßt sich jedoch sehr gut als Sekundärstrategie verwenden. Gibt es zwei Filme mit dem gleichen Primärwert, so wird man den größeren Film der beiden ersetzen. Ebenso läßt sich die *vergangene Zeit seit der letzten Anfrage* als Entscheidungskriterium für eine Sekundärstrategie einsetzen.

All diese Faktoren sollten aber nicht nur für schon im Cache gespeicherten Videos berücksichtigt werden, sondern auch für einen *neu nachgefragten Film*. In einigen Fällen wird es besser sein, den angeforderten Film direkt an den Kunden zu übermitteln, *ohne ihn in einem Cache zu speichern*. Dies sollte vor allem dann geschehen, wenn es sich bei dem angeforderten Video um eine wenig populären Film handelt.

Nun lassen sich für Algorithmen in Videoverteilungssystemen auch einige Besonderheiten denken. Man könnte z.B. die Filme mit unterschiedlichen *Prioritäten* belegen, um einige Filme vor einer Ersetzung zu schützen (Top-Ten der Woche) und andere dafür geradezu prädisponieren. Eine weitere Überlegung geht dahin, daß der Inhalt des Caches zu bestimmten Zeiten neu zusammengestellt wird, d.h es findet eine periodische *Speicherreinigung* statt. Denkbar wäre ein Rhythmus von zwölf Stunden. Dies würde dem unterschiedlichem Nutzerverhalten bei verschiedenen Tageszeiten Rechnung tragen. Kinderfilme werden kaum um ein Uhr morgens benötigt, dagegen sollten Horrorfilme (Filme ab 18 werden erst nach zehn Uhr zur Sendung freigegeben) nur in den Nachstunden angeboten werden. Die Schwierigkeit liegt hier allerdings in der Übertragung des jeweiligen neuen Speicherinhaltes. Dies dauert recht lange und blockiert die Leitungskapazitäten. Das Problem der Überlastung von Letitungen sollte besonders beachtet werden, da dieser Sachverhalt in der Literatur leider of ignoriert wird. Der Übergang muß möglichst gleitend vonstatten gehen und zu Zeiten geringer Anfragen vorgenommen werden.

Als letzten Punkt soll hier noch die Möglichkeit der *Kooperation von Servern* aufgeführt werden. Es ist natürlich von Vorteil, wenn man auf den Inhalt der anderen Servern zugreifen kann und sich nicht bei jeder unerfüllbaren Anfrage an die Bibliothek wenden muß. Damit eine solche Koordination funktioniert, benötigt man sogenannte Cooperative Caching Algorithmen. Diese ersetzen aber nicht die Replacement Algorithmen, sondern dienen zur Verwirklichung des Cooperative Caching. Intern kann jeder Cache seine persönliche Strategie weiterhin frei wählen.

9. Es wird hier von Spielfilmen ausgegangen. Natürlich sind auch kleinere Dokumente, wie Serienfolgen, Berichte, Nachrichtensendungen usw. als Video-on-Demand-Objekte denkbar. Ist dies der Fall, so ist die Dokumentengröße ein relevanter Faktor.

In Kapitel 8 werden die oben aufgeführten, durch Kursivschrift hervorgehobenen Einflußfaktoren genau erklärt und in einer Tabelle für die einzelnen Algorithmen zusammengestellt. Außerdem werden diese Parameter in den Bewertungen der folgenden Algorithmen wieder aufgegriffen. Zu bemerken ist noch, daß die Notation in dieser Arbeit vereinheitlicht wurde, um die Vergleichbarkeit der verschiedenen Algorithmen zu verbessern.

3 Paging

3.1 Virtueller Speicher

In den 50er stand man vor dem Problem, daß viele Programme zu groß für den zur Verfügung stehenden Speicher waren. 1961 fand Fotheringham eine Lösung, die heute unter dem Namen virtueller Speicher bekannt ist. Die Grundidee ist dabei folgende: Da die gesamte Größe des Programms und der Daten den hierfür verfügbaren physikalischen Speicher sprengt, wird das Betriebssystem nur jene Teile des Programms im Speicher halten, die aktuell verwendet werden und den Rest auf der Platte lassen. Beispielsweise kann ein 1M Programm auf einer 256k Maschine laufen, wenn zu jedem Zeitpunkt jene 256k des Programms ausgewählt werden, wobei benötigte Programmteile zwischen der Platte und dem Speicher ein- und ausgelagert werden.

Die meisten virtuellen Speichersysteme verwenden dabei eine Technik, die Seitenwechsel[1] genannt wird. Jeder Computer verfügt über Speicheradressen, die von Programmen erzeugt werden können. Diese programmgenerierten Adressen heißen virtuelle Adressen und bilden den virtuellen Adreßraum. Auf einem Computer ohne virtuellen Speicher wird die virtuelle Adresse direkt auf den Speicherbus gegeben und veranlaßt, daß das Speicherwort mit derselben Adresse gelesen wird. Wird jedoch ein virtueller Speicher verwendet, werden die virtuellen Adressen nicht direkt auf den Speicherbus gegeben. Statt dessen werden sie der Speicherverwaltungseinheit (MMU) übergeben, die die virtuelle Adresse auf die Speicheradresse abbildet.

Die Abbildung 5 zeigt ein Beispiel, wie dieses Abbilden arbeitet. Der Computer erzeugt 16-Bit Adressen, von 0 bis 64k. Diese sind virtuelle Adressen. Der in diesem Beispiel verwendete Computer verfügt nur über 32k Hauptspeicher, obwohl 64k Programme geschrieben werden können. Diese können nicht vollständig in den Speicher geladen und bearbeitet werden. Eine vollständige Kopie des Speicherbildes des Programms (bis zu 64k) muß auf der Platte vorhanden sein, so daß Teile bei Bedarf eingelagert werden können.

Der virtuelle Adreßraum ist in Einheiten unterteilt, den sogenannten Seiten. Die entsprechenden Einheiten des Hauptspeichers heißen Seitenrahmen. Die Seiten und Seitenrahmen haben alle die gleiche Größe. In diesem Beispiel sind sie 4k groß. Der Transfer zwischen Speicher und Platte erfolgt immer in Einheiten von Seiten.

Was geschieht nun, wenn ein Programm eine nicht eingelagerte Seite verwenden will? Sobald der Computer dies erkennt wird der Ablauf unterbrochen. Diese Unterbrechung nennt man einen Seitenfehler[2]. Das Betriebssystem wählt einen wenig genutzten Seitenrahmen (mit Hilfe von Seitenersetzungsalgorithmen) aus und schreibt seinen Inhalt auf die Platte zurück. Es holt die gerade angesprochene Seite in den soeben freigewordenen Seitenrahmen und fährt mit der die Unterbrechung verursachenden Instruktion fort. [[TAN90]]

3.2 Seitenersetzungsalgorithmen

Seitenersetzungsalgorithmen kann man in zwei Arten unterteilen. Beim sogenannten „Demand-Paging“ wird bei Auftreten eines Seitenfehlers genau eine Seite, nämlich die Angeforderte in den Zwischenspeicher geladen. Alternativ dazu gibt es auch Strategien, die beim Auftreten einer Fehlseitenbedingung neben der angeforderten Seite noch weitere Seiten in den Puffer einlesen und zwar meist solche, die der angeforderten Seite auf dem Externspeicher physisch benachbart sind („Prepaging“). Diese Prepaging-Strategien[3] haben den Vorteil, daß das einmalige Lesen von m benachbarten Seiten geringere Kosten verursacht als das m-fache Lesen von je einer Seite. Andererseits besteht beim Prepaging eine bestimmte Wahrscheinlichkeit dafür, daß die im voraus eingelesenen Seiten nie

1. paging
2. page fault
3. siehe auch Anhang A

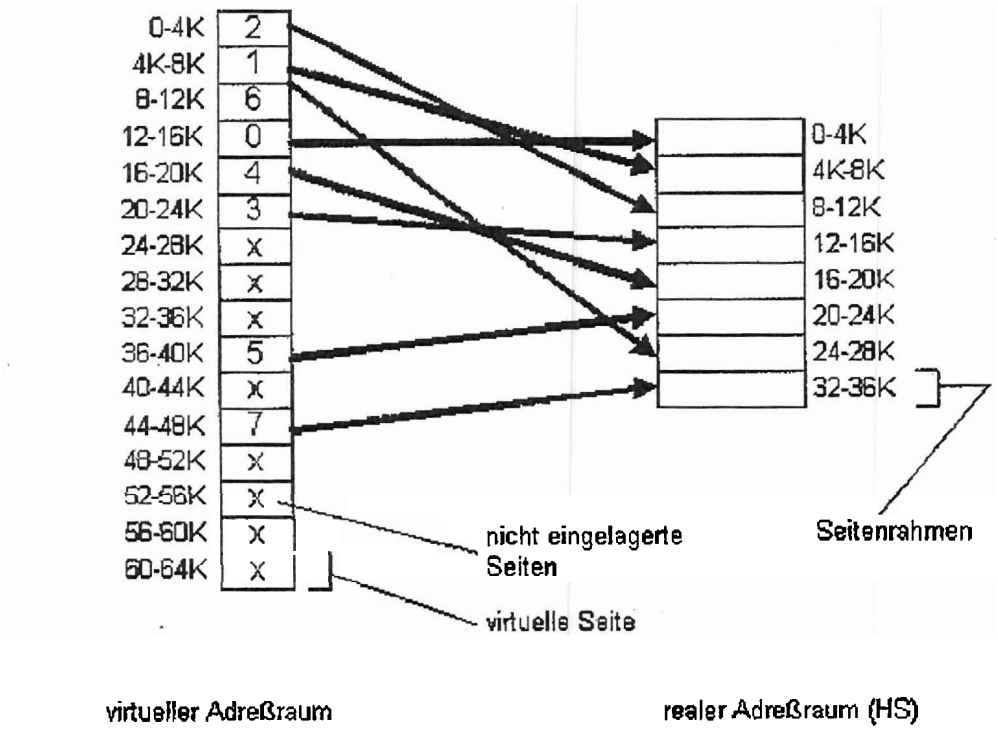


Abb. 5: Beziehung zwischen virtueller Adresse und Hauptspeicheradresse [[TAN90]]

gebraucht werden und daher einen unnötigen Leseaufwand verursachen. Weit aus schlimmer ist aber das Verdrängen andere Seiten aus dem Puffer, die vielleicht noch einmal benötigt werden. Die Anwendbarkeit von Prepaging-Strategien hängt somit stark von der physischen Sequentialität der Datenzugriffe ab. Im Rahmen dieser Arbeit wird nur auf Demand-Paging-Strategien eingegangen. Prepaging-Strategien gehören eher zu dem Thema Prepaging als zum Thema Caching.

Videofilme lassen sich, wie schon im Kapitel 3.3 erwähnt, entweder vollständig oder in Blöcken zerlegt übertragen. Ein ganzer Film bzw. ein Block würde demnach bei den Seitenersetzungsalgorithmen genau einer Seite entsprechen.

Gemeinsames Ziel der folgenden Strategien ist die Minimierung der Fehlseitenrate für eine fest vorgegebene Größe N des Systempuffers. Dieses Ziel erreicht man besten, indem man die Referenzwahrscheinlichkeit für eine Seite ermittelt. Die wichtigsten Kriterien sind dabei das Alter, die Referenzhäufigkeit und der letzte Referenzzeitpunkt der Seite im Speicher. Unter „Alter" wird hier die Zeit verstanden, die eine Seite seit ihrer letzten Einlagerung im Puffer verbracht hat. Sie ist im allgemeinen nicht identisch mit der Zeit, die seit der letzten Referenz vergangen ist, da eine Seite mehrfach angesprochen werden kann, während sie sich im Zwischenspeicher befindet. Die meisten Strategien benutzen eines oder mehrere der genannten Kriterien.

In allgemeinen konkurrieren stets mehrere Prozesse (Programme) um die Seitenrahmen[4]. Diese Seitenrahmen entsprechen der Kapazität eines Puffers und würden im Falle von VoD-Systemen den verfügbaren Speicherplätzen für die einzelnen Blöcke oder ganze Filme entsprechen. Die Aufgabe einer Ersetzungsstrategie ist nicht nur die Auswahl einer zu ersetzenden Seite eines einzelnen Prozesses, sondern auch die Entscheidung darüber, wie die insgesamt vorhandenen Seitenrahmen auf die parallelen Prozesse zu verteilen sind.

4. auch Pufferrahmen genannt

In diesem Zusammenhang ist die Unterscheidung zwischen lokalen und globalen Strategien hilfreich, die hier folgerndermaßen definiert sind. Eine Strategie heißt lokal, wenn sie ihre Ersetzungsentscheidung allein auf Grund des Referenzverhaltens des Prozesses fällt, der die Fehlseitenbedingung ausgelöst hat. Eine Strategie heißt global, wenn auch das Referenzverhalten konkurrierender Prozesse die Ersetzungsentscheidung beeinflußt.

Lokale Strategien lassen sich am einfachsten für feste Partitionen implementieren, d.h. jeder Prozeß erhält eine konstante Anzahl von Seitenrahmen, unabhängig von seinem aktuellen Referenzverhalten. Effizienter sind jedoch lokale Strategien mit variablen Partitionen, bei denen sich die Größe der Partitionen jedes einzelnen Prozesses seinem Referenzverhalten dynamisch anpaßt. [[EFF81]]

3.3 Möglichkeiten des Einsatzes in Videoverteilungssystemen

Zunächst sollen hier noch einige allgemeine Unzulänglichkeiten der Seitenersetzungsalgorithmen, bei der Anwendung in Videoverteilungssystemen angeführt werden:

Bei den folgenden Strategien sind die Seitengrößen als konstant anzunehmen. Dies stellt kein Problem dar, wenn die Filme in gleich große Blöcke zerlegt werden. Allerdings führt die Speicherung von einzelnen Blöcken zu den in Kapitel 3.3 erwähnten Problemen.

Werden die Filme nun als eine vollständige Einheit übertragen, so müßten die Seitenrahmen mindestens so groß sein, wie der größte Film. Dies bedeutet aber auch, daß bei Videos die weniger Speicherplatz benötigen nun *Kapazitäten verschwendet werden*, da die Speicherrahmen zu groß sind.

Da die Seitenersetzungsalgorithmen für virtuelle Speicher[5] entwickelt wurden und die Übertragungswege zwischen Platte und Speicher sehr kurz sind, werden bei den „konventionellen" Seitenersetzungsalgorithmen Faktoren wie *Ladezeit und Bandbreite* nicht berücksichtigt.. Doch gerade diese Faktoren spielen bei Video-on-Demand-Systemen eine große Rolle.

Ein weiterer großer Nachteil liegt darin, daß *neu aufgerufene Seiten auch immer im Puffer gespeichert werden.* In VoD-Systemen sollte zuerst geprüft werden, ob es sich überhaupt lohnt einen neuen Film zu speichern, wenn dafür ein anderer entfernt werden muß. Ist der neue Videofilm unpopulärer als jeder im Puffer gespeicherte Film, so sollte man ihn besser direkt an den Kunden übertragen, ohne ihn zu speichern.

Die hier aufgeführten Nachteile werden bei den einzelnen Bewertungen nicht mehr erwähnt, da man sie jedem der folgenden Algorithmen zurechnen kann.

Trotz dieser Mängel sollen Seitenersetzungsalgorithmen in dieser Arbeit nicht unterschlagen werden, da sie zu einem allgemeinem Überblick über vorhandene Ersetzungsalgorithmen gehören. Außerdem ist das Funktionsprinzip einiger der nachfolgenden Algorithmen für Videoverteilungssysteme sehr interessant und einer weiteren Untersuchung wert. Mit den erhaltenen Untersuchungsergebnissen lassen sich evtl. neue, für VoD-Systeme geeignete Algorithmen kreieren.

3.4 Konventionelle globale Seitenersetzungsalgorithmen

In diesem Abschnitt werden solche Strategien behandelt, die aus einer festen Anzahl von N Seiten eine zu ersetzende Seite auswählen. Der tatsächliche Bedarf einzelner Prozesse an Seiten wird nicht explizit ermittelt. Solche Verfahren können entweder global im gesamten Puffer oder lokal in festen Partitionen eingesetzt werden. [[EFF81]]

---

5. siehe Kapitel 4.1

3.4.1 FIFO (First-In-First-Out Algorithm)

Beschreibung:

Hier wird diejenige Seite ersetzt, die am längsten im Puffer ist, also die Seite mit dem höchsten Alter. Zur Veranschaulichung von FIFO kann man sich alle Pufferseiten kreisförmig angeordnet vorstellen. In der Mitte des Kreises befindet sich ein Zeiger, der zu jedem Zeitpunkt auf eine bestimmte Pufferseite zeigt. Dies ist die älteste Seite im Speicher (First-In): Tritt nun ein Seitenfehler auf, so wird die Seite ersetzt, auf die der Zeiger zeigt. Anschließend wird er auf die nächste Seite im Kreis fortgeschaltet.

Wegen ihrer Einfachheit ist die Strategie FIFO sehr beliebt. Sie hat jedoch den Nachteil, daß sie die Referenzhäufigkeit einer Seite während ihres Aufenthalts im Puffer in keiner Weise berücksichtigt. Seiten können also auch dann immer wieder aus dem Speicher verdrängt werden, wenn sie sehr viel häufiger angesprochen werden als andere Seiten im Puffer. Gut geeignet ist FIFO daher nur für strikt sequentielles Anfrageverhalten. [[EFF81]]

Parameter:

- Alter der einzelnen Seiten

Ergebnisse aus Studien:

In den Tabellen 5 und 6, sowie in der Abbildung 34 wird der FIFO Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Als Anwendung für vollständige Filme ist FIFO ein denkbar ungeeigneter Algorithmus, da die unterschiedliche Popularität der einzelnen Filme überhaupt nicht berücksichtigt wird.

FIFO ist nur geeignet, wenn der Film in einzelne Blöcke aufgeteilt wird. Ist das der Fall und sind schon einzelne Blöcke des Videos im Puffer, so werden die Blöcke nun nacheinander ein- und ausgeladen wie es der Reihenfolge entspricht.

3.4.2 LFU (Least Frequency Used Algorithm)

Beschreibung:

Hier wird die Seite mit der geringsten Referenzhäufigkeit ersetzt. Als Referenzhäufigkeit wird dabei die Anzahl der Referenzen auf die Seite, seit ihrer Einlagerung in den Puffer (absolute Häufigkeit) definiert. Um diese Referenzhäufigkeit ermitteln zu können, hält ein Zähler für jede Seite die Anzahl der Anfragen fest. Soll nun eine Seite ersetzt werden, wählt man diejenige mit dem niedrigsten Zählerstand aus. Für den Fall, daß zwei oder mehr Pufferseiten dieselbe Häufigkeit haben existiert eine Sekundärstrategie. Häufig wird dafür das LRU-Verfahren gewählt, aber auch FIFO oder RANDOM sind denkbar.

Bei näherer Betrachtung zeigt sich, daß LFU einen großen Nachteil besitzt. Da das Alter oder der letzte Referenzzeitpunkt einer Seite im Speicher keine Rolle spielen, kann eine Seite, die irgendwann einmal außerordentlich häufig hintereinander angesprochen wurden, wegen ihrer hohen Anfragehäufigkeit einen Seitenrahmen auf unbestimmte Zeit blockieren, auch wenn sie vielleicht nie mehr gebraucht wird. [[EFF81]]]

Um zu verhindern, daß einige Seiten einen so hohen Zählerstand erreichen, daß sie nie mehr gelöscht werden können, wird am Anfang ein sogenannter Durchschnittszähler definiert. Erreicht dieser Durchschnittszähler im Laufe der Anfragen eine bestimmte Höhe, so werden alle Zähler halbiert und der

Durchschnittszähler auf Null zurückgesetzt. Trotz allem tritt dabei noch folgendes Problem auf. Sucht der Algorithmus die am wenigsten benutzte Seite, wird er in den meisten Fällen die gerade neu hinzugekommene Seite finden und somit löschen. Neue Seiten haben also eine sehr schlechte "Überlebenschance". [[RL96]]

Deshalb ist der LFU Algorithmus nicht in seiner reinen Form, sondern nur in Modifikationen einsetzbar.

Der LFU Algorithmus ist auch bekannt als **NREF** (Number of References) Algorithmus.

Parameter:

- Anzahl der Anfragen an eine Seite

Implementierungsmöglichkeiten:

- Wahl des Zählerstandes des Durchschnittszählers, bei dem alle anderen Zählerstände halbiert werden

Ergebnisse aus Studien:

In der Tabelle 2, 5 und 6, sowie in den Abbildungen 10, 18, 34, 35, 36, 37, 38 und 23 wird der LFU Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Zunächst läßt sich sagen, daß die Häufigkeit der Anfragen bei Video-on-Demand-Systemen ein sehr wichtiges Auswahlkriterium ist. Die Referenzhäufigkeit entspricht schließlich der Popularität eines Filmes. Durch die in der Beschreibung aufgeführten Nachteile, wie Unlöschbarkeit bei lang zurückliegenden hohen Anfragen und Kurzlebigkeit von neu eingelagerten Filmen, erkennt man schon, daß der LFU eine unzureichende Strategie darstellt. Filme die man gerade gespeichert hat und von denen man annehmen kann, daß sie sehr populär werden, würden mit dieser einfachen Strategie sofort wieder gelöscht werden. Auch hat es keinen Sinn Videos, die vor mehreren Woche stark gefragt waren, nun aber überhaupt keine Nachfrage mehr haben, weiterhin im Speicher zu belassen.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

3.4.3 MRU (Most Recently Used Algorithm)

Beschreibung:

Bei diesem Algorithmus wird die Seite, die als letztes angesprochen wurde (also die „neueste" Seite) zur Ersetzung ausgewählt. Das widerspricht natürlich der Annahme, daß Seiten, die gerade erst eingeladen wurden auch in nächster Zeit noch benötigt werden. Deshalb wird dieser Algorithmus auch nicht in seiner reinen Form angewendet, sondern meist als interne Strategie in einem anderen Algorithmus[6] eingesetzt. Erwähnen kann man hier noch, daß der MRU Algorithmus quasi die Negation der LRU-Strategie darstellt. [[JCL90]]

Parameter:

- vergangene Zeit seit der letzten Anfrage

6. Dies ist z.B. im Priority-Hints Algorithmus auf Seite 35 oder dem SIZE-MRU Algorithmus auf Seite 67 der Fall.

Bewertung:

Wie schon in der Beschreibung erwähnt kann der MRU Algorithmus sinnvoll nur als interne Strategie in einem anderen Algorithmus eingesetzt werden. In diesem Fall läßt sich keine Bewertung für die Eignung in Videoverteilungssystemen vornehmen.

3.4.4 LRU-k

Beschreibung:

Der LRU-k Algorithmus ist, wie schon am Namen erkennbar, eine Erweiterung des LRU. Bei dieser Strategie wird nicht nur die letzte Referenz berücksichtigt, sondern die letzten K-Referenzen. Das bedeutet, daß z.B. bei LRU-2 die Seite ersetzt wird, deren zwei letzten Anfragen am weitesten zurückliegen. Daraus läßt sich auch leicht erkennen, daß der LRU-1 Algorithmus genau dem LRU Algorithmus entspricht.

Das LRU-k Verfahren berücksichtigt bei seiner Entscheidung die Referenzhäufigkeit einer Seite. Trotzdem entspricht es nicht der LFU-Strategie, denn es besteht ein entscheidender Unterschied zwischen den beiden Methoden. Der LRU-k besitzt eine eingebaute Vorstellung von „Alter" indem er nur die letzten K Referenzen berücksichtigt. LFU in seiner reinen Form hingegen hat keine Möglichkeit zwischen neueren und älteren Anfragehäufigkeiten zu unterscheiden. Es ist für ihn somit nicht möglich ein neues Anfragemuster zu erkennen, da die alten Anfragen immer noch in gleicher Weise mitberücksichtigt werden. [[OOW93]]

Parameter:

- vergangene Zeit seit den letzten K-Anfragen

Implementierungsmöglichkeiten:

- Wahl des Faktors K

Ergebnisse aus Studien:

Die erste Spalte der Tabelle 1 stellt die Puffergröße B dar. Die vier folgenden Spalten zeigen die Trefferquote von LRU-1, LRU-2, LRU-3 und A_0[7] und die letzte Spalte zeigt das effektive Größenverhältnis des Puffers von LRU-1 zu LRU-2.

Als erstes wird das Verhältnis B(1)/B(2) der obersten Reihe von Tabelle 1 betrachtet. Der B(2)-Wert stimmt mit dem B-Wert der Reihe (B = 60) überein, wobei die gemessene Trefferquote von LRU-2 den Wert 0,291 hat. Um dieselbe Trefferquote mit dem LRU-1 Algorithmus zu erreichen, benötigt man eine Puffergröße von 140 Seiten. Somit ergibt des Verhältnis B(1)/B(2) genau 2,3 (= 140/60). Hinsichtlich

7. A_0 stellt einen optimalen Algorithmus dar.

des Kosten/Leistungs-Verhältnisses übertrifft der LRU-2 den LRU-1 mit einem Faktor von mindestens zwei.

B	LRU-1	LRU-2	LRU-3	A_0	B(1)/B(2)
60	0.14	0.291	0.300	0.300	2.3
80	0.18	0.382	0.400	0.400	2.6
100	0.22	0.459	0.495	0.500	3.0
120	0.26	0.496	0.501	0.501	3.3
140	0.29	0.502	0.502	0.502	3.2
160	0.32	0.503	0.503	0.503	2.8
180	0.34	0.504	0.504	0.504	2.5
200	0.37	0.505	0.505	0.505	2.3
250	0.42	0.508	0.508	0.508	2.2
300	0.45	0.510	0.510	0.510	2.0
350	0.48	0.513	0.513	0.513	1.9
400	0.49	0.515	0.515	0.515	1.9
450	0.50	0.517	0.518	0.518	1.8

Tab. 1: Simulationsergebnisse für den LRU-k Algorithmus [[OOW93]]

Des weiteren erkennt man, daß die Resultate von LRU-3 sehr nahe an den Ergebnissen der Optimalstrategie A_0 herankommen. Dieses trifft allerdings nur bei stabilen (längere Zeit anhaltende) Anfragemustern zu. Sind diese Muster dagegen dynamische, so reaagiert der LRU-3 darauf langsamer, als der LRU-2. Der Grund liegt darin, daß der LRU-3 mehr Referenzen benötigt, um sich der veränderten Anfragefrequenz anzupassen. Deshalb wird der LRU-2 als ein generell effizienter Algorithmus bewertet und im allgemeinem dem LRU-3 vorgezogen. [OOW93]

B	LRU-1	LRU-2	LFU	B(1)/B(2)
100	0.005	0.07	0.07	4.5
200	0.01	0.15	0.11	3.25
300	0.02	0.20	0.15	3.0
400	0.06	0.23	0.17	2.75
500	0.09	0.24	0.19	2.4
600	0.13	0.25	0.20	2.16
800	0.18	0.28	0.23	1.9
1000	0.22	0.29	0.25	1.6
1200	0.24	0.31	0.27	1.66
1400	0.26	0.33	0.30	1.5
1600	0.29	0.34	0.31	1.5
2000	0.31	0.36	0.33	1.3
3000	0.38	0.40	0.39	1.1
5000	0.46	0.47	0.44	1.05

Tab. 2: Trefferquotenvergleich der Algorithmen LRU-k und LFU [[OOW93]]

Die Leistungswerte von LFU (Tabelle 2) sind erstaunlich gut. Der Hauptnachteil des LFU liegt jedoch in seiner mangelhaften Anpassung an neue Anfragemuster. Auf Grund dessen wird der LFU immer schlechtere Werte liefern, als der LRU-2 Algorithmus. [[OOW93]]

In den Tabellen 3 und 4, sowie in der Abbildung 40 wird der LRU-k Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Bei der Bewertung der LRU-Strategie wurde schon beschrieben, daß die Auswahl des zu ersetzenden Videos, allein auf der Grundlage der letzten Anfrage nicht zu einem optimalen Ergebnis führt. Mit dem LRU-k wird der Algorithmus nun verbessert, da er mit der Einführung eines k-ten Faktors nun auch in einem gewissen Umfang das Anfragemuster der Nutzer berücksichtigt. Ein Problem liegt nun in der Bestimmung eines geeigneten Wertes für K. Trotzdem genügt es bei Videoverteilungssystemen nicht, nur nach den letzten drei oder vier Anfragen die entsprechende Auswahl zu treffen. Eine Einschätzung über die Popularität eines Filmes läßt sich erst nach ein paar Dutzend Anfragen treffen.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

3.4.5 CLOCK

Beschreibung:

Bei diesem Verfahren wird sowohl das Alter als auch der letzte Referenzzeitpunkt einer Seite berücksichtigt. Der Name CLOCK stammt von der Veranschaulichung der Strategie durch das Zifferblatt einer Uhr.

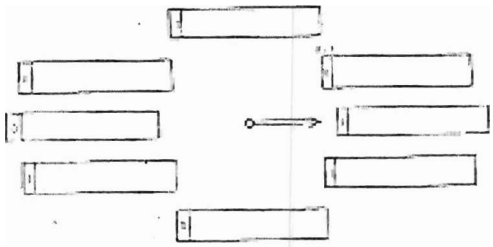


Abb. 6: Veranschaulichung der CLOCK-Strategie für 8 Seitenrahmen [[EFF81]]

Wie bei FIFO sind die Seitenrahmen kreisförmig angeordnet, und es gibt auch einen kreisenden Zeiger, der eine zu ersetzende Seite auswählt. Zusätzlich hat nun jeder Seitenrahmen ein Bit, das bei jeder Referenz der Seite auf 1 gesetzt wird. Beim Eintritt einer Fehlseitenbedingung wird zunächst die Seite geprüft, auf die der Zeiger zeigt. Ist deren Referenzbit = 0, so wird die Seite ersetzt und der Zeiger wird fortgeschaltet. Ist das Referenzbit = 1, so wird es auf 0 gesetzt, der Zeiger wird fortgeschaltet und die nächste Seite wird geprüft. Jede neu eingelagerte Seite überlebt also bei CLOCK im Gegensatz zu FIFO mindestens zwei volle Umdrehungen. Beim ersten Passieren des Zeigers wird lediglich ihr Referenzbit auf 0 gesetzt, die Seite bleibt jedoch im Speicher und erhält eine zweite Chance. Erst beim zweiten Passieren des Zeigers wird sie ersetzt, wenn sie nicht zwischendurch angesprochen wurde. Eine mögliche Variante von CLOCK wäre die Initialisierung des Referenzbits mit 0 bei der Erstreferenz.

Die Strategie CLOCK berücksichtigt in indirekter Weise den letzten Referenzzeitpunkt einer Seite durch das Referenzbit. Je kürzer die Zeitspanne seit der letzten Referenz ist, desto geringer ist die Wahrscheinlichkeit, daß das Referenzbit beim Passieren des Zeiger gleich Null ist. Aber auch das Alter einer Seite beeinflußt die Ersetzungsentscheidung: Unter allen Seiten, die nach ihrer Einlagerung nicht mehr angesprochen wurden, wird die älteste zuerst ersetzt. [[EFF81]]

Der CLOCK Algorithmus wird auch als **Second Chance** Algorithmus bezeichnet.

Parameter:

- Alter der einzelnen Seiten
- Anzahl der Anfragen an eine Seite (es kann aber immer nur eine Anfrage berücksichtigt werden)
- Vergangene Zeit seit der letzten Anfrage

Implementierungsmöglichkeiten:

- Einstellung des Zählers bei Einlagerung einer neuen Seite

Ergebnisse aus Studien:

In der Tabelle 3, sowie der Abbildung 30 wird der CLOCK Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Da das Alter eines Videos, sowie der Zeitpunkt der letzten Anfrage als Ersetzungsentscheidung in Videoverteilungssystemen nur langfristig geeignet ist, ist der GLOCK Algorithmus nicht brauchbar.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

3.4.6 GCLOCK (Generalized CLOCK Algorithm)

Beschreibung:

Die Strategie GCLOCK ist eine Erweiterung der Strategie CLOCK. Pro Pufferseite gibt es nicht nur ein Referenzbit, sondern zusätzlich einen Zähler. Wie bei CLOCK wird eine Seite nur ersetzt, wenn sie beim Kreisen des Zeigers mit einem Zählerwert = 0 vorgefunden wird.

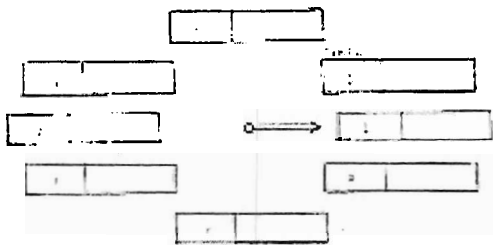


Abb. 7: Veranschaulichung der Strategie GCLOCK für 8 Seitenrahmen [[EFF81]]

Andernfalls wird der Zähler um 1 erniedrigt und der Zeiger fortgeschaltet. Dem Implementierer einer GCLOCK-Strategie bleiben nun zwei Entscheidungen überlassen:

- die Initialisierung des Zählers beim Einlagern einer neuen Seite,
- die Art der Inkrementierung des Zählers bei jeder Referenz, welche die Seite im Puffer vorfindet.

Dabei führen unterschiedliche Initialisierungen zu unterschiedlichen Überlebenschancen neu geladener Seiten im Systempuffer.

Eine Variante wäre dabei die Anwendung von GCLOCK für eine Prepaging-Strategie. Bei jeder Seitenreferenz wird nicht nur die angeforderte Seite in den Puffer geladen, sondern eine variable Anzahl weiterer, physisch benachbarter Seiten. Da diese zusätzlichen Seiten nicht mit Sicherheit benötigt werden (sie wurden ja nicht angefordert), sollten sie die von ihnen belegten Seitenrahmen nicht zu lange blockieren. Deshalb wurde durch Implementierung von GCLOCK die Möglichkeit vorgesehen, den zusätzlich eingelagerten Seiten eine geringere Überlebenschance im Puffer zu geben als der angeforderten Seite.

Die Möglichkeit einer veränderlichen Initialisierung stellt eine weiter Variante der GCLOCK-Strategie dar. Diese Modifikation basiert auf der Idee, daß man während der Bearbeitung der Transaktionslast die Überlebenschancen derjenigen Seiten gezielt erhöhen sollte, die gerade eine überdurchschnittlich

hohe Fehlseitenrate haben. Der Algorithmus mit dieser Eigenschaft wird mit **DGCLOCK** (dynamischer GCLOCK) bezeichnet.[[EFF81]]

Parameter:

- Alter der einzelnen Seiten
- Anzahl der Anfragen an eine Seite

Implementierungsmöglichkeiten:

- Einstellung des Zählers bei Einlagerung einer neuen Seite
- Art der Inkrementierung des Zählers bei jeder Referenz (Gewichtung der Seiten)

Ergebnisse aus Studien:

In der Tabelle 3, sowie der Abbildung 30 wird der GCLOCK Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Durch die Implementierungsmöglichkeiten ist der GCLOCK Algorithmus interessanter, als sein Vorgänger. Bei Video-on-Demand-Systemen könnte man Filme die einen große Anfrage erwarten lassen mit einem hohen Zählerwert einlagern. Andere Filme dagegen werden mit niedrigeren Zählerwerten eingelagert und können außerdem mit einem höheren Inkrementierungswert belastet werden. Damit lassen sich Videos unterschiedlich gewichten. Der Nachteil liegt aber darin, daß man im Voraus entschieden muß zu welcher „Klasse" ein Film zählt. Der Algorithmus paßt sich also nicht dem Anfragemuster an, statt dessen hängt der Nutzen der Strategie von Vorentscheidungen ab. Außer man verwendet eine dynamische Initialisierung und somit den DGLOCK Algorithmus.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

3.4.7 LRD (Least Reference Density Algorithm)

Beschreibung:

Der Algorithmus LRD funktioniert wie folgt: Es wird diejenige Seite aus dem Zwischenspeicher ersetzt, welche die geringste Referenzdichte hat. Die Referenzdichte ist dabei definiert als die Referenzhäufigkeit während eines bestimmten Referenzintervalls. Im Unterschied zu LFU geht nicht die absolute Häufigkeit der Anfragen in den LRD ein, sondern nur die Häufigkeit seit Intervallanfang. Durch hinreichend kurze Intervalle ist es möglich, stark gehäufte Referenzen die schon sehr lange zurückliegen, nicht mehr in die Vorhersage des Anfrageverhaltens in naher Zukunft eingehen zu lassen. Auf diese Weise wird die Lokalität im Referenzverhalten besser berücksichtigt als bei LFU. Bei der Suche nach einer Seite mit der geringsten Referenzdichte kann sich nun wie bei LFU zeigen, daß zwei oder mehr Pufferseiten dieselbe Referenzdichte haben. Auch hier muß eine Sekundärstrategie die letzte Entscheidung fällen.Zur Festlegung der Intervalle, auf welche die Referenzdichte bezogen ist, sind verschiedene Verfahren denkbar. Zwei davon werden im folgenden kurz dargestellt:

Variante 1:

Die Intervallgröße ist identisch mit dem Alter der Seite, also der Anzahl der Referenzen, die seit der Einlagerung der Seite in den Puffer vergangen ist. Diese Variante bewertet zwei Seiten mit gleicher Referenzhäufigkeit nach ihrem Alter, d.h. die älteste Seite wird ersetzt. Im Beispiel der Abbildung 8 wird

deutlich, wie sowohl Referenzhäufigkeit als auch Alter in die Referenzdichte der einzelnen Seiten eingehen.

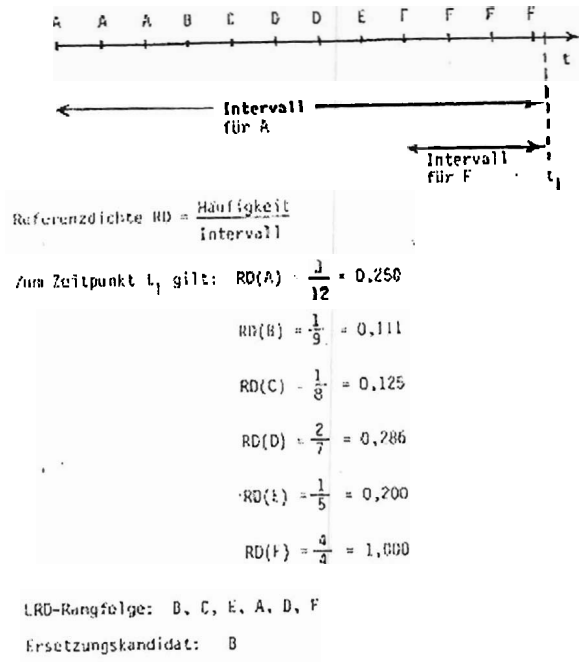


Abb. 8: LRD, Variante 1, Intervall = Alter der Seite [[EFF81]]

Variante 2:

Die Intervallgröße ist konstant. Jeweils nach einer festen Anzahl von Anfragen wird die Ersetzungsinformation sämtlicher Seiten aktualisiert, z.B. durch Subtraktion einer Konstanten[8] von allen seitenbezogenen Referenzzählern. Dadurch läßt sich ein periodisches Altern von früheren Referenzen implementieren, so daß häufig auftretende Referenzen um so weniger berücksichtigt werden, je älter sie sind. Die Anfragen aus dem aktuellen Intervall gehen voll in die Dichte ein, ältere Referenzen in geringerem Maße. Ein Beispiel wird in der Abbildung 9 dargestellt.

Beide Varianten der LRD-Strategie lassen sich sehr einfach mit Hilfe eines Referenzzählers RZ durchführen, der für jede einzelne Seite die Anfragehäufigkeit zählt. Für Variante 1 wird zusätzlich das Alter der Pufferseite benötigt, welches durch den Einlagerungszeitpunkt EZ dargestellt wird. Daneben gibt es einen globalen Zähler GZ, der bei jeder logischen Anfrage inkrementiert wird. Bei der Einlagerung einer neuen Seite i wird jeweils der Wert GZ nach EZ(i) gespeichert und RZ(i) auf 1 gesetzt. Die Referenzdichte einer Seite zu einem späteren Zeitpunkt ist dann gegeben durch folgende Gleichung:

$$RD_i = RZ_i / (GZ - EZ_i)$$

Für Variante 2 der Strategie LRD gibt es mehr Möglichkeiten der Einflußnahme bei der Implementierung. Sowohl die Intervallgröße als auch die Dekrementierungskonstante sind frei wählbar. [[EFF81]]

Parameter:

- Intervallgröße (window size)

8. Dekrementierungskonstante

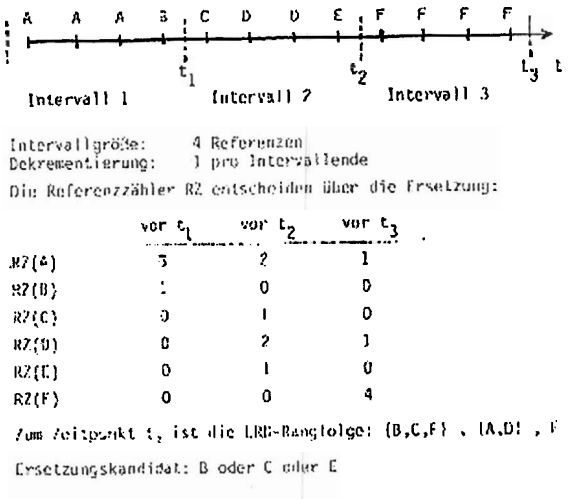


Abb. 9: LRD, Variante 2, konstantes Intervall („Periodisches Altern") [[EFF81]]

- EZ_i = Einlagerungszeitpunkt einer Seite i (Alter der einzelnen Seiten)
- GZ = Zählerstand des Globalen Zählers (Anfragen des Systems insgesamt)
- RZ_i = Zählerstand des Referenzzählers für eine Seite i (Anfragen in einem Intervall)
- RD_i = Referenzdichte einer Seite i

Implementierungsmöglichkeiten:

- Wahl der Intervallgröße
- Wahl der Dekrementierungskonstante

Ergebnisse aus Studien:

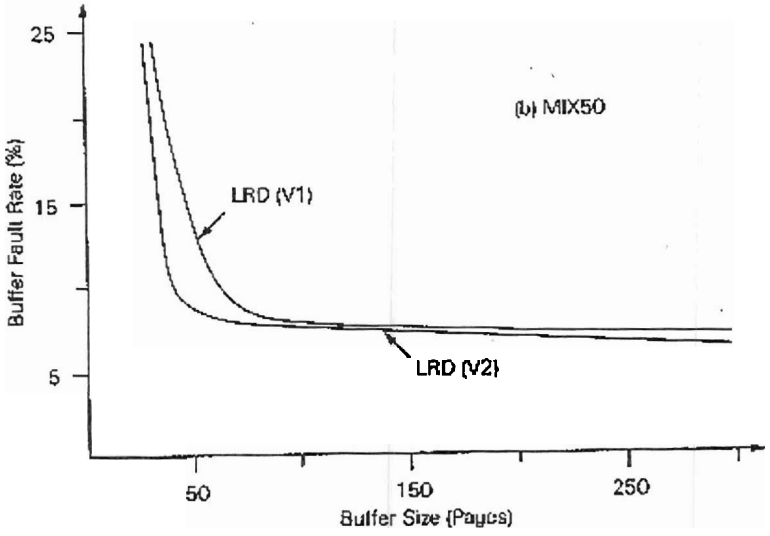


Abb. 10: Fehlerquote der LRD-Strategie [[EH84]]

Für LRD (V2) wurden folgende Parameter verwendet: Intervallgröße = 10, Dekrementierungskonstante = 2 nach jeder dritten Anfrage. Durch die guten Anpassungsmöglichkeiten von LRD (V2) an das spezifische Referenzverhalten erhält man für LRD (V2) bessere Werte, als für LRD (V1). [[EH84]]

Bewertung:

Der LRD Algorithmus ist für VoD-Systeme nicht uninteressant. Durch die Benutzung von Intervallen läßt sich der Nachteil von LFU kompensieren. Anfragen die weiter zurückliegen werden nun weniger berücksichtigt, als neue Anfragen. Dieses periodische Altern (mit Hilfe der Dekrementierungskonstante) läßt Änderungen im Anfragemuster stärker hervortreten. Die meist hohen Anfragen bei neuen Filmen gehen bei dieser Strategie mehrere Wochen (Intervalle) später nicht mehr oder nur in geringem Maße in die Ersetzungsentscheidung ein. Referenzdichten stellen für Videoverteilungssysteme somit ein besseres Entscheidungs-kriterium dar, als die reine Referenzhäufigkeit.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

3.4.8 2Q (Two Queue)

Beschreibung:

Beim 2Q Algorithmus werden neue Seiten zunächst in einen speziellen Puffer, der A1-Reihe, gespeichert. Diese A1-Reihe wird mittels der FIFO-Strategie verwaltet. Wird nun eine in der A1-Schlange gespeicherte Seite wiederum aufgerufen, so wechselt diese Seite in die Am-Reihe. In dieser Reihe wird der LRU Algorithmus verwendet. Wird die in der A1-Reihe gespeicherte Seite nicht mehr aufgerufen, dann wird sie aus dem Puffer ausgewiesen.

Problematisch stellt sich die Verteilung der verfügbaren Seitenrahmen auf die A1- und die Am-Reihe dar. Wird A1 zu klein gewählt, so ist es für eine Seite überaus schwierig überhaupt einmal in die Am-Schlange zu gelangen. Wenn die Größe der A1-Reihe zu groß gerät, dann stehen zu wenig Seitenrahmen für den Am-Teil zur Verfügung. Dadurch sinkt die Leistung, da der eigentliche Puffer nun zu wenig Seiten aufnehmen kann.

Die Lösung des Problems liegt darin, nur die Hinweise (Adressen) auf die Seiten in der A1-Schlange zu speichern und nicht die Seite selber. Damit kommt A1 mit sehr wenig Speicherplatz aus und Am kann all seine Seitenrahmen für eigene Zwecke verwenden. Allerdings funktioniert dieses Verfahren nur, wenn man eine stationären Verteilung voraussetzt. In der Realität muß man aber davon ausgehen, daß sich die Aufenthaltsorte der Seiten sehr schnell ändern können und die Hinweise (Adressen) damit schnell veralten.

Es gibt aber auch noch eine andere Möglichkeit den Platzbedarf von A1 zu minimieren. Zunächst unterteilt man A1 in A1in und A1out. Die neuesten der angefragten Seiten werden in der A1in-Reihe gespeichert, ältere müssen aus dieser Reihe entfernt werden. Deren Adressen werden jedoch dafür in der A1out-Reihe gespeichert. Ein gegebener Puffer unterteilt sich somit in Am, A1in (maximale Größe Kin, minimale Größe 1) und A1out (maximale Größe Kout). [[JS94]]

Parameter:

- vergangene Zeit seit der letzten Anfrage (LRU-Prinzip)
- Alter der einzelnen Seiten (FIFO-Prinzip)
- K = Anzahl der gespeicherten Seiten

Implementierungsmöglichkeiten:

- Einteilung des Speichers in Am, A1 (bzw. A1in und A1out)

Ergebnisse aus Studien:

Number of page slots	LRU/2	LRU	Gclock	2nd Chance	2Q Kin=30%	2Q Kin=20%
100	.086	.083	.083	.083	.096	.090
200	.164	.144	.144	.141	.196	.181
500	.284	.234	.236	.223	.334	.329
1000	.384	.328	.327	.318	.405	.405
2000	.454	.425	.425	.418	.465	.464
5000	.544	.537	.538	.532	.556	.557
10000	.616	.607	.607	.602	.626	.624
20000	.678	.671	.671	.665	.681	.680

Tab. 3: Trefferquotenvergleich der Algorithmen LRU-2, LRU, GCLOCK, 2nd-Chance[1] und 2Q [[JS94]]

1. Der 2nd Chance Algorithmus wird in dieser Arbeit unter dem Namen CLOCK geführt.

Bewertung:

Aufgrund des sehr großen Speicherplatzbedarfs eines einzigen Filmes, besteht bei Videoverteilungssystemen keine Möglichkeit den Speicher in 2 oder 3 Segmente aufzuteilen. Die Idee, nur die Adressen der Filme zu speichern ist zwar nicht schlecht, wird aber in einem dynamischen System, wie es das Video-on-Demand-System eines ist, nicht funktionieren.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

3.5 Konventionelle lokale Seitenersetzungsalgorithmen

Die im vorigen Abschnitt dargestellten Ersetzungsstrategien waren lediglich in der Lage, aus einer festen Anzahl N von Pufferseiten eine bestimmte zur Ersetzung auszuwählen. Eine Abschätzung des Speicherbedarfs jeder einzelnen Transaktion in Abhängigkeit von ihrem aktuellen Referenzverhalten wurde nicht geleistet. Strategien dieses Typs lassen sich, wie bereits erwähnt, entweder global einsetzen, wobei sie ohne Kenntnis von Transaktionszugehörigkeiten ihre Ersetzungsentscheidung fällen, oder sie werden lokal in festen Partitionen P_1 , P_2 , ... P_N eingesetzt, wobei jeder von N parallelen Transaktionen eine Partition fest zugeordnet wird. Im Gegensatz dazu sind lokale Strategien mit variablen Partitionen in der Lage, entsprechend dem aktuellen Speicherbedarf die Partitionsgröße jeder Transaktion dynamisch wachsen und schrumpfen zu lassen. Zwar hat auch bei globalen Verfahren jede Transaktion eine variierende Anzahl von Seitenrahmen zu ihrer Verfügung, doch hängt die Zuteilung nicht allein vom Referenzverhalten der Transaktion selbst ab, sondern vor allem von den parallel ablaufenden Transaktionen. Dagegen arbeiten lokale Strategien nach dem Prinzip „jedem nach seinem eigenen Bedarf". Eine Transaktion gibt unter einer lokalen Strategie „freiwillig" Seiten ab, wenn sie für einen bestimmten Zeitabschnitt mit weniger Seiten effizient arbeiten kann. Dadurch werden Seitenrahmen verfügbar, die eventuell einer neu zu startenden Transaktion zur Verfügung gestellt werden können.[[EFF81]]

3.5.1 WS (Working Set Algorithm)

Beschreibung:

Die Working-Set-Strategie ist primär ein Speicherzuteilungsverfahren mit variablen Partitionen. Als Working-Set (Arbeitsmenge) wird hier die Menge derjenigen Seiten bezeichnet, die innerhalb der letz-

ten y Referenzen von der betrachteten Transaktion angesprochen wurden. Der Parameter y heißt Fenstergröße (window-size).

Die Grundidee der WS-Strategie ist die Annahme, daß der Working-Set einer Transaktion zum Zeitpunkt t mit großer Wahrscheinlichkeit gerade die Seiten enthält, die auch in näherer Zukunft benötigt werden. Sie versucht daher, jeder Transaktion ihre Arbeitsmenge zur Verfügung zu stellen. Wenn eine Transaktion in einer Phase hoher Lokalität[9] gerät, verkleinert sich ihr Working-Set bei konstanter Fenstergröße. Sie versucht dann nicht, die ihr zuvor zugeteilten Seitenrahmen zu behalten und in der Lokalitätsphase effizienter abzulaufen, sondern gibt einige ihrer Seitenrahmen ab (variable Partitionsgröße).

Im Unterschied zu allen bisher dargestellten Ersetzungsstrategien läßt sich bei der WS-Strategie keine Rangfolge aller Seiten bezüglich der Ersetzung angeben. Alle Seiten, die zum Ersetzungszeitpunkt nicht zur Arbeitsmenge einer aktiven Transaktion gehören dürfen ohne Unterschied ersetzt werden. Dieses Verfahren trifft also nur die Entscheidung „ersetzbar" oder „nicht ersetzbar". Daraus läßt sich erkennen, daß das primäre Ziel von WS die optimale Verteilung der Betriebsmittel „Seitenrahmen" an die konkurrierenden Transaktionen (optimale Speicherzuteilung) ist. Lokalität im Referenzverhalten wird nur innerhalb des Fensters der Größe y berücksichtigt. Die aktuelle Auswahl aus der Menge der ersetzbaren Seiten kann durch eine beliebige Sekundärstrategie (z.B. LRU) vorgenommen werden. [[EFF81]][[TAN90]]

Parameter:

- Window-size
- Seiten die zur jeweiligen Arbeitsmenge gehören

Ergebnisse aus Studien:

In der Abbildung 13 wird der WS Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Das zyklische Verhalten von Programmschleifen gibt es in Videoverteilungssystemen anschaulich nicht. Daher muß der WS Algorithmus anders interpretiert werden. Statt konkurrierende Programmen gibt es konkurrierende Nutzer, die natürlich alle den jeweils gewünschten Film im Puffer gespeichert haben möchten (da so ihre Anfragen schneller erfüllt werden können). Eine Möglichkeit wäre z.B. die Nutzer in verschiedene Zuschauergruppen einzuteilen. Dabei lassen sich verschieden Einteilungskriterien vorstellen, wie z.B. Alter der Nutzer, Nutzungshäufigkeit des Systems etc. Jeder dieser Zuschauergruppen kann nun eine Anzahl von Seitenrahmen zugeordnet werden.

Nimmt man an, daß ein Puffer 15 Filme speichern kann, so könnte man der Altersgruppen von 6-10 drei Seitenrahmen (z.B. für Zeichentrickfilme), der Altersgruppe 11-16 zwei Seitenrahmen (z.B. für Abenteuerfilme), der Altersgruppe 16-25 fünf Seitenrahmen (z.B.für Oskar prämierte Filme) usw. zurechnen. Wird festgestellt, daß eine Altersgruppe in einem bestimmten Intervall nicht vertreten ist (z.B. die Altersgruppe von 6-10 wird zwischen 21 und 6 Uhr keine Filme anfordern), werden die freigewordenen Seitenrahmen auf andere Altersgruppen verteilt.

Eine weitere Möglichkeit wäre es, die Filme in verschieden Klassen einzuteilen, wie z.B. Western, Science-fiction, Dramen, Action-Filme, Stummfilme, Heimatfilme usw. Jeder dieser Klassen würden dann eine entsprechende Zahl an Speicherrahmen zugeordnet werden.

Wird ein Film in Blöcken zerlegt übertragen, so kann man mit dieser Strategie festlegen, wieviele Blöcke ein Video in den Puffer laden darf.

---

9. Der Prozeß greift für längere Zeitabschnitte immer nur auf eine kleine Untermenge, der in seinem Adreßraum vorhandene Seiten.

3.5.2 PFF (Page Fault Frequency Algorithm)

Beschreibung:

Die Strategie PFF ist der Working-Set-Strategie sehr verwandt. Sie benutzt die aktuelle Fehlseitenrate eines Prozesses zur Entscheidung über die Zuordnung von Seitenrahmen. Als Parameter der Strategie wird eine Soll-Fehlerrate F vorgegeben. Sobald die tatsächliche Fehlseitenrate einer Transaktion größer als F wird, erhält sie zusätzliche Seitenrahmen, bis die Fehlseitenrate wieder auf den Sollwert gesunken ist. Wird dagegen die tatsächliche Fehlseitenrate kleiner als F, so gibt der Prozeß Seitenrahmen ab, bis F wieder erreicht ist. Zur Implementierung von PFF muß lediglich die Fehlseitenrate F_T jeder Transaktion ermittelt werden. Ist zum Ersetzungszeitpunkt $F_T > F$, so wird die neu einzulesende Seite in einem neuen Rahmen untergebracht. Der Prozeß erhält also einen zusätzlichen Rahmen. Ist jedoch zum Ersetzungszeitpunkt $F_T < F$, so werden alle Seiten freigegeben, die seit der letzten Fehlseitenbedingung nicht angesprochen wurden. Im Unterschied zum WS-Verfahren wächst und schrumpft also der Working-Set einer Transaktion nur zum Zeitpunkt einer Fehlseitenbedingung. Allgemein läßt sich sagen, daß der PFF Algorithmus eine schnelle Reaktion auf ein plötzliches Anwachsen oder Abfallen in den Speicherplatzanforderungen liefert. [[EFF81]]

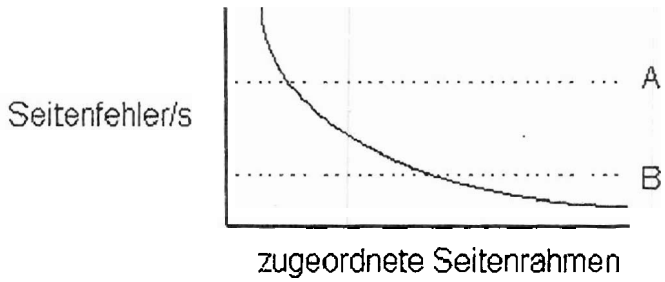


Abb. 11: Seitenfehlerrate als Funktion der Anzahl der zugeordneten Seitenrahmen [[TAN90]]

Parameter:

- Fehlseitenrate F_T jedes Prozesses
- Zeit seit der letzten Anfrage für jede Seite

Implementierungsmöglichkeiten:

- Einstellung der Soll-Fehlseitenrate F

Bewertung:

Auch hier könnte man, wie beim WS Algorithmus eine Einteilung nach Zuschauergruppen oder Filmklassen vornehmen. Jedoch würde durch die ständige Anpassung an das Anfrageverhalten Filme ständig ein- und ausgeladen werden. Dies ist in VoD-Systemen wenig sinnvoll, da das Speichern von Filmen sehr lange dauert und hohe Kosten verursacht. Speicherplatzzuordnungen sollten sich besser an langfristiges Verhalten (mehrere Stunden), als an kurzfristige Verhaltensänderungen orientieren.

Wird ein Film in Blöcken zerlegt übertragen, so kann man mit dieser Strategie festlegen, wieviele Blöcke ein Video in den Puffer laden darf.

3.6 Verwendung eines Inter-Reference Gap Models

Die nachfolgenden Algorithmen basieren auf Vorhersagen der zukünftigen Anfragen. Als Grundlage für diese Vorhersagen verwendet man einen sogenannten IRG (Inter-Reference Gap). Dabei definiert man den IRG für eine Seite, als den Zeitintervall zwischen aufeinanderfolgenden Referenzen für ein und dieselbe Seite. Der IRG stream für eine Seite stellt dagegen die Sequenz von hintereinander gelegenen IRG values dieser Seite dar.

Zur Veranschaulichung dient folgendes Beispiel: Eine Seite a wird zu den Zeiten t_1, t_2, t_3, t_4 usw. nachgefragt. Somit ergibt sich ein IRG stream für a: $t_2 - t_1$, $t_3 - t_2$, $t_4 - t_3$ usw. Diese Zeitwerte sind dabei nur virtuell zu sehen. Das heißt, jede Speicheranfrage wird in einer Zeiteinheit abgewickelt. Die reale absolute Zeit zwischen aufeinanderfolgenden Anfragen ist hier unwesentlich.

Mit einer Markov Kette wird nun für jede dieser IRG streams ein Modell erstellt. Unter der Verwendung der vergangenen IRG values liefert dieses Modell eine Vorhersage für die Schätzung des zukünftigen IRG value. Diese Vorhersage liefert ein Level z predictor. Er schätzt die Wahrscheinlichkeit des nächsten Symbols X_t (mit dem Wert x) mit Hilfe folgender Formel:

$$\Pr\{X_t = x / X_i = x_i, 1 < i < t - 1\} = m_x / N_{t-1}$$

N_{t-1} stellt dabei die Anzahl der Ereignisse von X_{t-z}^{t-1} in X_1^{t-2} dar und m_x ist die Anzahl der Ereignisse von X_{t-z}^{t-1} mit dem verketteten x in X_1^{t-1}.

Zum näheren Verständnis wird im nun ein Beispiel ausführlich berechnet:

Folgende Anfragereihe ist gegeben: „bcaababbaccacabcab+acda". Die Seite a wurde zu den Zeitpunkten 3,4,6,9,12,14,17,19,22 nachgefragt. X_1^9 = 3 1 2 3 3 2 3 2 3 stellt somit die zugehörige IRG Reihe dar.

Welchen Wert wird nun der nächste IRG value X_{10} annehmen? Mit dem obigen Modell lassen sich die Wahrscheinlichkeiten für verschiedene Werte berechnen. Dabei geht man wie folgt vor:

Level 2:

Die zwei letzten IRG values werden besonders berücksichtigt: $X_{t-z}^{t-1} = X_8^9$, wobei $X_8 = 2$ und $X_9 = 3$. Dadurch ergibt sich die Folge 23, die im Bereich $X_1^{t-2} = X_1^8$ genau zweimal vorkommt, d.h. $N_{t-1} = 2$. Nun soll geschätzt werden mit welcher Wahrscheinlichkeit der zehnte IRG value eine 1, 2 oder 3 sein wird. Dabei wird die obige Folge durch den möglichen IRG value (nämlich X_{10}) ergänzt und anschließend die gegebene Reihe auf die Häufigkeit des Vorkommens der Folge überprüft.

Die Folge 2 3 1 kommt überhaupt nicht , die Folge 2 3 2 kommt einmal und die Folge 2 3 3 kommt auch einmal in der Reihe vor.

Mit diesen Erkenntnissen ergeben sich folgende Werte:

$m_x = 0$ (für $X_{10} = 1$), $m_x = 1$ (für $X_{10} = 2$), $m_x = 1$ (für $X_{10} = 3$)

$N_{t-1} = 2$ (Wert von X_{10} hat hier keinen Einfluß)

Level 2: $\Pr\{X_{10} = 1 / X_8 = 2, X_9 = 3\} = 0$

$\Pr\{X_{10} = 2 / X_8 = 2, X_9 = 3\} = 0{,}5$

$\Pr\{X_{10} = 3 / X_8 = 2, X_9 = 3\} = 0{,}5$

Aus diesem Ergebnis läßt sich erkennen, daß die Wahrscheinlichkeit mit der X_{10} eine 2 oder eine 3 sein wird jeweils 50% beträgt.

Level 1:

Hier wird nur der letzten IRG value besonders berücksichtigt: $X_{t-z}^{t-1} = X_9^9$, wobei $X_9 = 3$. Dadurch ergibt sich die Folge 3, die im Bereich $X_1^{t-2} = X_1^8$ genau viermal vorkommt, d.h. $N_{t-1} = 4$.

Die obige Folge wird nun wiederum um den jeweiligen Wert von X_{10} ergänzt. Dabei kommt die Folge 3 1 einmal, die Folge 3 2 zweimal und die Folge 3 3 einmal in der Reihe vor.

Mit diesen Erkenntnissen ergeben sich folgende Werte:

$m_x = 1$ (für $X_{10} = 1$), $m_x = 2$ (für $X_{10} = 2$), $m_x = 1$ (für $X_{10} = 3$)

$N_{t-1} = 4$

Level 1: $Pr\{X_{10} = 1 / X_9 = 3\} = 0{,}25$

$Pr\{X_{10} = 2 / X_9 = 3\} = 0{,}5$

$Pr\{X_{10} = 3 / X_9 = 3\} = 0{,}25$

Aus diesem Ergebnis läßt sich erkennen, daß die Wahrscheinlichkeit mit der $X_{10} = 1$ oder $X_{10} = 3$ jeweils 25% und bei $X_{10} = 2$ sogar 50% beträgt.

Level 0:

Hier wird nun nicht untersucht, wie oft eine Folge in der Vergangenheit aufgetreten ist, sondern alleine die Häufigkeit von X_{10} in der vorgegebenen Reihe entscheidet über das wahrscheinliche Auftreten des jeweiligen Wertes. Der Wert 1 kommt nur einmal, der Wert 2 dagegen schon dreimal und der Wert 3 sogar fünf mal in der Reihe vor.

Damit ergeben sich folgende Wahrscheinlichkeiten:

Level 0: $Pr\{X_{10} = 1\} = 0{,}11$

$Pr\{X_{10} = 2\} = 0{,}33$

$Pr\{X_{10} = 3\} = 0{,}55$

Der Grund für diese verschiedenen Schichten von predictors liegt darin, daß man ein System entwickeln möchte, welches noch gute Ergebnisse auch bei einem Ausfall eines k-ten predictors liefert. Das Scheitern eines level k predictors tritt ein, wenn X_{t-z}^{t-1} nicht in der Reihe X_1^{t-2} vertreten ist ($N_{t-1} = 0$). In so einem Fall wechselt man zu dem level k-1 predictor und falls nötig auch noch zu unteren levels. [PG95]

3.6.1 IRG-k (Inter Reference Gap Algorithm)

Beschreibung:

Wird die angefragte Seite im Zwischenspeicher nicht gefunden, wird eine Vorhersageroutine[10] aufgerufen, welche die Seite mit der größten Vorwärtsdistanz (dies entspricht der am weitesten entfernten vorhergesagten Distanz) finden soll. Ist dieser Prozeß der Schätzung nicht erfolgreich (aus Gründen die im folgendem noch erläutert werden), so wird das LRU-Verfahren angewendet. Andernfalls ersetzt man die Seite mit der größten vorhergesagten Vorwärtsdistanz.

Der predictor der diese Distanz für jede Seite schätzen soll bedient sich dabei einem IRG Modell der Klasse k. Soll nun die Vorwärtsdistanz einer Seite berechnet werden, so wird ein level k predictor von z = k bis z = 0 angewendet. Dieser Prozeß endet sobald man einen IRG Wert größer als die aktuelle Lücke von a vorhersagt.

Zum besseren Verständnis dient noch einmal das obige Beispiel:

Die Seite a wird während der Zeiteinheiten 23 und 24 nicht nachgefragt, d.h. die aktuelle Lücke von a beträgt 2. Mit der Verwendung eines IRG Modells der Klasse 2 wird nun der level 2 predictor zur Berechnung der entsprechenden Distanz herangezogen. Dieser schätzt den nächsten IRG auf 3 (da 2 mittlerweile schon unmöglich ist). Ist die aktuelle Lücke größer als alle vergangenen IRGs dann scheitert der predictor.In solch einem Fall wird, wie oben schon erwähnt der LRU Algorithmus benutzt.[PG95][PG97]

Parameter:

- IRG-Modell
- Größe der aktuellen Lücke
- Als Sekundärstrategie: LRU

Ergebnisse aus Studien:

Aus der Tabelle 4 lassen sich zwei wichtige Aussagen ziehen. Erstens läßt sich erkennen, daß IRG1 nur geringfügig besser ist als IRG0. Tatsächlich ist es sogar so, daß in manchen Fällen IRG1 schlechter als IRG0 ist. Der Hauptgrund liegt in der langsameren Anpassung von IRG1 bei drastischen Änderungen im IRG stream. Das bedeutet, daß bei drastischen Änderungen im IRG stream, IRG1 mehr unkorrekte Schätzungen liefert, als der IRG0.

Algorithm	Cache Size (number of words)						
	512	1024	2048	4096	8192	16384	32768
LRU	0.429	0.343	0.286	0.216	0.142	0.064	0.045
LRU2	0.453	0.375	0.309	0.236	0.139	0.084	0.054
LRU3	0.463	0.384	0.309	0.223	0.147	0.096	0.051
IRG0	0.386	0.320	0.265	0.204	0.142	0.064	0.045
IRG1	0.380	0.315	0.262	0.203	0.142	0.064	0.045
IRG2	0.378	0.315	0.261	0.194	0.135	0.062	0.045
OPT	0.313	0.246	0.188	0.130	0.075	0.048	0.040

Tab. 4: Fehlerquotenvergleich der Algorithmen LRU-k und IRG-k [PG95]

10. siehe obige Einführung über IRG

Zweitens entsprechen die Werte bei großen Speichern der Fehlerquote von LRU. Dies liegt an der Unfähigkeit von IRG bei großen Speichern zukünftige Entwicklungen richtig vorauszusagen. Je größer ein Puffer ist, desto weniger Informationen stehen über die einzelnen Seiten zur Verfügung, da die Seiten trotz geringer Anfgragehäufigkeit im Puffer gespeichert werden können. Somit wird der predictor meist versagen und statt dessen die vereinbarte Sekundärstrategie angewendet, der LRU Algorithmus. Andererseits stehen bei einem kleinen Speicher für die einzelnen Seiten eine lange Reihe von Daten zur Verfügung mit den gute Schätzungen möglich sind. [PG95]

Die Beziehung zwischen LRU, IRG0 und MIN[11] wird in der Abbildung 12 noch einmal verdeutlicht.

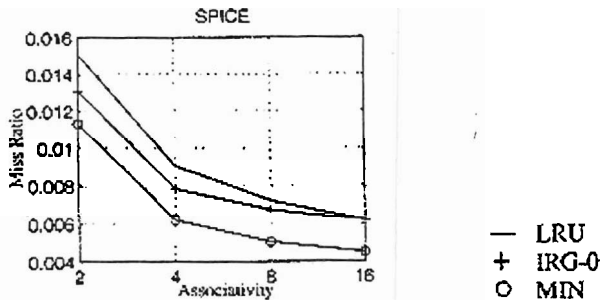


Abb. 12: Fehlerquotenvergleich der Algorithmen LRU, IRG-0 und MIN [PG97]

In der Abbildung 14 wird der IRG-k Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Der IRG-k Algorithmus versucht mittels vergangener Werte das zukünftige Referenzverhalten zu ermitteln.Gerade in Videoverteilungssystemen könnte man mit den Daten der vergangenen Anfragen sehr gut den nächsten Anfragezeitpunkt ermitteln. Auch das Problem, daß es in großen Speichern zuwenig Daten über die einzelnen Seiten gibt, tritt bei VoD-Systemen kaum auf. Jeder Puffer kann nur eine kleine Anzahl von Filmen speichern, über die dann auch eine Vielzahl von Informationen zur Verfügung stehen.

Ein Problem besteht allerdings bei neuen Filmen, die noch keine Daten über ihr Anfrageverhalten zur Verfügung stellen können. Hier könnte man versuchen aus anderen Quellen Daten in das System einzuspeisen. Dabei gibt es verschiedene Möglichkeiten, wie z.B. Daten von Videotheken, Auswertungen der Anzahl von Kinobesuchern oder die Erwartung des Anbieters (entsprechend der Werbeintensität).

Das Prinzip des IRG-Modells ist also ein sehr interessantes Verfahren für Videoverteilungssysteme, um zukünftige Anfragemuster vorherzusagen. Es ist schon deshalb einer genaueren Untersuchung wert, da die benötigten Daten auf jeden Fall gespeichert werden (für Statistiken, Endabrechnungen usw.) und somit kaum Mehraufwand entsteht.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

11. Achtung: MIN stellt hier einen optimalen Algorithmus dar und nicht den LRU-MIN.

3.6.2 WIRG-k

Beschreibung:

Diese Strategie stellt eine Kombination aus dem IRG-k und dem WS Algorithmus dar. Es handelt sich also hierbei um einen lokalen Seitenersetzungsalgorithmus mit variablen Partitionen. Wie beim WS, so gibt es auch hier einen Parameter y (window-size) von dem abhängt, ob man eine Seite behält oder besser entfernt, um den freigewordenen Speicherrahmen anderen Prozessen zur Verfügung zu stellen. Nach jeder Anfrage einer Seite a wird der nächste IRG value von a aufgrund der vergangen IRGs geschätzt. Ist der geschätzte Wert < y behält man die Seite, andernfalls wird sie entfernt.

Dabei gibt es zwei mögliche Problemfälle. Bei einer zu großen Schätzung des IRG values wird die Seite gelöscht, obwohl sie innerhalb der nächsten y Referenzen benötigt wird. Dieser Nachteil läßt sich mit der WIRG-k Strategie leider nicht lösen. Der zweite Problemfall tritt bei einer zu kleinen Abschätzung auf. Hier behält man die Seite, obwohl die Referenzzeit jenseits der nächsten y Referenzen liegt oder die Seite in Zukunft sogar nie mehr aufgerufen wird. Im letzteren Fall bliebe die Seite auf ewig im Puffer, da sie ja nie mehr benötigt wird und somit auch nicht mehr auf ihren nächsten IRG value hin untersucht werden würde. Dieses Problem läßt sich allerdings lösen, indem man noch einmal den IRG predictor für im Speicher befindliche Seiten benutzt, die seit mehr als y Zeiteinheiten nicht mehr aufgerufen worden sind. Ist der vorhergesagte nächste IRG value kleiner als y, dann behält man die Seite. Im anderen Fall wird sie ersetzt. [PG95]

Parameter:

- Window-size y
- IRG-Modell

Implementierungsmöglichkeiten:

- Wahl der window-size Größe y

Ergebnisse aus Studien:

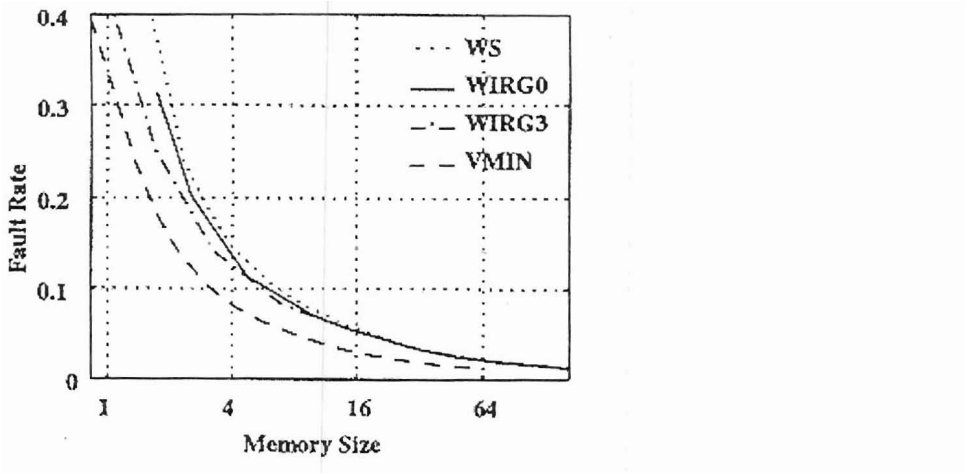


Abb. 13: KFehlerquotenvergleich der Algorithmen WS, WIRG-k und VMIN[1] [PG95]

1. VMIN (Variable-Space Optimum Replacement Algorithm) stellt einen optimalen lokalen Seitenersetzungsalgorithmus dar.

Bewertung:

Auch hier könnte man, wie beim WS Algorithmus eine Einteilung nach Zuschauergruppen oder Filmklassen vornehmen. Die Anwendung der WIRG-k Strategie könnte aber zu einem ständigen ein- und ausladen von Filmen führen, was bei Video-on-Demand-Systemen nicht sinnvoll ist. Entscheidend ist natürlich die Wahl der window-size Größe y. Wählt man diese hinreichend groß, so werden Filme seltener entfernt. Die Gefahr, daß Filme ewig im Puffer verbleiben ist durch den obigen Algorithmus ausgeschlossen. Als gut zu bewerten ist, daß die Entscheidung über die Entfernung eines Videofilmes von den vergangenen Anfragewerten des Filmes abhängt. Die Popularität spielt hier somit eine große Rolle.

Wird ein Film in Blöcken zerlegt übertragen, so kann man mit dieser Strategie festlegen, wieviele Blöcke ein Video in den Puffer laden darf.

3.6.3 HIST

Beschreibung:

Für jede Seite werden für die Abschätzung des nächsten IRG value nur die letzten h Referenzen dieser Seite berücksichtigt Dies läuft darauf hinaus, daß nur die letzten h Zeiten für jede Seite gespeichert werden müssen (ähnlich zu dem LRU-k Algorithmus).

Wie in der Abbildung 14 zu sehen, genügt schon eine kleine Zeitreihe von Werten, um den LRU Algorithmus zu übertreffen. Will man andererseits so gute Werte liefern wie der IRG-0, dann benötigt man sehr lange Zeitreihen. [PG97]

Parameter:

- Faktor h
- IRG-Modell (aufbauend auf die letzten h Referenzen)

Implementierungsmöglichkeiten:

- Wahl des Faktors h

Ergebnisse aus Studien:

In der Abbildung 14 wird der HIST Algorithmus mit anderen Algorithmen verglichen

Bewertung:

Bei dieser Strategie wird nur ein Teil der vergangenen Anfragedaten gespeichert und somit die verfügbaren Kapazitäten entlastet. Da in VoD-Systemen grundsätzlich die Anfragedaten zur Ermittlung der Popularität gespeichert werden sollten und der HIST Algorithmus schlechtere Werte liefert als die IRG-k-Strategie, sollte man direkt den IRG-k Algorithmus anwenden.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

3.6.4 BITS

Beschreibung:

Bei dieser Methode werden nur grobe Werte des IRG gespeichert. Jeder IRG value wird ungefähr auf seine nächste Zweierpotenz geschätzt, d.h. ein IRG value von x wird auf $2^{(\log 2\,(x))}$ angenähert. So braucht man nicht für jeden IRG value einen eigenen Frequenzzähler, sondern es genügt ein einzelnes Bit zu speichern, d.h. man zeichnet nur auf, ob dieser IRG value in der Vergangenheit auftritt oder nicht. [PG97]

Parameter:

- IRG-Modell

Ergebnisse aus Studien:

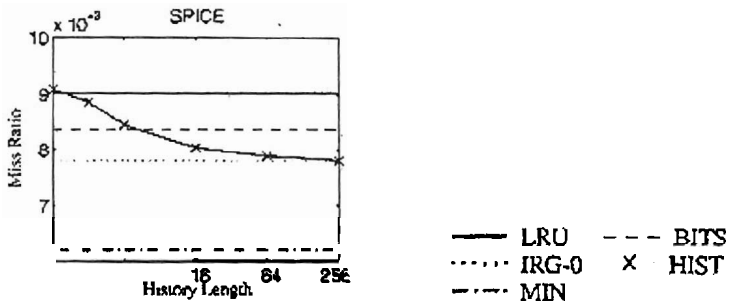


Abb. 14: Vergleich verschiedener Fehlerquoten der Algorithmen LRU, IRG-0, MIN, BITS und HITS [PG97]

Bewertung:

Mit dem BITS Algorithmus wird die zu speichernden Datenmenge noch weiter reduziert. Wie in der Bewertung von HIST schon erwähnt, hat es bei Videoverteilungssystemen keinen Sinn bei der Aufzeichnung der vergangenen Werte zu sparen. Auch hier sollt man lieber gleich den IRG-k Algorithmus anwenden, da er immer noch die besten Werte liefert.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

3.7 Prioritäts-basierende Algorithmen

Um diese Algorithmen anwenden zu können, bedarf es bei der Speicherorganisation einiger Besonderheiten. Zunächst einmal sind die Prozesse nach ihrer Priorität geordnet. Jeder Prozeß enthält einen sogenanntes „transaction set", wobei ein transaction set alle zur Ausführung des jeweiligen Prozesses notwendigen Seiten enthält. Des weiteren unterscheidet man noch in fixed und unfixed pages. Fixed pages sind dabei diejenigen Seiten welche gerade bearbeitet werden. Solche Seiten können nie zur Ersetzung ausgewählt werden. Sobald die Bearbeitung beendet ist werden die bis dahin fixed pages zu unfixed pages. Des weiteren existiert ein timestamp Feld, welches den Zeitpunkt seit der letzten Referenz einer Seite angibt, wobei der Zählerstand der jeweiligen Seite über einen globalen Zähler ermittelt wird. Je größer der Wert im timestamp Feld, desto kürzer ist die Zeit seit der letzten Anfrage. [JCL90]

3.7.1 Priority-Hints Algorithm

Beschreibung:

Wird bei dieser Strategie Speicherplatz benötigt so wählt man die zu ersetzenden Seiten unter den unfixed pages der Prozesse mit niedrigerer Priorität, als der Priorität des anfordernden Prozesses aus. Man beginnt dabei mit dem Prozeß der niedrigsten Priorität und sucht dort nach unfixed pages. Bei der Auswahl der zu ersetzenden Seite unter diesen unfixed pages bedient man sich der MRU-Strategie. Gibt es in dem Prozeß keine unfixed pages oder reichen diese nicht aus, so arbeitet man sich langsam Prozeß für Prozeß nach oben. Beim Erreichen einer Transaktion mit gleicher oder gar größerer Priorität als der des anfordernden Prozesses, werden die neuesten unfixed pages dieses anfordernden Prozesses zur Ersetzung ausgewählt. Das bedeutet, daß keine Transaktion Speicherplatz von einem Prozeß gleicher oder höherer Priorität erhalten kann. In diesem Fall kann man die Strategie eher als lokal, denn als global[12] bezeichnen. Kann keine Seite gelöscht werden, so wird die Anfrage in eine Speicherwarteschlange eingereiht. Außerdem kann bei laufenden Prozessen niedrigerer Prioritäten der Prozeß mit der niedrigsten Priorität unterbrochen werden, um Speicherplatz zu schaffen.

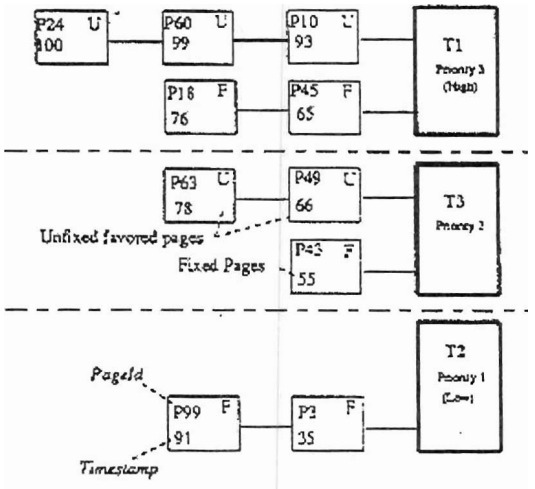


Abb. 15: Beispiel einer Priority-Hints Speicherorganisation [JCL90]

12. Zur Erklärung der Begriffe lokal und global siehe Kapitel 3.23.24.2.

Ein Beispiel unter der Verwendung von Abbildung 15 soll das Verfahren noch illustrieren:

Der Prozeß T1 startet eine Anfrage für die Seite P6, welche sich nicht im Puffer befindet. Bei der Suche nach einer ersetzbaren Seite wird nun mit dem Prozeß T2 begonnen, da dieser die niedrigste Priorität hat. Im obigen Fall sind keine unfixed pages vorhanden so daß man auf den nächsthöheren Prioritätslevel springen muß. In T3 wird nun die Seite P63 zur Tilgung ausgewählt (Anwendung der MRU-Strategie). Wären in den Prioritätsstufen 1 und 2 keine unfixed pages vorhanden, so würde die Wahl auf P24 des Prozesses T1 (also des anfordernden Prozesses) fallen. [JCL90]

Parameter:

- Priorität der jeweiligen Prozesse
- Zugehörigkeit der Seite zu einem Prozeß
- Zeit seit der letzten Anfrage einer Seite i

Implementierungsmöglichkeiten:

- Zuordnung einer Priorität zu jedem Prozeß

Ergebnisse aus Studien:

In der Abbildung 8 wird der Priority-Hints Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Grundsätzlich ist die Verteilung von Prioritäten in Videoverteilungssystemen keine schlecht Idee. Der Anbieter könnte z.B. die zehn Filme, die in der letzten Woche am häufigsten nachgefragt wurden, für die kommende Woche mit einer hohen Priorität belegen, um sie vor einer Ersetzung zu schützen. Man könnte außerdem Filme in verschiedene Klassen einteilen und diesen Klassen unterschiedliche Prioritäten zuordnen. Klassen die nur selten nachgefragt werden (z.B. Heimatfilme, Stummfilme usw.), kann man dann mit einer niedrigen Priorität belegen, um sie bei Speicherbedarf als erstes zu entfernen. Auch die Einteilung in fixed und unfixed pages könnte man so belassen, damit nur die Filme ersetzt werden können, die gerade nicht genutzt werden.

Das Problem liegt hier eher im verwendeten internen Algorithmus der einzelnen Klassen. Es wird hier immer diejenige Seite ersetzt, die als letztes nachgefragt wurde (MRU Algorithmus). Dies ist in VoD-Systemen nicht sinnvoll. Schließlich kann man nicht davon ausgehen, daß gerade frequentierte Filme für eine längere Zeit nicht mehr nachgefragt werden, eher das Gegenteil. Als interne Strategie sollte man also über ein besseres Verfahren nachdenken, wie z.B. LFU oder LRU.

Wird ein Film in Blöcke zerlegt übertragen, so stellt der Film einen Prozeß und seine einzelnen Blöcke die Seiten dar. Da aber die Blöcke nur in einer bestimmten Reihenfolge ein- und ausgeladen werden dürfen, müßte hier als interne Strategie der FIFO Algorithmus angewendet werden. Des weiteren muß jedem einzelnem Film eine andere Priorität zugeordnet werden, um die Blöcke nicht zu vermischen.

3.7.2 Priority-LRU

Beschreibung:

Bei diesem Algorithmus gibt es zwei Faktoren, die bei der Auswahl der zu ersetzenden Seite berücksichtigt werden. Erstens die Wahrscheinlichkeit mit der die Seite bald wieder aufgerufen wird und zweitens die Priorität des zugehörigen Prozesses. Diese beiden Faktoren sind durch das timestamp Feld der im Puffer befindlichen Seiten, beziehungsweise durch die Verwendung von Prioritäts-basierenden

LRU Schlangen für den eingeteilten Zwischenspeicher miteinbezogen. Diese Reihe von potentiell ersetzbaren Seiten wird hier als P_{LRU} bezeichnet. In Abbildung 16 besteht P_{LRU} aus den Seiten P1, P17 und P16.

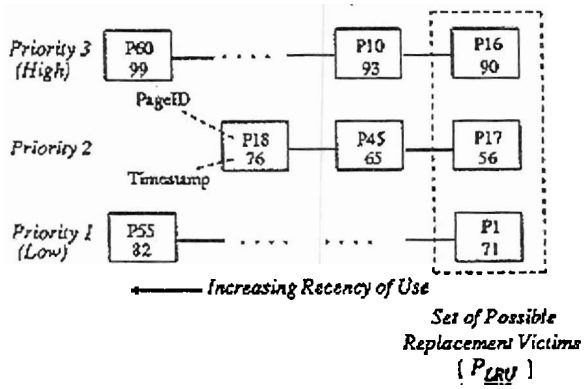


Abb. 16: Beispiel einer Priority-LRU Speicherorganisation [JCL89]

Die Idee dieser Ersetzungsstrategie ist, daß die zuletzt benutzte Seite der niedrigsten Priorität ausgewählt werden soll. Allerdings mit folgendem Vorbehalt: Seiten die sich erst seit einer bestimmten Zeit im Puffer befinden werden durch einen sogenannten W_R[13]-Parameter geschützt. W_R stellt dabei einen Schwellwert dar und verhindert das sofortige Löschen einer gerade gespeicherten Seite. Indem man diesen Parameter variiert, ist es möglich auch die relative Gewichtung von Priorität und Neuheit (vergangene Zeit seit der letzten Anfrage) zu verändern. Man nimmt weiter an, daß m die Anzahl der Seiten im P_{LRU} wiedergibt, d.h. m entspricht der Anzahl der LRU Schlangen im obigen System. Bei der Suche der zu tilgenden Seite fahndet man unter den m Kandidaten nach dem am besten geeigneten. In der niedrigsten Prioritäts-Schlange wird die Suche gestartet, indem man prüft ob der mögliche Kandidat unter den schützende W_R-Parameter fällt. Ist dies der Fall, so springt man auf die nächsthöhere Prioritätsebene und wiederholt den Prozeß. Das Verfahren endet sobald ein „Opfer" gefunden wurde oder keine weitere Suche mehr möglich ist. Fallen alle Mitglieder des P_{LRU} unter den W_R-Schwellwert, so wird standardmäßig einfach unter den möglichen Seiten diejenige Seite mit der niedrigsten Priorität ausgewählt.

Ein Beispiel unter der Verwendung von Abbildung 16 soll diese Methode noch illustrieren:

Der globale timestamp Zähler soll 100 betragen und W_R sei 25. Bei der Suche einer zu ersetzenden Seite wird nun mit der Seite P1, der LRU Seite mit dem niedrigsten Prioritätslevel, gestartet. Solange 100-71>25 wird P1 als Opfer ausgewählt. Ist dagegen W_R = 30 dann fällt P1 unter den Schwellwert. In diesem Fall wird P17 zur Ersetzung ausgewählt. [JCL89]

Parameter:

- Priorität der jeweiligen Prozesse
- Zugehörigkeit der Seite zu einem Prozeß
- Zeit seit der letzten Anfrage einer Seite i
- P_{LRU} = Reihe von potentiell ersetztbaren Seiten
- W_R = Schwellwert der die Mindestverbleibdauer einer Seite seit einer Anfrage angibt
- m = Anzahl der Seiten im P_{LRU}

13. W_R steht für Window of Replacement

Implementierungsmöglichkeiten:

- Zuordnung einer Priorität zu jedem Prozeß
- Wahl des Schwellwertes W_R

Ergebnisse aus Studien:

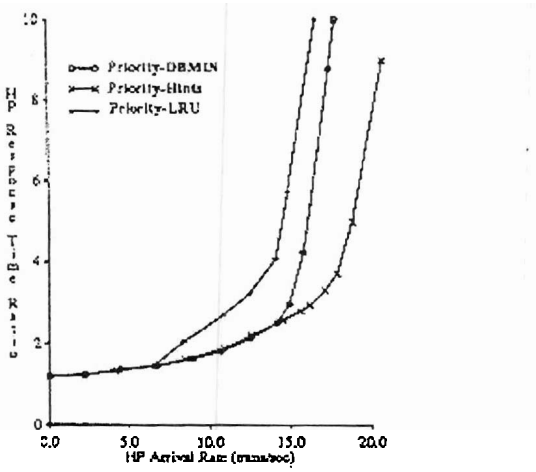


Abb. 17: Antwortzeiten der Algorithmen Priority-Hints, Priority-LRU und Priority-DBMIN[1] [JCL90]

1. Priority-DBMIN gehört zu den sogenannten seitentypbezogenen Seitenersetzungsalgorithmen, die in dieser Arbeit nicht aufgeführt werden, weil sie für VoD-Systeme nixht geeignet sind.

Bewertung:

Der Priority-LRU Algorithmus funktioniert vom Prinzip her ähnlich wie der Priority-Hints Algorithmus. Auch hier wird die Idee der Prioritätenvergabe angewendet, jedoch mit einem anderen internen Algorithmus. Der Kritikpunkt, der bei der Bewertung von Priority-Hints geäußert wird, ist hier gelöst. Statt der ungeeigneten MRU-Strategie wird hier das LRU-Verfahren angewendet. Außerdem gibt es durch den Schwellwert W_R noch einen weiteren Schutzmechanismus, mit dem man eine Mindestverbleibdauer im Puffer einstellen kann. Es handelt sich hier um einen für ein Videoverteilungssystem recht interessanten Algorithmus, dessen Prinzip (Prioritätenvergabe) weitere Untersuchungen lohnenswert macht.

Für Filme, die in Blöcken zerlegt übertragen werden, gilt dasselbe, wie bei der Bewertung der Priority-Hints-Strategie.

3.8 Übergang zu variablen Seitengrößen

Bei den bisher angeführten Strategien wurden immer konstante Seitengrößen angenommen, und damit waren auch die Kosten, die beim Laden einer Seite entstehen, für jede Seite gleich groß. Das die Realität allerdings anders aussieht ist leicht einzusehen. Die Seitengrößen variieren erheblich und damit natürlich auch die Kosten für deren Einlagerung. Dies läßt sich auch sehr schön in der Abbildung 18

erkennen.

Page	*Size*	*Cost*
1	12	31
2	5	20
3	3	8
4	2	24
5	6	4
6	4	2
7	1	2
8	8	29
9	30	50
10	13	29
11	7	24

Abb. 18: Seitengrößen- und Kostenvergleich einzelner Seiten [CO76]

Die beiden folgenden Algorithmen stellen nun eine der ersten Strategien dar, die variable Seitengrößen berücksichtigten. [CO76]

3.8.1 LEC (Least Expected Cost Algorithm)

Beschreibung:

Bei diesem Verfahren spielen die erwartenden Kosten für die Erfüllung der nächsten Anfrage eine große Rolle. Diese erwartenden Kosten ergeben sich aus dem Produkt der Wahrscheinlichkeiten einer Anfrage einer Seite i und den Kosten C_i . Bei den Kosten C_i handelt es sich um Bearbeitungskosten, die bei der Beschaffung einer Seite i entstehen. Die Anzahl der Anfragen einer bestimmten Seite i in einer festgelegten Zeit t drückt man durch die Variable u^t_i aus. Somit ergeben sich die zu erwartenden Kosten mit folgender Formel:

$$\textbf{Wert} = u^t_i * C_i$$

Der LEC Algorithmus ersetzt nun die Seite, deren Produkt von u^t_i und C_i am kleinsten ist. Hat jede Seite die gleichen Kosten, dann entspricht LEC der LFU-Methode. [CO76]]

Parameter:

- u^t_i = Anzahl der Anfragen einer i-ten Seite in einer bestimmten Zeit t
- C_i = erwartete Kosten für eine bestimmte Seite i

Ergebnisse aus Studien:

In der Abbildung 10 wird der LEC Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Durch die Einführung von variablen Seitengrößen und unterschiedlichen Kosten kommt man den Anforderungen von Video-on-Demand-Systemen schon näher. Die Verbindung von der Anzahl der Anfragen an einen Film und dessen verursachenden Kosten stellt einen recht einfachen und gleichzeitig wirksamen Algorithmus dar, wobei die Kosten für die Beschaffung eines Filmes nicht so einfach zu definieren sind. Sie bestehen aus vielen Faktoren und sind teilweise schwierig zu berechnen. Man

denke nur an die Kosten für den benutzten Speicherraum im Puffer, den Transportkosten über gemietete Leitungen (Telekom), den Gebühren für den Film (GEMA, Kirch-Gruppe) usw.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

3.8.2 LECS (Least Expected loss per unit Size Algorithm)

Beschreibung:

Der LECS ist eine Erweiterung des LEC. Zu dem Produkt aus u^t_i und C_i kommt jetzt auch noch die Größe der einzelnen Seite in Form von S_i hinzu. Zum Zeitpunkt t wird dann die Seite i, welche den kleinsten Wert des Produktes

$$\textbf{Wert} = (u^t_i * C_i) / S_i$$

hat ersetzt. LECS reduziert sich zu LFU, wenn die Kosten und Speicherparameter (C_i , S_i) sich mit dem Index i nicht verändern.

Sind die Parameter dagegen variabel, dann hat der Algorithmus den Vorteil, daß nun gleichzeitig die Referenzhäufigkeit der Seiten, die Kosten der Anfrage und die Größe der Seiten berücksichtigt werden. Die Gewichtung dabei ist so, daß in dem Fall zweier Seiten mit identische Kosten und Anfragehäufigkeiten die Seiten ausgewählt wird, die den größten Speicherplatz besetzt hält. [CO76]

Parameter:

- u^t_i = Anzahl der Anfragen einer i-ten Seite in einer bestimmten Zeit t
- C_i = erwartete Kosten für eine bestimmte Seite i
- S_i = Seitengröße einer bestimmten Seite i

Ergebnisse aus Studien:

Die Abbildung 10 zeigt deutlich die Notwenigkeit der Berücksichtigung von Seitengröße und Kosten bei Ersetzungsalgorithmen. Die relativen Kosten der Standardalgorithmen LRU und LFU entfernen sich mit zunehmender Puffergröße vom „idealen" Algorithmus (IDEAL), bis zu dem Zeitpunkt bei dem der

Puffer so groß ist, daß er alle Seiten unterbringen kann. LEC und LECS dagegen nähern sich nach einer anfänglichen Divergenz dem Ideal an. [CO76]

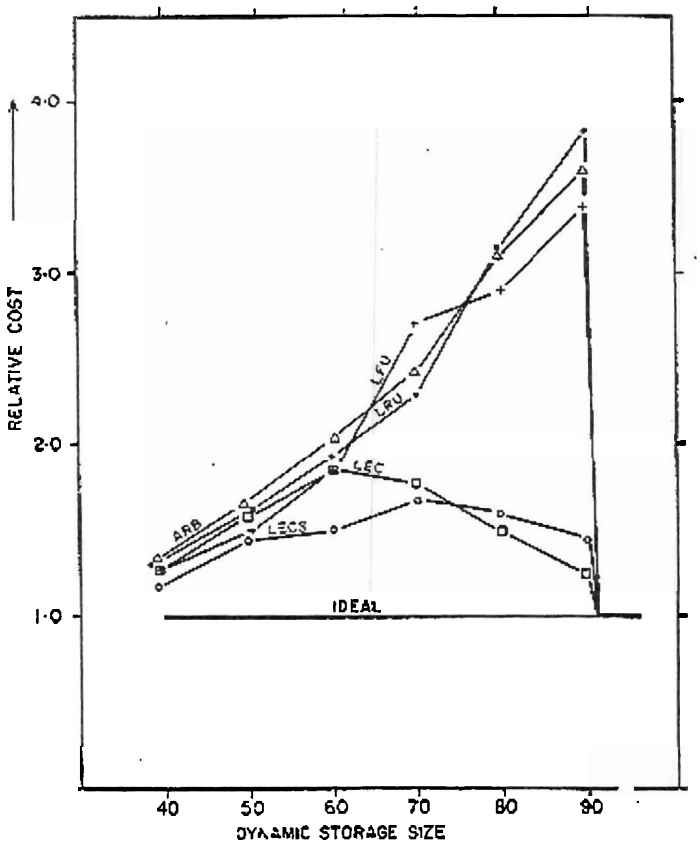


Abb. 19: Relative Kosten/Speichergröße für eine einheitliche Verteilung[1] [CO76]

1. Einheitliche Verteilung bedeutet, daß jede Seite dieselbe Wahrscheinlichkeit einer Anfrage hat.

Bewertung:

Der LECS Algorithmus funktioniert ähnlich zur LEC-Strategie bis zu dem Unterschied, daß nun auch die Seitengröße berücksichtigt wird. Da alle Videofilme sehr viel Speicherplatz benötigen sollte die Größe der einzelnen Filme bei der Ersetzungsentscheidung keine zu große Rolle spielen. Bei dieser Strategie stellt die Seitengröße auch eher eine Sekundärstrategie dar, die vor allem dann von entscheidender Bedeutung ist, wenn zwei Seiten mit identische Kosten und Anfragehäufigkeiten auftreten.

Einzelne Blöcke eines Films nach dieser Strategie zu bewerten, ist wie in Kapitel 3.3 erklärt nicht sinnvoll.

4 Caching zwischen Speicher und Platte

4.1 Dateisysteme

Bei der Verwendung eines virtuellen Speichers kann ein Programm nur eine bestimmte Menge an Informationen innerhalb seines zugewiesenen Adreßraumes speichern. Die Speicherkapazität ist also begrenzt. Für einige Anwendungen ist der Speicherplatz dennoch ausreichend, aber für viele andere ist er zu klein. Die Lösung diese Problems liegt darin, die Informationen auf eine Diskette oder andere externe Speicher abzulegen. Dabei werden die Daten in Einheiten, sogenannten Files gespeichert. Den Teil des Betriebssystems, der die einzelnen Files verwaltet bezeichnet man als Filesystem oder Dateisystem.

Der Zugriff auf eine Platte ist wesentlich langsamer als der zum Speicher. Aufgrund dieser Gegebenheit sind viele Dateisysteme so konzipiert worden, daß die Anzahl der notwendigen Plattenzugriffe reduziert wird. Die gebräuchlichste Technik der Zugriffsreduzierung ist die des Block-Caches oder Puffer-Caches. In diesem Zusammenhang ist ein Cache eine Zusammenfassung von Files, die zur Platte gehören, die aber im Speicher gehalten werden, um die Leistung zu verbessern.

Verschiedene Algorithmen können für die Verwaltung des Caches eingesetzt werden, aber ein gebräuchlicher ist der, der alle Leseaufträge prüft, ob das gewünschte File im Cache ist. Ist er im Cache, kann der Leseauftrag ohne einen Plattenzugriff erfüllt werden. Ist die Datei nicht im Cache, wird er als erstes in den Cache gelesen und dann bei Bedarf kopiert. Nachfolgende Anforderungen für dasselbe File können dann durch den Cache befriedigt werden. Wenn eine Datei in einen vollen Cache geladen werden muß, ist eine andere Datei zu entfernen und auf die Platte zu schreiben, wenn er seit dem letzten Zugriff modifiziert worden ist.

Im Kapitel 5.3 werden nun verschiedene Algorithmen beschrieben, mit denen man entscheiden kann, welche der Files ersetzt werden. Die entscheidende Änderung zu den Seitenersetzungsalgorithmen liegt in der Berücksichtigung der unterschiedlichen Größen der einzelnen Files. Trotzdem können in Dateisystemen auch die früher entwickelten Seitenersetzungsstrategien ohne weiteres verwendet werden. Ein angenehmer Unterschied zwischen Paging und dem Cache besteht darin, daß Zugriffe auf den Cache im Vergleich relativ selten sind, da in den Caches nicht nur einzelne Seiten, sondern vollständige Dateien gespeichert werden. [TAN90][TAN92]

4.2 Möglichkeiten des Einsatzes in Videoverteilungssystemen

Wie schon bei den Seitenersetzungsalgorithmen werden hier zunächst einige allgemeine Unzulänglichkeiten und Annahmen der folgenden Algorithmen aufgeführt, die bei der Anwendung in Videoverteilungssystemen vorliegen.

Die Bewertung für die Eignung in VoD-Systeme erfolgt nun immer unter der Annahme, daß ein File einem Videofilm entspricht. Ein *Film* wird also immer *vollständig* und nicht in Blöcken übertragen.

Leider findet sich auch hier kein Algorithmus der zunächst prüft, ob es sich überhaupt lohnt einen neuen Film in den Cache aufzunehmen, wenn dafür ein anderer entfernt werden muß. Alle *neu aufgerufenen Files werden immer im Cache gespeichert.* Ebenso werden auch hier die Faktoren *Ladezeit und Bandbreite* nicht berücksichtigt.

Das Problem, daß ein modifiziertes File zurück geschrieben werden muß entfällt hier, da an einem Film grundsätzlich keine Änderungen vorgenommen werden können. Das Video wird entweder gelöscht oder an einen nahe liegenden Server übertragen, der diesen Film benötigt. Ein grundsätzliches zurückschicken an den Sender würde zu einer unnötigen Belastung der Übertragungsleitungen führen.

Die hier aufgeführten Nachteile werden bei den einzelnen Bewertungen nicht mehr erwähnt, da man sie jedem der folgenden Algorithmen zurechnen kann.

Replacement Algorithmen, die für Dateisysteme entwickelt wurden sind für Videoverteilungssysteme nun schon interessanter als die älteren Seitenersetzungsstrategien. Von Vorteil ist hier vor allem,

daß der Speicher nicht in feste Rahmen eingeteilt ist, sondern der Speicherplatz genau dem Größenbedarf angepaßt wird. Man verhindert so die Verschwendung von Speicherkapazitäten.

4.3 Strategien

Caching Strategien in Dateisystemen sind sehr verwandt zu Seitenersetzungsstrategien. Deshalb könnten die meisten lokalen Seitenersetzungsstrategien auch unter diesem Kapitel geführt werden. Besonders zu nennen sind dabei die Algorithmen **FIFO**, **LFU**, **LRU-k**, die im Kapitel 4.4 beschrieben werden.

4.3.1 FBR (Frequency Based Replacement Algorithm)

Beschreibung:

Der FBR ist eine Weiterentwicklung des LFU. Auch hier werden die jeweiligen Referenzhäufigkeiten für jede Datei durch einen Zähler ermittelt. Die Abwandlung liegt in der Unterteilung des Speichers in drei Sektionen. Die erste Sektion (new section) ist der am häufigsten frequentierte Teil des Caches und beansprucht F_{new} des Speichers. Das letzte Teilstück (old section) enthält die am wenigsten benutzten Dateien und beansprucht F_{old} des Speichers. Im Mittelteil (middle section) werden jeweils die neu hinzugekommenen Files abgelegt, deren Zähler auf eins gesetzt wird.

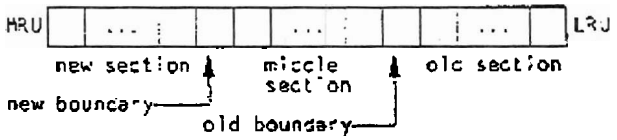


Abb. 20: Die drei sections des Caches [DR90]

In ersten Teil des Speichers sind die Dateien nach dem MRU-Prinzip angeordnet. Um hohe Zählerstände und damit eine Unlöschbarkeit[1] von Files zu verhindern, werden die Zähler der „new section"-Dateien nicht mehr bei jeder Anfrage inkrementiert, sondern konstant gehalten. Allerdings wird die angesprochene Datei ganz nach vorne geschoben (siehe Abbildung 21) und damit der Gefahr einer Zählerdekrementierung ausgesetzt. Bei einem Miss[2] wird nämlich der Zähler der Datei, die an erster Stelle steht auf eins gesetzt und damit in den mittleren Bereich verschoben.

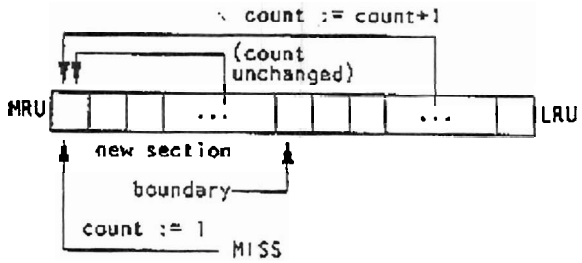


Abb. 21: Verwendung der New Section [DR90]

1. Der Nachteil von LFU wird kompensiert.
2. Anfrage nach einer Datei, welche nicht im Cache gespeichert ist.

Zu ersetzende Einheiten werden nun ausschließlich aus dem Endteil des Speichers herausgesucht. Dabei wird die Datei, deren Zählerstand am kleinsten ist ersetzt. Gibt es mehrere Dateien mit dem gleichen Zählerwert, so wird der LRU Algorithmus angewendet. Das mittlere Teilstück erlaubt den Files erstmal eine gewisse Zählersumme zu bilden, bevor sie ersetzt werden können. Der Grund liegt darin, daß neue Dateien nicht sofort ausgewiesen werden können, da ihre Anfrage erst sehr kurze Zeit zurückliegt. Im Speicher gibt es bestimmt ältere Dateien mit gleichem Zählerstand aber weiter zurückliegenden Anfragen. [DR90]

Parameter:

- Anzahl der Anfragen
- Zeit seit der letzten Anfrage (Sekundäralgorithmus LRU)
- F_{new} = Größe der ersten Sektion
- F_{old} = Größe der dritten Sektion

Implementierungsmöglichkeiten:

- Einteilung des Speichers in drei Sektionen

Ergebnisse aus Studien:

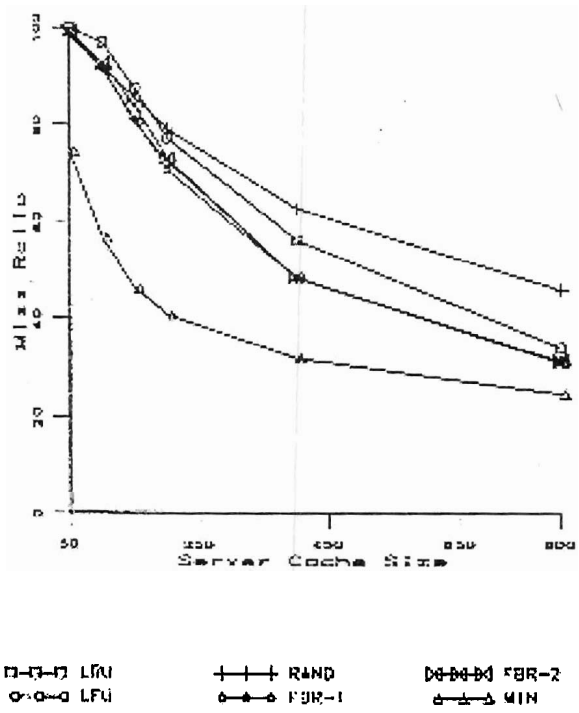


Abb. 22: Fehlerquotenvergleich der Algorithmen LRU, LFU, RAND, FBR und MIN[1] [WEB92]

1. Mit MIN wird hier der LRU-MIN Algorithmus bezeichnet.

Die beiden Algorithmen FBR-1 und FBR-2 unterscheiden sich hinsichtlich ihrer Speichereinteilung wie folgt:

FBR-1: $F_{new} = 25\%, F_{old} = 60\%$
- FBR-2: $F_{new} = 25\%, F_{old} = 40\%$

[WEB92]

Bewertung:

Wie schon beim LFU Algorithmus erwähnt, ist die Häufigkeit der Anfragen bei Video-on-Demand-Systemen ein sehr wichtiges Auswahlkriterium, da die Referenzhäufigkeit der Popularität eines Filmes entspricht. Bei diesem oben beschriebenen FBR Algorithmus ist nun der Nachteil der Unlöschbarkeit von Files durch die Dreiteilung des Speichers kompensiert. Trotzdem reicht es bei Videoverteilungssystemen natürlich nicht aus, die Referenzhäufigkeit eines Films als einziges Kriterium anzuwenden. Ein Algorithmus der sich der "Optimalität" annähert, muß mehr als einen Einflußfaktor berücksichtigen.

4.3.2 LEN (File Length Algorithm)

Beschreibung:

Eine Modifikation der LRU-Methode stellt der LEN Algorithmus dar. Dieser ersetzt die Datei mit dem niedrigsten Wert folgender Formel:

$$\textbf{Wert} = NR_i * S_i$$

NR_i stellt dabei die Anzahl der Referenzen an eine Datei dar, S_i die File-Größe selbst. Im Gegensatz zum LFU berücksichtigt die Gleichung nun auch die Größe der im Cache befindlichen Dateien. Gibt es z.B. eine kleine und eine große Datei mit hoher Referenzhäufigkeit, wobei die kleine Datei eine leicht höhere Anfragehäufigkeit hat, so würde LFU immer die große Datei ersetzen, obwohl die Wiederbeschaffung des großen Objektes vergleichsweise teurer ist. Solche Mängel können mit dem LEN Algorithmus vermieden werden. Allerdings besteht hier, wie auch beim LFU, das Problem, daß große Dateien aufgrund früherer hoher Referenzhäufigkeiten für eine lange Zeit im Cache verbleiben, auch wenn sie nicht mehr benötigt werden. In so einem Fall blockiert eine große Datei den Speicherplatz, der für viele kleinere Dateien ausreichen würde. [MAF93]

Parameter:

NR_i = Anzahl der Anfragen an eine Datei i

- S_i = File-Größe

Ergebnisse aus Studien:

cache size (kb)	FIFO	LRU	LFU	LEN
500	46.7 sec	46.6 sec	46.2 sec	46.4 sec
1000	45.6 sec	45.4 sec	45.0 sec	45.2 sec
1500	44.7 sec	44.5 sec	44.1 sec	44.3 sec
2000	44.1 sec	43.8 sec	43.4 sec	43.6 sec
2500	43.5 sec	43.3 sec	43.0 sec	43.3 sec
3000	43.1 sec	42.8 sec	42.7 sec	42.8 sec
3500	42.7 sec	42.6 sec	42.5 sec	42.6 sec
4000	42.5 sec	42.4 sec	42.4 sec	42.4 sec
4500	42.3 sec	42.3 sec	42.3 sec	42.3 sec
5000	42.3 sec	42.3 sec	42.3 sec	42.3 sec

Tab. 5: Durchschnittliche Anwortzeit der Algorithmen FIFO, LRU, LFU und LEN [MAF93]

cache size (kb)	FIFO	LRU	LFU	LEN
500	29.7%	30.1%	34.0%	32.2%
1000	39.9%	40.9%	46.6%	43.9%
1500	49.4%	51.1%	58.3%	55.0%
2000	57.4%	60.4%	66.5%	62.7%
2500	64.6%	68.4%	71.4%	68.3%
3000	70.3%	73.9%	75.4%	73.2%
3500	75.0%	77.3%	78.2%	76.4%
4000	78.2%	79.6%	79.7%	78.7%
4500	80.2%	80.3%	80.3%	80.3%
5000	80.6%	80.5%	80.5%	80.6%

Tab. 6: Trefferquotenvergleich der Algorithmen FIFO, LRU, LFU und LEN [MAF93]
Tab. 7:

Der LEN Algorithmus bevorzugt große Dateien gegenüber kleinen Files, um Kosten und Zeit zu sparen (siehe Tabelle 5). Allerdings wird dieser Vorteil durch die etwas schlechtere Trefferquote kompensiert (siehe Tabelle 6) , da durch die Bevorzugung großer Files viel Speicherplatz blockiert wird, der gewinnbringender von vielen kleineren Dateien genutzt werden könnte. Der LEN Algorithmus ist der LFU-Strategie deshalb unterlegen. [MAF93]

Bewertung:

Die Größe der Filme spielt bei VoD-Systemen nur eine untergeordnete Rolle, da sie sowieso alle riesig sind. Deshalb würde man bei Nutzung der Referenzhäufigkeit als einziges Auswahlkriterium, lieber gleich den LFU- oder FBR Algorithmus verwenden.

4.3.3 STWS (Space-Time Working Set Algorithm)

Beschreibung:

Der STWS Algorithmus kombiniert die File-Größe mit der Zeit, die seit der letzten Anfrage nach einer Datei vergangen ist. Dabei wird jedem File das Ergebnis folgender Formel zugwiesen:

$$\text{Wert} = S_i * LA$$

Die Files mit den jeweils größten Werten werden aus dem Cache gelöscht. [SMI81] [PD96]

Parameter:

- S_i = File-Größe
- LA = vergangene Zeit seit der letzten Anfrage

Ergebnisse aus Studien:

In der Tabelle 8 und der Abbildung 28 wird der STWS Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Gerade die Größe eines Filmes und dessen letzter Referenzzeitpunkt sind für Videoverteilungssysteme keine sehr aussagekräftige Faktoren. Sie sind nur als Sekundärstrategie hilfreich, wenn es mehr als einen Film mit dem gleichen Primärwert gibt.

4.3.4 STP**y (Space-Time Product Algorithm)

Beschreibung:

Bei dieser Modifizierung von STWS werden nun die Variablen Zeit und Größe unterschiedlich gewichtet. Jedem Dokument wird ein Wert nach folgender Berechnung zugeordnet:

$$\text{Wert} = S_i*(LA**y)^3$$

Dabei wird die vergangene Zeit seit der letzten Anfrage durch den Exponent y erhöht und somit stärker berücksichtigt. Der Parameter y soll einen Wert nahe 1 haben. Hat man keine Kenntnisse für eine geeignete Zahl y, so nimmt man nach einer Untersuchung [SMI81] am besten den Wert 1.4 an. [PD96]

Parameter:

- S_i = File-Größe
- fsLA = vergangene Zeit seit der letzten Anfrage

Implementierungsmöglichkeiten:

- Wahl des Parameters y, möglichst nahe 1 (meist 1.4)

Ergebnisse aus Studien:

In der Abbildung 23 werden die Ergebnisse bei verschiedenen Einstellungen des y-Faktors für den STP**y Algorithmus dargestellt. Der als Vergleich hinzugezogene STWS Algorithmus stellt dabei nichts anderes dar, als ein STP**1,0. Das beste Ergebnis liefert hier der STP**1.4, allerdings unterliegt er trotzdem noch dem Stochopt[4]. [SMI81]

---

3. Der Faktor y stellt einen Exponenten dar.
4. Dieser Stochopt Algorithmus stellt eine nicht-realisierbare stochastische optimale Strategie dar (ähnlich zum OPT im Kapitel 2.3).

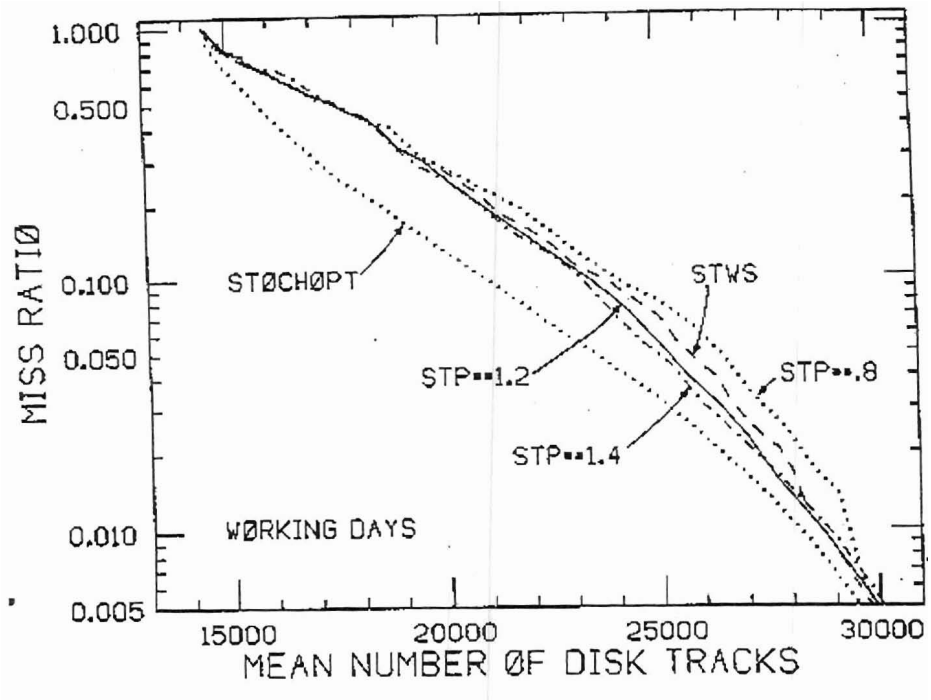


Abb. 23: Fehlerquotenvergleich für den STP**y Algorithmus bei variierendem y-Wert [SMI81]

In der Tabelle 8 und der Abbildung 28 wird der STP**y Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Hier gilt, wie beim STWS Algorithmus, daß die Größe eines Filmes und dessen letzter Referenzzeitpunkt für Videoverteilungssysteme keine sehr aussagekräftige Faktoren sind. Eine unterschiedliche Gewichtung beider Parameter ändert an dieser Tatsache nichts.

5 Caching im World Wide Web

5.1 Motivation für die Verwendung von Cache Systemen im WWW

Der Grund für die Verwendung von Cache Systemen im Internet, ist in dem explosionsartigen Anstieg von Clients und Servers seit 1993 zu sehen. Gerade durch die verstärkte Verwendung des World Wide Webs, dessen Seiten viele Multimedia Elemente (Graphiken, Filme, Musik) enthalten und damit große Datenmengen transferiert werden müssen, werden die bestehenden Leitungen so stark strapaziert, daß lange Wartezeiten für die User entstehen.Um dieses Problem in den Griff zu bekommen gibt es verschiedene Lösungsideen. Zum einen kann man die bestehenden Bandbreiten vergrößern um das wachsende Datenvolumen zu bewältigen. Dies erfordert aber viel Zeit und Geld, da erst neue Leitungen verlegt werden müssen. Eine wesentlich billiger Verbesserung der Internetleistung erhält man durch die Verwendung von Caches.

Dabei kann man drei Arten von Caches unterscheiden:

1. Plattencache beim Client
2. Memorycache beim Server
3. Einsatz von Cache-Servern

Vom Plattencache beim Client profitiert nur ein einzelner User und ist mit dem Hintergrund zu sehen, daß ein User häufig auf die gleichen WWW-Seiten zugreift. Durch den Memorycache beim Server soll der Zugriff auf häufig verwendete Seiten beschleunigt werden, da der Festplattenzugriff entfällt. Cache-Server werden hauptsächlich bei Firmen, Internet Providern oder Universitäten eingesetzt. Ihre Mitarbeiter greifen auf diesen Cache-Server zu und somit können viele User davon profitieren. [DEN96]

Um Verwechselungen zu vermeiden, soll hier noch erwähnt werden, daß die Level-Caches beim Caching im Internet eine leicht andere Bedeutung haben als im Kapitel 2.1 erwähnt. Unter einem Level 1-Cache versteht man hier das Caching auf den Rechnern des Anwenders und ein Level 2-Cache ist auf einem oder mehreren Servern im Netz implementiert. [DOM98]

5.2 Vor- und Nachteile des Caching im WWW

Vorteile:

Die Beantwortung von angefragten Seiten aus einem Cache führt zur Verringerung des Datenverkehrs im Netz und zur Entlastung des Servers, auf dem die Files ursprünglich gespeichert wurden, da nicht mehr alle Anfragen über das Netz an ihn geschickt werden müssen. Darüber hinaus werden die Antwortzeiten verkürzt, wodurch ein Client nicht mehr so lange auf die Beantwortung seiner Anfrage warten muß. Des weiteren werden die Verzögerungszeiten für einen Verbindungsaufbau reduziert. Je mehr User auf gleiche Seiten zugreifen, desto stärker wirken sich diese Vorteile eines Caches aus.

Des weiteren soll noch erwähnt werden, daß die Einführung von Cache-Systemen die Möglichkeit von Monitoring des WWW-Betriebs gewährt. Davon profitieren vor allem die Cache Betreiber wie z.B. Firmen, die nun mitprotokollieren können, welche Seiten von welchem User besucht wurden. Durch das Sperren bestimmter Seiten kann verhindert werden, daß die Mitarbeiter die Arbeitszeit mit Surfen verbringen, anstatt zu arbeiten. [DEN96][DOM98]

Nachteile:

Neben den Vorteilen von Cache-Systemen, gibt es aber auch eine Reihe von Problemen bei der Verwendung von Caches. Diese können aber durch geschickte Strategien so verringert werden, daß sie kaum noch von Bedeutung sind. Das Hauptproblem für den Client besteht in der Gefahr, daß durch den Cache Seiten verschickt werden, die nicht mehr aktuell sind, da die Files auf dem Original-Server verändert worden sind.

Ein anderer Nachteil ist mehr für die Betreiber von Servern von Bedeutung. Die Verwendung von Cache-Servern verringert die für den Serverbetreiber sichtbare Anzahl der Zugriffe auf seine Seiten, da der Cache-Server einmal auf die Seite zugreift und danach alle User die im Cache liegende Seiten verwenden. Das hat vor allem Konsequenzen bei der Festlegung von Preisen für die Werbung in bestimmten Web-Seiten, da ein Betreiber eines Servers keine genaue Auskunft darüber geben kann, wie hoch die „Einschaltquote“ seiner angebotenen Informationen ist. [DEN96][DOM98]

5.3 Allgemeine Einführung in Ersetzungsalgorithmen für das WWW

Um die Leistung von Ersetzungsalgorithmen zu vergleichen verwendet man im allgemeinem zwei Maße. Erstens die Fehlerquote (miss probability) und zweitens die normale Anforderungszeit. Als Fehlerquote wird die Wahrscheinlichkeit definiert, daß sich ein Dokument nicht im Cache befindet. Die normale Anforderungszeit berechnet sich dagegen wie folgt.

$$t = (T_c + p * T_{nc}) / T_{nc}$$

Dabei entspricht p der Fehlerquote, als T_c bezeichnet man die durchschnittliche Zeit für den Zugang in den Cache und T_{nc} als die durchschnittliche Zeit für den Abruf eines Dokumentes, welches sich nicht

im Cache befindet. Nimmt man nun an, daß T_{nc} viel größer als T_c ist und t annähernd gleich zu p, so folgt daraus, daß t kleiner wird wenn p kleiner wird.

Das hat zur Folge, daß man den Cache so groß wie möglich gestalten möchte. Die Cache-Größe ist in der Praxis aber begrenzt. Des weiteren hängt die Fehlerquote von der Größe der im Speicher befindlichen Dokumente ab. Für eine gegebene Speichergröße sinkt die Anzahl der gecachten Dokumente und somit auch die Trefferquote, mit der Zunahme der durchschnittlichen File-Größe. In verschiedenen Untersuchungen wurde schon bewiesen, daß kleinere Dokumente öfter angefordert werden als große. Das führt zu der Erkenntnis, den Ersetzungsalgorithmus nicht nur von der zeitlichen Lokalität, sondern auch von der File-Größe abhängig zu machen.

Ein weiterer Faktor, der zu berücksichtigen ist, stellt der Aufwand zur Neubeschaffung dar. Die Entscheidung ein Dokument aus dem Cache auszuweisen, welches vorher von einem weit entfernten Server heruntergeladen wurde, sollte sehr sorgfältig überlegt werden.

Bei der Auswahl der richtigen Ersetzungsstrategie für einen Cache muß man den Zustand des Caches und das neu hinzukommende Dokument i untersuchen:

1. Soll Dokument i gecacht werden?
2. Ist die Anwort von Frage 1 positiv aber nicht genug freier Speicherplatz im Cache verfügbar, um das Dokument i aufnehmen zu können, muß überlegt werden welches der anderen Dokumente herausgenommen werden soll, um den nötigen Platz zu schaffen.

Der Zustand des Caches wird wie folgt definiert:

1. Die Menge der im Cache gespeicherten Dokumente.
2. Für jedes Dokument gibt es eine Reihe von Zustandsvariablen, die statistische Informationen über das jeweilige Dokument enthalten.

Beispiele für Zustandsvariablen eines Dokumentes i:

- S_i = File-Größe
- LA_i = vergangene Zeit seit der letzten Anfrage an eine Datei i
- D_i = benötigte Zeit zum Laden des Files i vom Web-Server in den Cache
- ttl_i = geschätzte Zeit, bis des Dokument veraltet ist[5]

Nimmt man z.B. den bekannten Algorithmus LRU, so ergibt sich für diesen folgende Gleichung: $W_i = 1 / LA_i$. Bei $W_i = S_i$ wird bei der Wahl des zu ersetzenden Dokumentes ausschließlich aufgrund der File-Größe entschieden. Die generelle Form der Funktion hängt von der Anwendung oder spezieller von den erwarteten Dokumenten- und Anworttypen des Systems ab. Eine mögliche allgemeine Gleichung könnte wie folgt lauten:

$$\mathbf{Wert} = (w_1 D_i + w_2 S_i) / ttl_i + (w_3 + w_4) / LA_i * t$$

Der zweiter Term auf der rechten Seite berücksichtigt die zeitliche Lokalität. Damit meint man, daß identische Anfragen mit großer Wahrscheinlichkeit in einem kleinem Zeitraum nahe beieinander folgen. Der erste Term beachtet die Kosten, die bei der Wiedergewinnung eines Dokumentes entstehen (Wartekosten, Speicherkosten im Cache), während der Faktor $1 / ttl_i$ anzeigt, daß die mit der Wiedergewinnung verbundenen Kosten eines Dokumentes so zunehmen, wie die Lebenszeit des Dokumentes abnimmt.

Da es sehr schwierig ist einen Wert für die Zustandsvariable ttl_i zu erhalten wird er in der folgenden Gleichung nicht mehr berücksichtigt.

$$\mathbf{Wert} = w_1 D_i + w_2 S_i + (w_3 + w_4 S_i) / LA_i$$

5. Veraltet bedeutet, daß der Inhalt des Dokmuentes verändert wurde.

Zur Bestimmung der Konstanten w_i muß man sich genau über das Ziel des Algorithmus im klaren sein. Ein Ziel kann z.B. die Maximierung der Trefferquote, die Minimierung der geschätzten Abrufzeit für einen beliebigen Nutzer oder die Minimierung der Cache-Größe für eine gegebene Trefferquote sein. [BM96]

5.4 Möglichkeiten des Einsatzes in Videoverteilungssystemen

Strategien, die für den Einsatz im Internet entworfen wurden, scheinen für Video-on-Demand-Systemen sehr geeignet. Dies liegt an der Berücksichtigung vieler Faktoren, die auch bei Videoverteilungssystemen von Bedeutung sind. Vor allem Parameter wie Ladezeit oder Bandbreite werden nur in Algorithmen des World Wide Web mit einbezogen.

Die Bewertung für die Eignung in VoD-Systeme erfolgt, wie auch im vorherigem Kapitel, immer unter der Annahme, daß ein Dokument einem Videofilm entspricht. Ein *Film* wird also immer *vollständig* und nicht in Blöcken übertragen.

Leider werden auch bei den meisten der folgenden Algorithmen die neu aufgerufenen Dokumente sofort im Cache gespeichert, ohne zu prüfen, ob dies überhaupt von Nutzen ist. Allerdings gibt es auch einige wenige Strategien, die das Prinzip des Conditional Overwrite[6] anwenden.

Die *Messung der Leistung* einer Strategie erfolgte bisher immer über den Vergleich der Treffer- bzw. Fehlerquoten der einzelnen Algorithmen. Es gibt aber auch noch andere Vergleichsmöglichkeiten.

Zum Beispiel bezog sich die bisherige Trefferquote (*Hit-Rate*) immer auf das vollständige Objekt, unabhängig seiner Größe. Mißt man dagegen die *Byte Hit-Rate*, so kommt man öfters zu anderen Ergebnissen hinsichtlich der Leistung einer Strategie. Hier spielt die Größe des einzelnen Dokumentes eine entscheidende Rolle. Durch die geringen Größenunterschiede der einzelnen Filme, ist diese Art des Leistungsvergleichs für Video-on-Demand-Systeme weniger von Nutzen.

Eine weitere Möglichkeit bietet die Messung der *Download-Zeit*. Gerade in VoD-Systemen spielt die Zeit, die ein Kunde auf den gewünschten Film warten muß eine bedeutende Rolle. Der Anbieter versucht natürlich diese Wartezeit so kurz wie möglich zu gestalten. Er wird also einen Algorithmus bevorzugen, mit dem die Download-Zeit der einzelnen Videos verkürzt wird.

5.5 Strategien

Caching Strategien, die für Virtuelle Speicher oder Dateisysteme kreiert wurden, lassen sich häufig auch im World Wide Web anwenden. Allerdings berücksichtigen diese Algorithmen nicht die Besonderheiten des Internets, wie z.B. Ladezeit oder die Frage, ob ein gecachetes Dokument schon veraltet ist[7]. Auch darf man nicht mehr von konstanten Seitengrößen ausgehen. Im diesem Kapitel wird auf mehrere „ältere“ Algorithmen zurückgegriffen, wie z.B. **LRU-k, CLOCK, GCLOCK**, die im Kapitel 4.4 beschrieben werden.

5.5.1 SIZE

Beschreibung:

Die Auswahl der zu ersetzenden Dokumente erfolgt rein nach der Größe der einzelnen Files, d.h. die größten Dokumente werden als erstes ersetzt. Dies hat den Vorteil, daß große Files die sehr viel Speicherplatz benötigen zugunsten vieler kleinerer Dateien gelöscht werden. Dadurch kann es aber auch passieren, daß kleine Files ewig im Cache verbleiben, obwohl sie nicht mehr benötigt werden. Dies führt dann natürlich zu einer niedrigeren Trefferquote. [NLN97]

Der SIZE Algorithmus wird auch als **SWS** Algorithmus (Space Working Set Algorithm) bezeichnet.

---

6. siehe Kapitel 3.2
7. siehe auch Kapitel 6.1

Da auch der Fall auftreten kann, daß mehrere Files dieselbe Größe haben, gibt es eine kleine Abwandlung der SIZE-Strategie, nämlich der sogenannte **LRU-SIZE** Algorithmus. Bei diesem wird als Sekundärstrategie die LRU-Methode angewendet. [TAT97]

Parameter:

- File-Größe
- bei LRU-SIZE: Vergangene Zeit seit der letzten Anfrage

Ergebnisse aus Studien:

In der Tabelle 8 und den Abbildungen 28, 31, 38, 32, 39, 40, 41 und 42 wird der SIZE Algorithmus mit anderen Algorithmen verglichen.

In der Abbildung 27 wird der LRU-SIZE Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Für den Einsatz in der Videoverteilung ist dieser Algorithmus denkbar ungeeignet, da alle Videofilme sehr viel Speicherplatz benötigen. Somit ist die Auswahl der zu löschenden Dateien anhand des Speicherplatzverbrauchs kein praktikables Auswahlkriterium in Video-on-demand Systemen.

5.5.2 AVI-MRU (Audio and Video-Most Recently Used Algorithm)

Beschreibung:

Beim AVI-MRU Algorithmus werden die ankommenden Dokumente in zwei Gruppen eingeteilt und unterschiedlich behandelt. Die erste Gruppe enthält nur Text- und Bilddokumente, bei denen der einfache LRU Algorithmus angewendet wird. Die zweite Gruppe beinhaltet alle Audio- und Videodateien. Diese Files werden an das Ende der LRU-Kette[8] gehängt (um als nächstes ersetzt zu werden), d.h. es wird der MRU Algorithmus angewendet. Audio- und Videodokumente sind in der Regel sehr groß und blockieren dadurch den Speicherplatz für kleinere Text- oder Bilddateien. Mit dem AVI-MRU Algorithmus werden nun Audio- und Videodokumente sehr schnell wieder aus dem Cache ausgewiesen. [RED95]

Parameter:

- Vergangene Zeit seit der letzten Anfrage
- Dokumententyp

Ergebnisse aus Studien:

In der Abbildung 29 wird der AVI-MRU Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Der AVI-MRU Algorithmus ist bemüht Audio- und Videodokumente als erstes aus dem Speicher zu weisen, um mehr Platz für kleinere Dateien zu schaffen. Dies ist natürlich in Videoverteilungssystemen nicht sinnvoll. Statt die Gruppenbildung vom Dateientyp abhängig zu machen, könnte man allerdings auch die Filme in zwei Klassen einteilen. Problematisch gestaltet sich hier allerdings die doch recht

8. Innerhalb einer LRU-Kette sind die Dokumente nach dem LRU-Prinzip angeordnet.

grobe Einteilung in nur zwei mögliche Gruppen. Das System ist der Prioritätenvergabe[9] nicht unähnlich, allerdings scheinen Prioritäten sinnvoller zu sein, da sie mehr Möglichkeiten offen lassen.

Außerdem wird hier die Ersetzungsentscheidung innerhalb der Kette leider nur von der vergangenen Zeit seit der letzten Anfrage abhängig gemacht. Wichtige Faktoren wie Anzahl der Anfragen, Ladezeit usw. werden hier völlig vernachlässigt.

5.5.3 SIZE-MRU

Beschreibung:

Hier basiert die Ersetzungsentscheidung überwiegend auf der Größe der einzelnen Dokumente. Jedes File, daß größer als ein vorher bestimmter treshhold-Parameter ist, wird mittels der MRU-Strategie verwaltet. Bei Dokumenten, die unter diesem treshhold-Wert liegen wird der LRU Algorithmus angewendet. Alle Files werden wie beim AVI-MRU Algorithmus in dieselbe LRU-Reihe eingereiht, wobei die „großen" Dokumente bei einer Anfrage an das Ende der Kette (um als nächstes ersetzt zu werden) gestellt werden. Auch mit dieser Strategie werden bevorzugt große Dokument als Ersetzungskandidat ausgewählt, um dadurch mehr Speicherplatz für kleinere Dateien zu schaffen. [RED95]

Parameter:

- Vergangene Zeit seit der letzten Anfrage
- File-Größe
- treshhold-Parameter

Implementierungsmöglichkeiten:

- Wahl des treshhold-Parameters

Ergebnisse aus Studien:

In der Abbildung 29 wird der SIZE-MRU Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Wie beim reinen SIZE Algorithmus ist eine Ersetzungsentscheidung rein aufgrund der File-Größe keine geeignete Strategie für VoD-Systeme. Bei relativ geringen Größenunterschiede wie es bei Filmen der Fall ist, stellt die Einführung eines treshhold-Parameters keine wesentliche Verbesserung dar.

5.5.4 LRU-MIN

Beschreibung:

LRU-MIN hat zum Ziel, die Anzahl der zu ersetzenden Files des LRU zu minimieren. Dabei wird eine Liste L der Files, geordnet nach deren Größe erstellt. Die File-Größe wird mit einem ganzzahligen Wert, genannt S_i festgelegt. P soll hier den benötigten Speicherplatz für die neue Datei darstellen. Der Algorithmus geht wie folgt vor:

- Zuerst wird $S_i = P$ gesetzt.

9. Siehe auch Prioritäts-basierende Algorithmen auf Seite 35.

- (*) Danach wird eine Liste von Files erstellt, deren Größe >= S_i ist (L kann auch leer sein). Aus dieser Liste werden nun die Dateien mit dem LRU Algorithmus entfernt bis L leer ist oder der freie Speicherplatz zumindest S_i entspricht.
- Wird noch Speicherplatz benötigt (d.h. P ist noch nicht erreicht) wird S_i auf $S_i/2$ herabgesetzt und das Verfahren ab * wiederholt.

[ASA+95]

Der LRU-MIN Algorithmus wird auch als **MIN** Algorithmus bezeichnet.

Parameter:

- Vergangene Zeit seit der letzten Anfrage (LRU)
- P = benötigter Speicherplatz
- S_i = File-Größe

Ergebnisse aus Studien:

In den Abbildungen 31, 27, 34 und 37 wird der LRU-MIN Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Für den Einsatz in der Videoverteilung ist dieser Algorithmus ungeeignet, da alle Videofilme sehr viel Speicherplatz benötigen. Die Auswahl des zu löschenden Films, anhand des Speicherplatzbedarfs eines neuen Videos ist keine brauchbare Strategie.

5.5.5 LRU-THOLD

Beschreibung:

Ist ein zu cachendes Objekt so groß, daß hierfür viele kleine Dateien aus dem Cache gelöscht werden müßten, wird es nicht gecacht. Stattdessen legt man eine maximale File-Größe einen sogenannten threshold-Parameter fest, ab der ein Objekt nicht mehr gecacht wird, es sei denn es ist genügend freier Platz im Speicher vorhanden. [ASA+95]

Der LRU-THOLD Algorithmus wird auch als **LRU-TH** Algorithmus bezeichnet.

Parameter:

- Vergangene Zeit seit der letzten Anfrage (LRU)
- File-Größe
- Threshold-Parameter

Implementierungsmöglichkeiten:

- Wahl des threshold-Parameters

Ergebnisse aus Studien:

In den Abbildungen 24 und 25 wird der LRU-THOLD Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Für Videoverteilungssysteme ist dieser Algorithmus ungeeignet, da kein Film aufgrund seiner Größe an einer Speicherung im Cache gehindert werden darf. Allerdings ist hier positiv zu bewerten, daß nicht jedes Objekt sofort gespeichert wird, sondern die Möglichkeit besteht, eine Speicherung zu verweigern.

5.5.6 LRU-k-TH

Beschreibung:

Der LRU-k-TH Algorithmus stellt eine anspruchsvoller Version der LRU-THOLD-Strategie dar. Er setzt sich aus dem LRU-k Algorithmus in Verbindung mit einem treshhold-Parameter zusammen.[TRS97][TAT98]

Parameter:

- vergangene Zeit seit den letzten K-Anfragen
- File-Größe
- Threshold-Parameter

Implementierungsmöglichkeiten:

- Wahl des threshold-Parameters
- Wahl des Faktors K

Ergebnisse aus Studien:

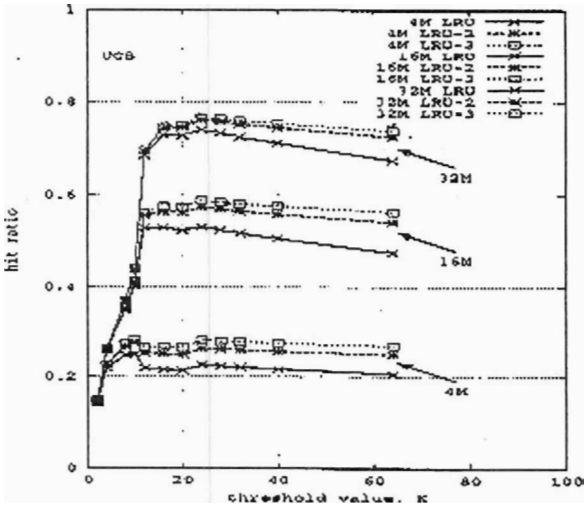


Abb. 24: Trefferquotenvergleich der Algorithmen LRU-TH und LRU-2,3-TH bei unterschiedlichen treshhold- Werten [TRS97]

Man erkennt, daß zwischen dem treshhold-Parameter und der Cacheleistung (Trefferquote) eine Abhängigkeit besteht. Die Trefferquote steigt mit dem treshhold-Parameter bis zu einem bestimmten Punkt, um dann langsam abzusinken.

Der LRU-2-TH Algorithmus weist deutlich eine bessere Leistung auf, als die LRU-TH-Strategie, vor allem für kleine Caches. LRU-3-TH übertrifft LRU-2-TH nur geringfügig. Alle drei Algorithmen weisen eine ähnliche Abhängigkeit ihrer Trefferquote hinsichtlich des treshhold-Parameters auf. Weit-

ere Experimente[10] haben keine signifikante Verbesserungen in der Cacheleistung für k = 4 und k = 5 gebracht. [TRS97]

In der Abbildung 27 wird der LRU-k-TH Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Ebenso wie beim LRU-THOLD ist hier positiv zu bewerten, daß eine Möglichkeit besteht, die Speicherung eines Filmes zu verhindern. Allerdings ist das dafür verwendete Auswahlkriterium (die Größe eines Filmes) denkbar ungeeignet. Dieser Nachteil kann auch nicht durch die bessere Berücksichtigung des Anfragemusters[11] der Nutzer ausgeglichen werden.

5.5.7 Adaptive LRU

Beschreibung:

Der Adaptive LRU Algorithmus arbeitet, wie der LRU-TH mit einem threshold-Parameter. Dieser Parameter gibt an, bis zu welcher Größe ein Dokument noch gespeichert werden darf. Im Gegensatz zum LRU-TH Algorithmus läßt sich der treshhold-Wert den aktuellen Gegebenheiten anpassen und stellt somit keinen fixen Parameter dar. Es darf somit nicht überraschen, daß im Cache befindliche Dokumente größer als der aktuelle treshhold-Wert sein können, da dieser Wert bei der früheren Speicherung des betreffenden Dokumentes eben größer war.

Wenn nun Speicherplatzbedarf besteht, werden zunächst alle Dokument, die größer als der (dynamisch verstellbare) aktuelle treshhold-Parameter sind aus dem Cache ausgewiesen. Danach verwendet man den puren LRU für die restlichen Objekte, bis wieder genügend Kapazitäten zur Verfügung stehen.

Bei der Einstellung des threshold-Wertes geht man wie folgt vor. Es wird mit einem Anfangswert gearbeitet, der periodische zu- oder abnimmt. Wird die Leistung mit der Änderung besser, so wird der Parameter weiter erhöht (verringert). Im anderen Fall ändert man die Taktik und verringert (erhöht) den threshold-Wert. [MAR96]

Parameter:

- Vergangene Zeit seit der letzten Anfrage (LRU)
- File-Größe
- Threshold-Parameter

Implementierungsmöglichkeiten:

- Wahl des Anfangswertes des threshold-Parameters
- Dynamik des treshhold-Parameters

10. Nähere Erläuterungen zu weiteren Experimenten siehe [TRS97]
11. Siehe auch Bewertung der LRU-k-Strategie im Kapitel 4.4.

Ergebnisse aus Studien:

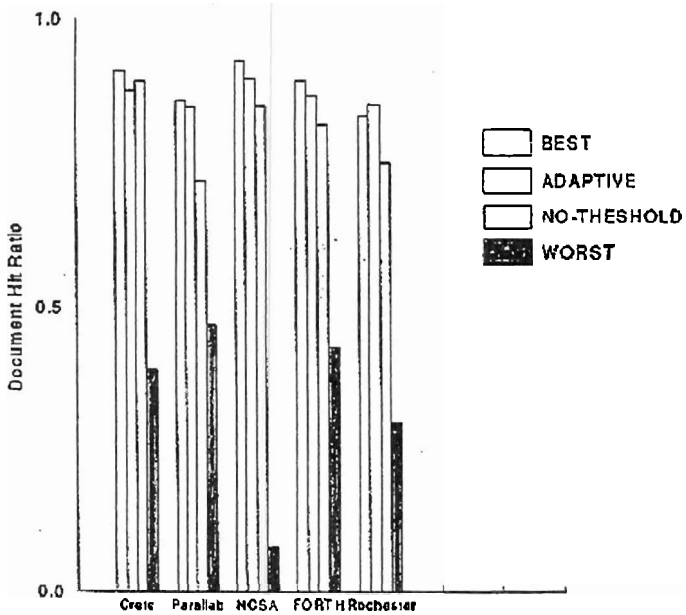


Abb. 25: Trefferquotenvergleich der Algorithmen LRU, LRU-TH und Adaptive LRU [MAR96]

Der Adaptive LRU Algorithmus wird in der Abbildung 25 mit den folgenden Strategien verglichen:

- Best: Die beste Leistung der LRU-TH-Strategie
- Worst: Die schlechteste Leistung der LRU-TH-Strategie
- No-Threshold: Alle Dokumente werden unabhängig ihrer Größe gespeichert

Bei einer der Simulationen (Rochester-Trace) übertraf der Adaptive LRU überraschenderweise den Best-Algorithmus. Dies liegt am etwas ungewöhnlichem Anfragemuster dieser speziellen Verteilung[12]. Die Ermittlung des optimalen treshhold-Wertes der Adaptive LRU-Strategie hängt somit nicht nur von der Cache-Größe und den angefragten Dokumenten ab, sondern auch vom dynamisch wechselndem Anfragemuster. [MAR96]

Bewertung:

Für Videoverteilungssysteme ist dieser Algorithmus nicht geeignet, da die Auswahl des zu ersetzenden Filmes nicht anhand seiner Größe getroffen werden kann. Daran ändert auch ein dynamisch verstellbaren treshhold-Parameter nichts. Allerdings ist hier wiederum positiv zu bewerten, daß nicht jedes Objekt sofort gespeichert wird, sondern die Möglichkeit besteht eine Speicherung zu verweigern.

5.5.8 GD (GreedyDual Algorithm)

Beschreibung:

Der Algorithmus verbindet mit jeder gecachten Datei einen Wert H. Wenn eine Datei am Anfang in den Cache geladen wird, so wird H mit den Kosten[13] (cost) , die bei der Einladung entstehen gleichgesetzt. Im Fall, daß nun Speicherplatz benötigt wird, ersetzt man das Dokument mit dem niedrigsten Wert von H (min_H) und alle anderen Dateien reduzieren ihren jeweiligen H-Wert um min_H. Bei einer Datei, die

12. Nähere Erläuterungen siehe Seite 904 [MAR96].
13. Die Kosten sind dabei immer positiv.

nun wiederum aufgerufen wird, wird der H-Wert wieder auf den ursprünglichen Zustand zurückgesetzt. Dadurch behalten Dateien, die erst kürzlich frequentiert wurden einen größeren Anteil der Originalkosten, als die Files die schon eine lange Zeit nicht mehr benötigt wurden. [CI97]

Parameter:

- cost = Kosten die bei der Einladung entstehen

Bewertung:

Die Ersetzungsentscheidung anhand der verursachenden Kosten festzulegen ist keine schlechte Idee. Auch bei Video-on-Demand-Systemen sollten die Kosten, die ein Film verursacht berücksichtigt werden. Problematisch ist allerdings die Berechnung der Kosten[14]. Der Begriff wird hier sehr allgemein verwendet. Bei einer möglichen Anwendung müssen die Kostenfaktoren genau definiert werden.

Filme, deren Speicherung hohe Kosten verursacht haben, sollten natürlich gegenüber Filmen, deren Einladungskosten geringer sind bevorzugt werden. Durch die Subtraktion von min_H wird auch sichergestellt, daß teuere Filme nicht auf ewig im Cache verbleiben. Der Kostenfaktor darf aber nicht das einzige Entscheidungskriterium darstellen.

5.5.9 Pitkow/Recker Algorithmus

Beschreibung:

Dieses Verfahren stellt eine Kombination von LRU und einem garbage collection Algorithmus dar. Zuerst wird ein Fenster (Window) angelegt, welches angibt in welchem Zeitraum die Anfragen nach einem File berücksichtigt werden. Dabei wird häufig ein Zeitraum von sieben Tagen gewählt. Außerdem bestimmt man einen sogenannten comfort level. Dieser ergibt sich aus der Differenz der gesamten Größe des Caches und dem beobachteten durchschnittlichen Wachstum der Datenbank.

Ist zuwenig Speicherplatz vorhanden, d.h. der comfort level ist zu klein, dann werden zunächst die Files deren Aufruf am längsten zurückliegen gelöscht (LRU-Verfahren). Dabei geht man tageweise vor, d.h. man löscht zuerst die Files, die z.B. seit sieben Tage nicht verwendet wurden, dann die, die seit sechs Tagen nicht mehr aufgerufen wurden usw. Gelangt man nun zu den Dateien die gerade erst am aktuellen Tag frequentiert wurden und es besteht trotzdem noch Speicherbedarf, so werden die größten Files gelöscht. Der Pitkow/Recker Algorithmus bevorzugt also kleine vor kurzem nachgefragt Dokumente vor großen und alten Files.

Die beiden Größen comfort level und window size sind in diesem Algorithmus veränderbare Variablen. Basierend auf der Größe der Dateien und der Anzahl der Anfragen an diese Files, kann der Cache-Inhalt jederzeit an dem aktuellen Zugangsmuster des Nutzers anpaßt werden. [PR94]

Parameter:

- Vergangene Zeit seit der letzten Anfrage
- File-Größe
- window-size
- comfort level

Implementierungsmöglichkeiten:

- Wahl der window-size

---

14. Zum Kostenbegriff siehe auch die Bewertung der LEC-Strategie auf Seite 39.

Ergebnisse aus Studien:

In der Abbildung 27 wird der Pitkow/Recker Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Der Pitkow/Recker Algorithmus berücksichtigt bei seiner Ersetzungsentscheidung nur die File-Größe und den Zeitpunkt der letzten Anfrage. Diese Faktoren spielen bei Videoverteilungssystemen keine große Rolle. Die Idee der periodischen Speicherreinigung könnte man allerdings anwenden. Wie beim Static Caching Algorithmus ließe sich eine Periode von einem halben Tag annehmen. Der Vorteil der Pitkow/Recker- Strategie gegenüber Static Caching ist, daß der Speicherinhalt auch während der laufenden Periode geändert werden kann und nicht nur zum Periodenanfang.

Problematisch ist auch hier die Neuzusammenstellung des Cache-Inhaltes zum Periodenanfang, hinsichtlich des Kosten- und Zeitaufwandes.

5.5.10 Static Caching

Beschreibung:

Die Idee des Static Caching basiert auf der Tatsache, daß sich das Muster einer Dokumentenanfrage für Web Server nur sehr langsam ändert. Dokumente die heute sehr oft frequentiert werden, werden aller Wahrscheinlichkeit nach auch noch morgen oft nachgefragt.

Aus dieser Beobachtung heraus wird zunächst für eine relativ lange Zeitperiode eine feste Anzahl von Web Dokumenten in Cache gespeichert. Dieses Set von gecachten Dateien wird periodisch neu zusammengestellt, um die Leistung des Caches zu maximieren. Ist der Zwischenspeicher erst einmal voll, so werden während der laufenden Zeitperiode keine Dokumente durch andere ersetzt. Die Static Caching Methode hat also nur einen primären Parameter, nämlich die Zeitperiode, die angibt wie oft die gecachten Dateien neu gruppiert werden müssen. Die Parameter für die Entscheidung der Neugruppierung stellen nur sekundäre Variablen dar und sind frei wählbar.

Eine mögliche Strategie wäre es, den Set der gecachten Dokumente einmal am Tag neu zusammenzustellen und zwar auf der Basis der weiter oben beschriebenen VALUE-Methode. Aber auch andere Strategien können hier Verwendung finden. [TAT97] [TRS97]

Beispiele dazu finden sich in den Abbildungen 33 und 34. Dort wird der Static Algorithmus mit der SLRU- bzw. WLFU-Strategie verknüpft.

Parameter:

- Zeitperiode
- Parameter der Sekundärstrategie (z.B. der VALUE-Strategie)

Implementierungsmöglichkeiten:

- Wahl der Zeitperiode

Ergebnisse aus Studien:

Die Entscheidung, welche Dokumente in jeder Periode in den Cache aufgenommen werden, wird bei dieser Simulation durch das Verhältnis Anzahl der Anfragen / File-Größe[15] getroffen. Je höher der Wert, desto eher wird das Objekt im Cache gespeichert.

15. dies entspricht dem Static WLFU auf Seite 69

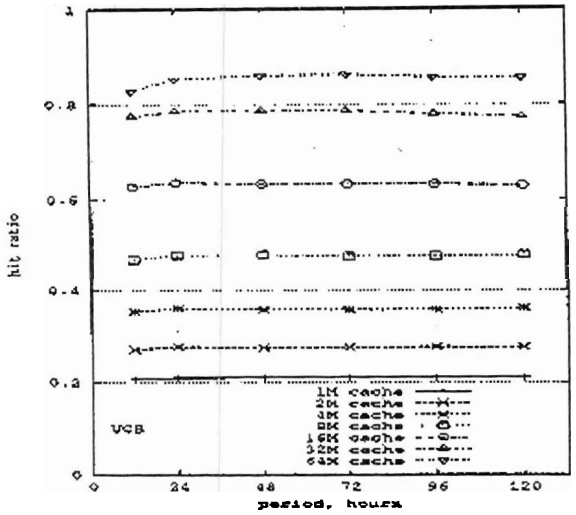


Abb. 26: Trefferquoten der Static Caching-Strategie bei unterschiedlicher Periodendauer [TRS97]

Die Abbildung 11 demonstriert, daß die Leistung der Static Caching-Strategie nur gering von der Periodendauer abhängt. Dies liegt an der sehr langsamen Änderung des Datenverkehrs: Populäre Dokumente verbleiben mindestens fünf Tage im Cache. Dies haben zahlreiche Studien belegt. [TRS97]

In den Abbildungen 33 und 34 wird der Static Caching Algorithmus noch mit anderen Algorithmen

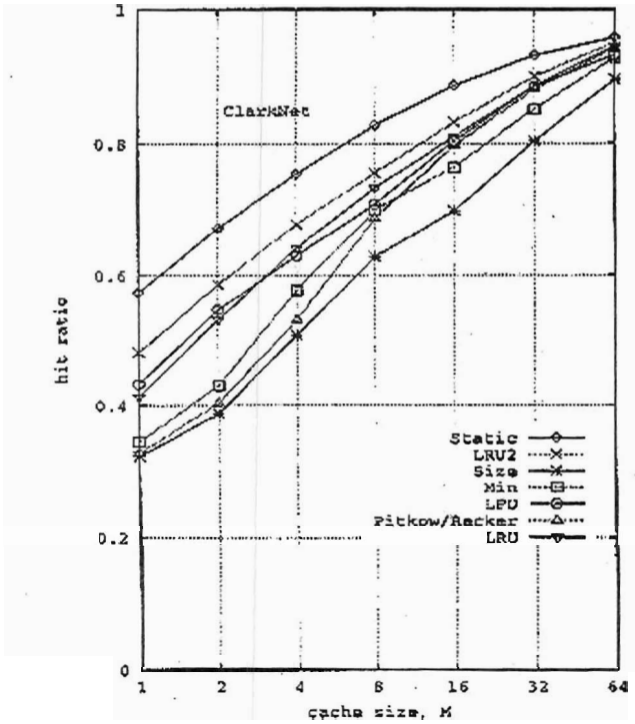


Abb. 27: Trefferquotenvergleich der Algorithmen Static, LRU-2-TH, LRU-SIZE, LRU-MIN[1], LFU, Pitkow/Recker und LRU[TRS97]

1. Die in der Abbildung 27 aufgeführten Algorithmenbezeichnungen sind leicht gekürzt LRU = LRU-SIZE, LRU2 = LRU-2-TH.

verglichen.

Bewertung:

Bei diesem Algorithmus würde der Cache-Inhalt nur in gewissen Zeitabständen verändert. Denkbar wäre zweimal am Tag den Inhalt des Speichers neu zusammenzustellen. Schließlich wird die Nachfrage nach Filmen, je nach Tageszeit unterschiedlich sein. Kinder sehen in der Regel nachts keine Filme, wohingegen spezielle Filme für Erwachsene (Pornos, Horrorfilme usw.) gerade in den Nachtstunden gefragt sind.

Problematisch ist allerdings, daß die Neuzusammenstellung des Cache-Inhaltes eine sehr zeit- und kostenaufwendige Angelegenheit ist. Die Übertragung der Filme dauert recht lange und blockiert natürlich die Leitungskapazitäten. Anders als beim WWW, wo vor allem kleine Dokumente übertragen werden, ist in VoD-Systemen jede Filmübertragung mit einem hohen Zeitaufwand verbunden. Der Übergang sollte deshalb möglichst gleitend vonstatten gehen und zu Zeiten geringer Anfragen vorgenommen werden.

5.5.11 STCWS (Space-Time-Cost Working Set Algorithm)

Beschreibung:

Die beiden Algorithmen STWS und STP**y[16] dienen zur Verbesserung des Cache Hit/Miss-Verhältnisses. Dies garantiert aber nicht den größten Nutzen für den Benutzer. Seit die Zugriffszeiten im Web sehr differieren, kann es von Nutzen sein, Objekte die länger zum Herunterladen brauchen zu bevorzugen. Diese Tatsache führt nun zu einem modifizierten Algorithmus namens STCWS. Bei dieser Methode werden die Files mit dem höchsten Produkt von

$$\textbf{Wert} = S_i * LA / D_i$$

ersetzt.[PD96]

Parameter:

- S_i = File-Größe
- LA = vergangene Zeit seit der letzten Anfrage
- D_i = benötigte Zeit zum Laden des Files i vom Web-Server in den Cache

16. siehe Kapitel 5.3

Ergebnisse aus Studien:

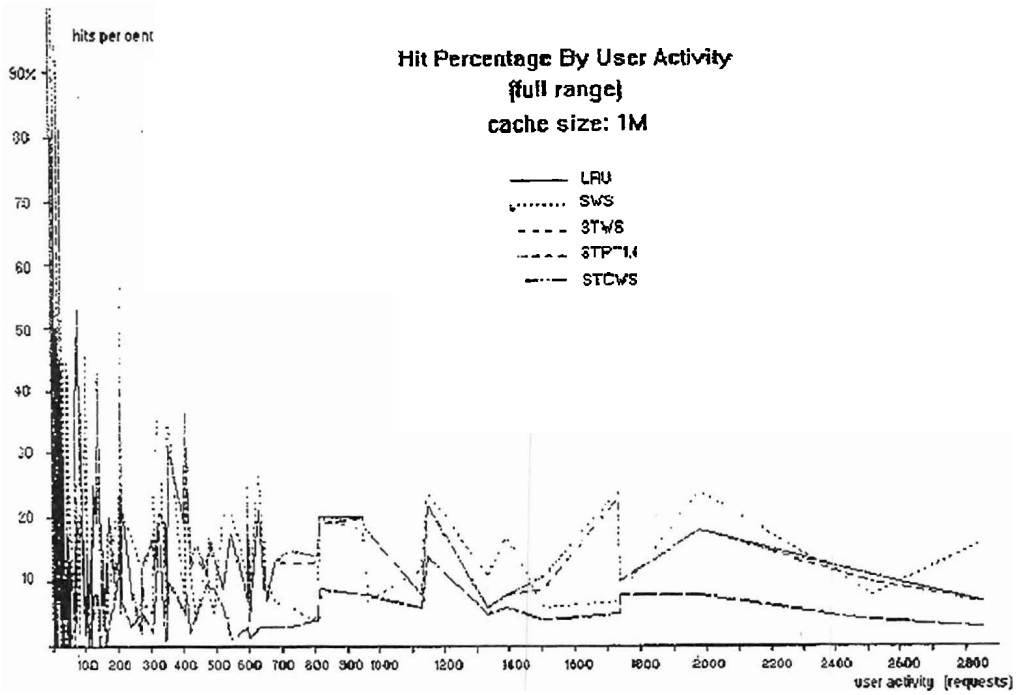


Abb. 28: Trefferquotenvergleich der Algorithmen LRU, SWS[1], STWS, STP**1.4 und STCWS [PD96]

1. Der SWS Algorithmus entspricht dem SIZE Algorithmus.

	LRU	SWS	STWS	STP**1.4	STCWS
100k :	6.7%	5.6%	3.2%	6.6%	3.2%
1M :	14.7%	14.1%	6.2%	14.4%	6.2%
100M :	44.8%	48.5%	49.0%	44.7%	44.6%

unlimited size: 49.3%

Tab. 8: Durchschnittliches Trefferquoten/Antwortzeit-Verhältnis bei variierender Cachegröße [PD96]

Bewertung:

Da das Laden von Videofilmen sehr lange dauert (vor allem wenn er direkt von der Zentrale geladen wird), ist die Ladezeit eines Filmes ein sehr wichtiges Kriterium bei der Auswahl des zu ersetzenden Videos. Die anderen verwendeten Faktoren der STCWS-Strategie jedoch sind bei Videoverteilungssystemen nicht so relevant. Interessant wäre hier eher die Berücksichtigung der Anfragen für ein Video insgesamt.

5.5.12 SPACExAGE

Beschreibung:

Dieser Algorithmus hat nun nicht nur eine LRU-Kette, wie die beiden oben beschriebenen Strategien (AVI-MRU und SIZE-MRU), sondern er beinhaltet mehrere LRU-Reihen. Mittels der File-Größe wird nun bestimmt, welche Dokumente in welche LRU-Ketten eingereiht werden. Dabei kann man z.B. folgendermaßen vorgehen: Zuerst bestimmt man eine Blockgröße von 4KB. Dann werden vier LRU-Reihen gebildet und die einzelnen Dokumente nach bestimmten Regeln auf die Reihen verteilt. Dokument die kleiner als zwei Blöcke sind werden in der ersten Reihe zu finden sein, in der zweiten Kette trifft man Dokumente mit der Größe von 3-8 Blöcken an usw. Jede dieser Reihen wird, wie der Name schon sagt, mittels der LRU-Strategie verwaltet.

Benötigt man nun Speicherplatz, so muß für jedes Dokument am Ende einer LRU-Reihe (in diesem Fall für vier Files) das SPACExAGE-Produkt gebildet werden.

$$\textbf{SPACExAGE} = S_i * LA$$

Die Datei mit dem höchsten Wert wird dann zur Ersetzung ausgewählt.

Es soll hier noch einmal betont werden, daß keine Einteilung des Speichers vorgenommen wird. Der Speicherplatz kann von jedem angefragten Objekt, egal wie groß es ist, frei benutzt werden. [RED95]

Parameter:

- LA = Vergangene Zeit seit der letzten Anfrage
- S_i = File-Größe

Implementierungsmöglichkeiten:

- Einteilung der LRU-Ketten

Ergebnisse aus Studien:

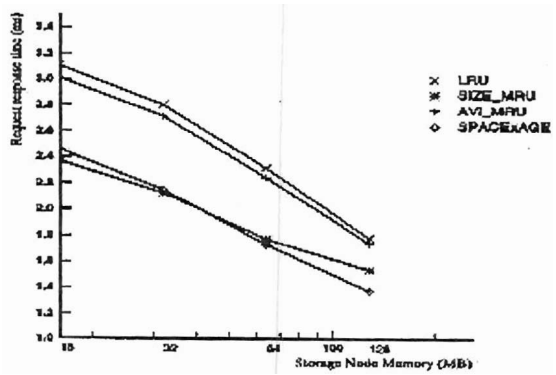


Abb. 29: Vergleich der Antwortzeiten der Algorithmen LRU, SIZE-MRU, AVI-MRU und SPACExAGE [RED95]

In den Abbildung 46 wird der SPACExAGE Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Gerade die beiden Faktoren File-Größe und vergangene Zeit seit der letzten Anfrage sind für Video-on-Demand-Systeme nur untergeordnete (sekundäre) Einflußgrößen. Durch die Bildung von mehreren LRU-Ketten verändert sich der grundsätzliche Gedanke, nämlich durch die Entfernung großer Dokumente Platz für kleinere zu schaffen, nicht. Somit stellt der SPACExAGE Algorithmus keine geeignete Strategie für VoD-Systeme dar.

6.5.13 SLRU (Self-adjusted LRU Algorithm)

Beschreibung:

Der SLRU Algorithmus entfernt das Dokument mit dem niedrigsten Wert folgender Formel:

$$\textbf{Wert} = 1 / (LA_i * S_i)$$

Obwohl der Algorithmus eine hohe Hit-Rate aufweist[17] ist er sehr zeit- und kostenaufwendig, weil bei jedem Miss für alle im Cache befindlichen Dokumente ein neuer Wert berechnet werden muß. [AWY+96][TAT98]

Es soll noch darauf hingewiesen werden, daß der SLRU im Gegensatz zum LRU-SIZE die File-Größe nicht als Sekundärstrategie verwendet, sondern schon in die primäre Strategie integriert.

Parameter:

- S_i = File-Größe
- LA_i = **vergangene** Zeit seit der letzten Anfrage an eine Datei i

Ergebnisse aus Studien:

In den Abbildungen 30, 33 und 34 wird der SLRU Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

In Videoverteilungssystemen ist es nicht sinnvoll die Entscheidung welcher Film ersetzt wird, ausschließlich auf die beiden Faktoren File-Größe und vergangene Zeit seit der letzten Anfrage basieren zu lassen. Die verwendeten Parameter lassen sich besser als Sekundärstrategie verwenden.

5.5.13 PSS (Pyramidal Selection Scheme Algorithm)

Beschreibung:

Aufgrund der Nachteile der SLRU-Strategie wurde ein neuer Algorithmus mit dem Namen PSS entwickelt. Um die zeit- und kostenaufwendige Neuberechnung eines Wertes für alle im Cache befindlichen Files zu vermeiden, werden nun mehrere LRU-Ketten[18] gebildet. Die Einteilung erfolgt hier aufgrund der $\log_2$SIZE-Größe der einzelnen Objekte. Nur diejenigen Dokumente, die sich am Kopf einer LRU-Reihe befinden stellen dabei potentielle Ersetzungskandidaten dar. Von diesen Files wird dann jenes mit dem niedrigsten Wert folgender Formel ersetzt.

$$\textbf{Wert} = 1 / (LA_i * S_i)$$

Die PSS-Strategie weist eine große Ähnlichkeit mit dem weiter oben beschriebenen SPACExAGE Algorithmus auf. Die Verteilung der Dokumente auf die einzelnen LRU-Ketten erfolgt statt mit der „reinen" File-Größe hier mit einem abgewandeltem $\log_2$SIZE, was im Prinzip zu keiner Änderung führt.[AWY+96][TAT98]

---

17. siehe Abbildung 30
18. Innerhalb dieser LRU-Ketten sind die Dokumente nach dem LRU-Prinzip angeordnet.

Parameter:

- S_i = File-Größe
- LA_i = vergangene Zeit seit der letzten Anfrage an eine Datei i

Implementierungsmöglichkeiten:

- Einteilung der LRU-Ketten

Ergebnisse aus Studien:

Anhand der Abbildung 30 erkennt man, daß die beiden Algorithmen PSS und SLRU hinsichtlich ihrer Hit-Rate identisch sind. Der PSS ist durch seinen größeren „Rechenbedarf" nur zeit- und kostenintensiver. [AWY+96]

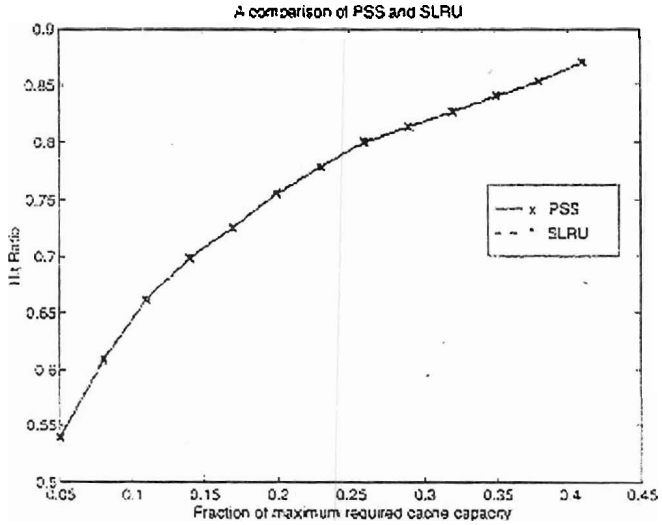


Abb. 30: Trefferquotenvergleich der Algorithmen PSS und SLRU [AWY+96]

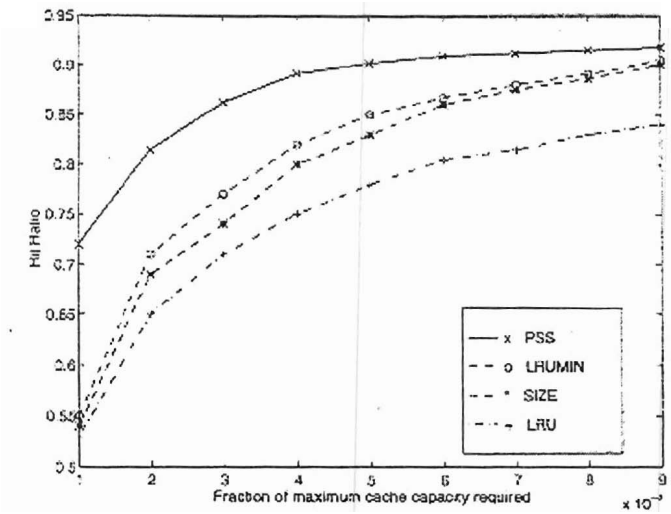


Abb. 31: Trefferquotenvergleich der Algorithmen PSS, LRU-MIN, SIZE und LRU [AWY+96]

In den Abbildungen 33 und 34 wird der PSS Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Wie schon in der Bewertung der SLRU-Strategie, so kann man auch hier wieder sagen, daß die beiden Parameter File-Größe und vergangene Zeit seit der letzten Anfrage für VoD-Systeme nur untergeordnete (sekundäre) Einflußfaktoren darstellen. Durch die Bildung von mehreren LRU-Ketten ändert sich daran nichts, nur der Rechenaufwand wird um einiges reduziert.

5.5.14 GD-SIZE (GreedyDual-Size Algorithm)

Beschreibung:

Der GD-SIZE stellt eine Erweiterung der einfachen GD-Strategie dar. Um auch die unterschiedlichen Größen der Dokumente zu berücksichtigen, ergibt sich hier der Wert H als das Verhältnis von Kosten zu File-Größe (cost / S_i). Die Kosten sind hier wie auch beim GD die Kosten, die bei der Einladung entstehen. Allerdings hängt die genaue Definition der Kosten von der Zielsetzung der Ersetzungsstrategie ab. Werden die Kosten gleich eins gesetzt, ist das Ziel die Trefferquote zu maximieren. Setzt man die Kosten dagegen mit der Download-Verzögerung gleich, so hat man die Absicht die durchschnittliche Verzögerung zu minimieren. Eine andere Möglichkeit wäre es die Kosten gleich den Netzwerkkosten zu setzen, um so die gesamten Kosten so niedrig wie möglich zu halten.

Im Fall, daß nun Speicherplatz benötigt wird, ersetzt man, wie beim GD Algorithmus, das Dokument mit dem niedrigsten Wert von H (min_H) und alle anderen Dateien reduzieren ihren jeweiligen H Wert um min_H. [NLN97][CI97]

Parameter:

- cost = Kosten die bei der Einladung entstehen
- S_i = File-Größe

Ergebnisse aus Studien:

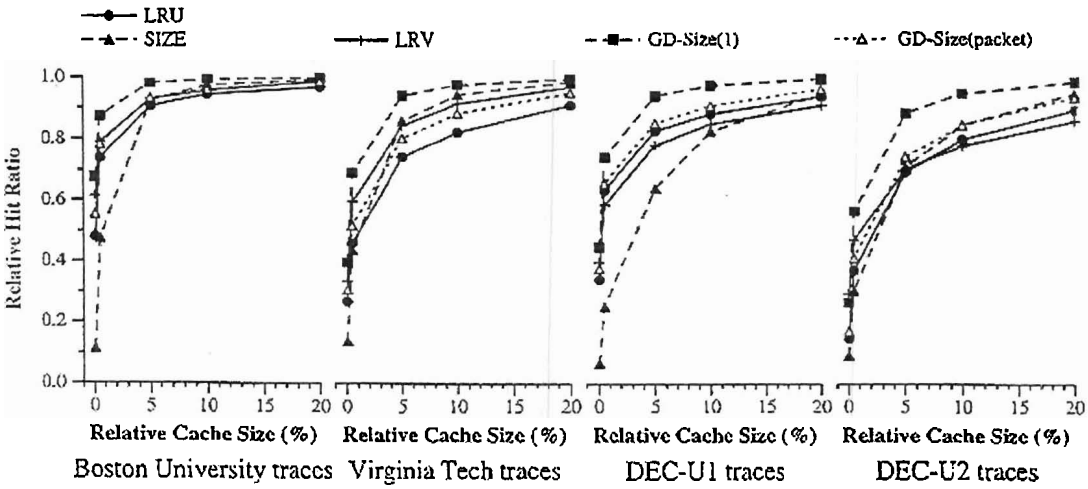


Abb. 32: Relativer Trefferquotenvergleich der Algorithmen LRU, SIZE, LRV, GD-SIZE(1) und GD-SIZE(packet)[CI97]

GD-SIZE(1) und GD-SIZE(packets) stellen zwei Modifikationen der GreedyDual-Size-Strategie dar. Bei der ersten Variante werden die Kosten für jedes Dokument auf eins gesetzt, während bei der zweiten Variante die Kosten gleich 2 + SIZE / 536 [19] (d.h. die geschätzte Anzahl von Netzwerkpaketen die gesendet und empfangen werden, wenn ein Miss auftritt) gesetzt. Somit führt der GD-SIZE(1) zu einer Maximierung der Trefferquote und der GD-SIZE(packets) versucht den Datenverkehr zu minimieren. [CI97]

In den Abbildungen 41 und 42 wird der GD-SIZE Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Zusätzlich zu den verursachenden Kosten wird nun auch noch die File-Größe berücksichtigt. Da Filme alle sehr große Dokument darstellen und in ihrer Größe nur schwach variieren, stellt der GD-SIZE Algorithmus keine wesentliche Verbesserung zum GD Algorithmus dar.

5.5.15 WLFU (Weighted LFU Algorithm)

Beschreibung:

Das Verhältnis von Nutzen zu Kosten ist ein guter Indikator, um den Wert eines Objektes zu bestimmen. Überträgt man dieses Prinzip auf das Internet, so kann man den Speicherbedarf eines Dokumentes vereinfacht mit den verursachten Kosten dieses Files gleichsetzen. Der Nutzen des Caching hängt von der seiner Leistung ab. Die Leistung wird hier als der Wert der Trefferquote definiert. Somit entspricht der Nutzen eines Dokumentes der Anzahl der Anfragen an dieses Objekt. Dies läßt sich sehr leicht einsehen, da die Hit-Rate um so größer ist, je mehr begehrte (häufig angefragte) Files im Cache gespeichert sind. Mit diesem Angaben ergibt sich folgender Wert:

$$\text{Wert} = NR_i / S_i$$

19. 1 Paket für die Anfrage, 1 Paket für die Antwort und SIZE/536 für zusätzlich Datenpakete, bei einer 536-Byte TCP Segmentgröße

Die zu ersetzenden Files werden nun anhand dieses Werte ausgesucht, d.h. Files mit dem kleinsten Wert werden als erstes ersetzt, da sie im Verhältnis zu den anderen Objekten ein schlechteres Nutzen-Kosten-Verhältnis haben. Diese Strategie hat leider einige Nachteile. Zum einen bezieht sie nicht den Zeitpunkt seit der letzten Anfrage in die Berechnung des Wertes eines Objektes mit ein. Was zur Folge hat, daß ein Dokument welches z.B. 1000 mal an einem Tag frequentiert wurde danach aber lange Zeit überhaupt nicht mehr, trotzdem mehrere Tage im Cache verbleibt. Ein anderes Problem tritt auf, wenn sich ein neues File anschickt populär zu werden. Bevor dieses neu hinzugekommene Dokument vor einer Ersetzung geschützt ist, dauert es einige Zeit bis es genügend Anfragen „gesammelt" hat. Bei einem voll besetzten Cache besteht also die Gefahr, daß eine neue Datei sofort wieder gelöscht wird, obwohl eine hohe Anfrage für diese Datei erwartet wird. Diese beiden Probleme führen zu einem Rückgang der Leistung. [TAT97][TAT98]

Der WLFU Algorithmus wird auch als **LFU-SIZE** Algorithmus bezeichnet.

Parameter:

- NR_i = Anzahl der Anfragen an ein Dokument i
- S_i = File-Größe

Ergebnisse aus Studien:

Die Bezeichnung Static in Verbindung mit einem anderen Algorithmus, soll hier die Kombination aus Static Caching[20] als primäre Strategie und einem zweiten Algorithmus als Sekundärstrategie darstellen.

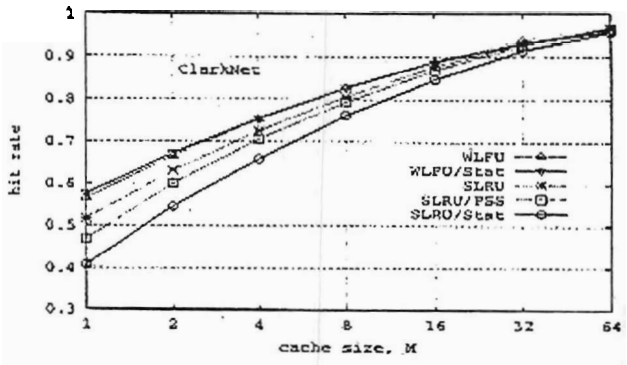


Abb. 33: Trefferquotenvergleich der Algorithmen SLRU, SLRU/PSS, Static SLRU, WLFU und Static WLFU [TAT98]

Der **Static SLRU** Algorithmus liefert hier die schlechtesten Ergebnisse. Dies liegt an der Eigenschaft der SLRU-Strategie nur den Zeitraum der letzten Anfrage an ein Dokument zu berücksichtigen. Das Anfragemuster, welches sich über einen längeren Zeitraum bildet, wird nicht mit einbezogen. Dadurch werden bei der Neugestaltung des Caches immer diejenigen Files hereingenommen, die gerade erst vor kurzer Zeit frequentiert wurden. Viele dieser Dokumente werden nur einmal angefragt und sollten deshalb eigentlich nicht in den Cache gespeichert werden, da sie erst im nächsten Zyklus wieder herausgenommen werden können. Im Gegensatz dazu steht der **Static WLFU**, der hauptsächlich die Anzahl der Anfragen an eine Datei als Entscheidungskriterium verwendet. Bei der periodischen Umbesetzung des Caches werden damit die am häufigsten frequentierten Dokumente in

20. zu Static Caching siehe Seite 69

den Cache übernommen. Dies führt, wie in Abbildung 33 zu sehen, zu recht guten Ergebnissen. [TAT98]

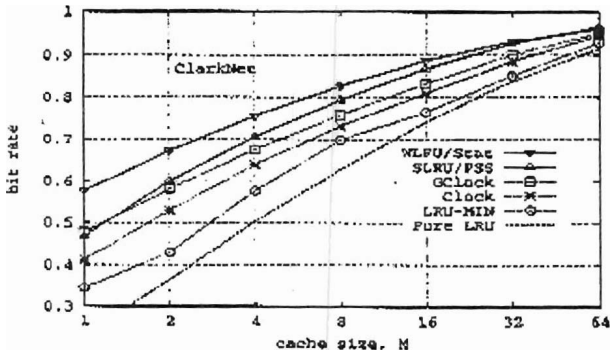


Abb. 34: Trefferquotenvergleich der Algorithmen CLOCK, GCLOCK, LRU-MIN, SLRU/PSS und Static WLFU [TAT98]

Die Algorithmen CLOCK und GCLOCK wurden zwar für Virtuelle Speicher entwickelt, werden aber im Originaltext[21] erstmals mit World Wide Web Algorithmen verglichen. Dabei stellt der GCLOCK sogar eine recht gute Strategie dar, vorausgesetzt der Cache ist nicht zu klein. [TAT98]

Bewertung:

Die Anzahl der Referenzen für einen bestimmten Film sind zwar für dessen Popularität sehr aussagekräftig, genügen aber nicht, um ein entsprechendes „Ersetzungsopfer" auszuwählen. Auch die Kombination mit der Größe schafft hier keine Verbesserung der Strategie. Der WLFU Algorithmus ist somit für den Einsatz in Videoverteilungssysteme nicht sinnvoll.

5.5.16 LNC-R (Least Normalized Cost Replacement Algorithm)

Beschreibung:

Der LNC-R Algorithmus wählt für den Ersatz das unrentabelste Dokument aus. Der Wert jedes Dokumentes wird wie folgt ausgerechnet:

$$\text{Wert} = (L_i * d_i) / S_i$$

Die Files mit dem jeweils kleinsten Werten werden aus dem Cache gelöscht. Hinter diesem Verfahren steckt die Idee, daß es rentabler ist ein File zu behalten, wenn es sehr viel Zeit kostet, dieses File später wieder in den Cache zu laden.

Zur Berechnung von L_i benötigt man folgende Formel:

$$L_i = K / (t - t_k)$$

Die Aufnahme der aktuellen Zeit in die Schätzung von L_i ist sehr wichtig, da sie ein „Altern" der Files gewährleistet und somit kein File auf ewig im Cache verbleiben kann. Außerdem werden bei diesem Algorithmus nur die letzten K Referenzen berücksichtigt, um zu verhindern, daß weit zurückliegende Anfragen zu stark berücksichtigt werden.

---

21. siehe [TAT98]

Immer wenn weniger als K Anforderungszeiten für ein File verfügbar sind (weil das Dokument gerade erst eingeladen wurde), berechnet man L_i unter der Verwendung der maximalen Anzahl von verfügbaren Anfragen. Der LNC-R Algorithmus vergleicht zuerst Files die nur eine Anforderungsanfrage haben in ihrer Rentabilität, dann alle Dokumente mit zwei Anforderungsanfragen usw. bis genügend Speicherplatz frei geworden ist.

Bei dieser Methode könnte es jetzt zu dem Problem des „Verhungerns" (starvation) von Files kommen, die weniger als K Anforderungsanfragen haben. Dieses Problem läßt sich wie folgt darstellen: Die Anfragen werden gelöscht, wenn die entsprechenden Dokumente aus dem Speicher ausgewiesen werden. Danach müssen die Anfragen erneut angesammelt werden, sobald das Files wieder in den Cache geladen wird. Weil das Objekt aber beim nächsten Platzmangel wiederum zur Ersetzung ausgewählt werden wird, gibt es keine Chance die nötigen K Anfragen anzusammeln. Somit wird das File ständig in den Speicher hinein- und wieder herausgeladen. Um diese „Verhungern" zu verhindern, behält der LNC-R die Anzahl der Anforderungsstichproben auch bei Ausweisung eines Dokumentes bei und ordnet sie bei erneuter Einspeicherung dem entsprechenden Objekt wieder zu. [SSV96] [SSV97]

Parameter:

- L_i = Durchschnittliche Anforderungsrate für File i
- d_i = Ladezeit
- S_i = File-Größe
- t = aktuelle Zeit
- t_k = Zeitpunkt der K-ten Anfrage
- K = Anzahl der Anforderungsanfragen

Implementierungsmöglichkeiten:

- Wahl des Faktors K

Ergebnisse aus Studien:

In der Abbildung 35 wird der LNC-R Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Für Video-on-Demand-Systemen stellt der LNC-R Algorithmus eine sehr interessante Strategie dar. Schließlich wird hier nicht nur die Ladezeit berücksichtigt, sondern auch die Anzahl der Anfragen. Wobei nur die letzten K Anfragen in die Wertberechnung eingehen und somit die aktuelle Popularität eines Filmes viel besser ermittelt werden kann. Außerdem kann man Filmen, die aus verschiedenen Gründen vorübergehend aus dem Cache ausgewiesen wurden, wieder ihre „alten" Anfragen zuweisen, um eine erneute Ersetzung zu verhindern.

5.5.17 LNC-A (Least Normalized Cost Admission Algorithm)

Beschreibung:

Das Hauptziel des LNC-A ist es zu verhindern, daß solche Dateien gecacht werden, die die Leistung des Systems verschlechtern. Dies wäre dann der Fall wenn Dokumente gespeichert werden, die unrentabler sind, als die schon im Cache gespeicherten Dateien. Eine neue Datei wird also nur im Cache gespeichert, wenn folgendes gilt:

Wert (neue Datei) > Wert (alte Datei)

Der Wert der alten Datei wird mit der Wertformel vom LNC-R Algorithmus berechnet, nämlich mit

$$\textbf{Wert} = (L_i * d_i) / S_i$$

wobei sich L_i folgendermaßen ermitteln läßt.

$$L_i = K / (t - t_k)$$

Doch wie kann man den Wert eines neuen Objektes berechnen? Die Größe der Datei und die Ladezeit stellen dabei keine Probleme, diese sind ja bekannt bzw. müssen sowieso geschätzt werden. Dagegen stellt die durchschnittliche Anforderungsrate schon ein größeres Problem dar. Wenn die neue Datei schon früher einmal in diesem Cache gespeichert war, so wird L_i mittels der alten Werte berechnet. Sind dabei weniger als K Werte vorhanden, so muß man sich mit diesen begnügen. Doch was macht man mit Dateien, die für den Cache völlig neu sind? Hier kann man leider nicht auf vergangene Werte zurückgreifen. In solch einem Fall basiert die Entscheidung nur auf den beiden bekannten Faktoren, d.h. es wird mit folgender Formel gerechnet.

$$\textbf{e-Wert (neue Datei)} = d_i / S_i$$ [22]

Ein neues Dokument wird nur dann im Cache gespeichert, wenn folgendes gilt:

e-Wert (neue Datei) > e-Wert (alte Datei)

Bei der Berechnung des e-Wertes der alten Datei werden dabei auch nur zwei der drei Faktoren berücksichtigt.[SSV96]

$$\textbf{e-Wert (neue Datei)} = d_i / S_i$$

Parameter:

- L_i = Durchschnittliche Anforderungsrate für File i
- d_i = Ladezeit
- S_i = File-Größe
- t = aktuelle Zeit
- t_k = Zeitpunkt der K-ten Anfrage
- K = Anzahl der Anforderungsanfragen

Implementierungsmöglichkeiten:

- Wahl des Faktors K

Bewertung:

Der LNC-A stellt einen der wenigen Algorithmen dar, die sich nach dem Prinzip des Conditional Overwrite[23] verhalten, d.h. neue Filme werden nur im Cache gespeichert, wenn sie populärer als die schon im Cache befindlichen Filme sind. Dieses Verfahren sollte auf jeden Fall in Videoverteilungssysteme angewendet werden. Da dieser Algorithmus aber nur das Conditional Overwrite-Prinzip verwendet und keine Ersetzungsentscheidungen trifft, kann er in seiner reinen Form nicht alleine eingesetzt werden.

---

22. e steht für estimated
23. siehe Kapitel 3.2

5.5.18 LNC-RA

Beschreibung:

Die Verbindung der LNC-R-Strategie mit dem LNC-A Algorithmus führt zu einer neuen Caching-Strategie, nämlich dem LNC-RA. Hier wird zuerst geprüft, ob die neue Datei überhaupt wertvoll genug ist, um sie im Cache zu speichern. Dies geschieht durch die Verwendung des LNC-A. Fällt die Entscheidung für das neue Dokument und benötigt man für diese neue Datei Platz im Cache, so wird der LNC-R Algorithmus angewendet. [SSV96]

Parameter:

- L_i = Durchschnittliche Anforderungsrate für File i
- d_i = Ladezeit
- S_i = File-Größe
- t = aktuelle Zeit
- t_k = Zeitpunkt der K-ten Anfrage
- K = Anzahl der Anforderungsanfragen

Implementierungsmöglichkeiten:

- Wahl des Faktors K

Ergebnisse aus Studien:

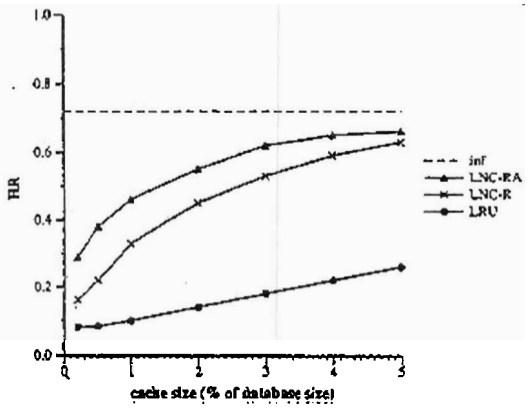


Abb. 35: Trefferquotenvergleich der Algorithmen LNC-RA, LNC-R und LRU [SSV96]

Die Linie „inf" soll hier die maximal erreichbare Trefferquote (für einen unendlichen Speicher) darstellen. Die Abbildung 35 zeigt, daß der LNC-RA sehr nahe an dieses Ideal herankommt. Für K wurde hier der Wert 4 gewählt. [SSV96]

Bewertung:

Durch die Kombination von LNC-R und LNC-A entsteht ein für Videoverteilungssysteme überaus geeigneter Algorithmus, der auf jeden Fall in Simulationen getestet werden sollte.

5.5.19 LNC-R-W3 (Web-Based Cache Replacement Algorithm)

Beschreibung:

Viele verschiedene Studien von Anforderungsmustern im World Wide Web haben gezeigt, daß Web-Nutzer eine starke Präferenz zur Anforderung kleiner Dokumente aufweisen. Die durchschnittliche Anforderungsrate eines Dokumentes D_i mit der Größe S_i läßt sich mit folgender Formel ausrechnen:

$$L_i = c / \text{power}(S_i, b)$$ [24]

Der Parameter b stellt die Abhängigkeit der Referenzrate von der File-Größe dar. Je höher der Wert von b, desto größer ist die Bevorzugung von kleineren Dokumenten seitens der Nutzer. In der Abbildung 36 werden die Simulationsergebnisse für unterschiedliche Cachegrößen aufgezeigt. Man erkennt, daß je nach Größe des Caches der optimale Wert für b variiert. Dieser optimale Wert wird in der Abbildung durch einen weißen Balken dargestellt.

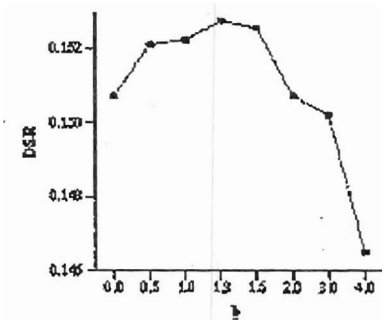


Abb. 36: Delay-savings ratio bei unterschiedlichen Werten für b und bei verschiedenen Cachegrößen [SSV97]

Die Leistung von b wird hier mittels seines delay-savings ratio (DSR) gemessen. Mit DSR drückt man die „Zeitersparnis" aus, die durch die Befriedigung einer Anfrage aus dem eigenen Cache, gegenüber der Erfüllung durch einen weit entfernten Server entsteht.

Konsequenterweise kann man die durchschnittliche Anforderungsrate nicht nur auf vergangene Anforderungsmuster basierend berechnen (wie bei LNC-R), sondern man muß auch die Größe der jeweiligen Files berücksichtigen. Das führt nun zu folgender Gleichung:

$$L_i = K / ((t - t_k) * \text{power}(S_i, b))$$

Mit dieser Gleichung und der „Wertformel" des LNC-R erhält man:

$$\text{Wert} = (K * d_i) / ((t - t_k) * \text{power}(S_i, b+1))$$

Die Files mit dem jeweils kleinsten Werten werden dann aus dem Cache gelöscht. [SSV97]

Parameter:

- L_i = Durchschnittliche Anforderungsrate für File i
- d_i = Ladezeit
- S_i = File-Größe

24. Power bedeutet hier, daß S_i und b Exponenten sind; c stellt eine Konstante dar.

- t = aktuelle Zeit
- t_k = Zeit der K-ten Anfrage
- K = Anzahl der Anforderungsanfragen
- b = Konstante
- c = Konstante

Implementierungsmöglichkeiten:

- Wahl des Faktors K
- Wahl der Konstanten b und c

Ergebnisse aus Studien:

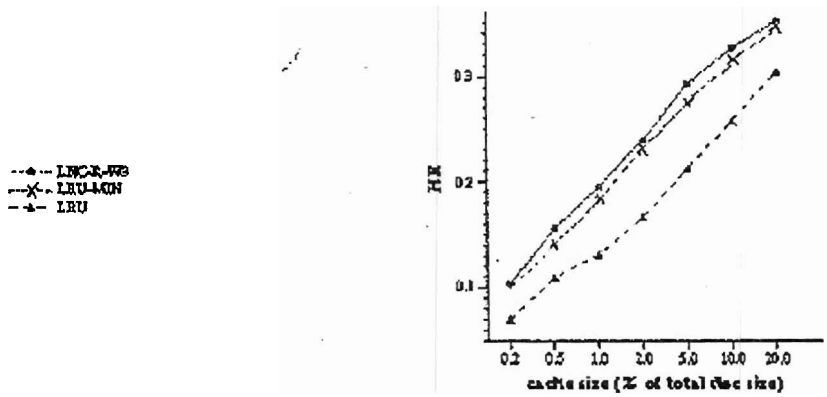


Abb. 37: Trefferquotenvergleich der Algorithmen LNC-R-W3, LRU-MIN und LRU [SSV97]

Für LNC-R-W3 wurden folgende Werte verwendet: b = 1,3 und K = 3

Bewertung:

Die Hauptveränderung der LNC-R-W3-Strategie im Vergleich zum LNC-R Algorithmus liegt in der Berücksichtigung der Tatsache, daß die einzelnen Nutzer kleine Dokumente bevorzugen. Dieser Sachverhalt wird durch die Verwendung eines Parameters b und der besonderen Beachtung der File-Größe berücksichtigt. In VoD-Systemen allerdings verliert die im World Wide Web gemachte Beobachtung (Bevorzugung kleiner Files) ihre Bedeutung. Ein Nutzer wird seinen Film nie aufgrund der File-Größe auswählen, sondern immer aufgrund anderer Gegebenheiten (Geschmack, Empfehlung, Preis usw.).

5.5.20 LRV (Lowest Relative Value Algorithm)

Beschreibung:

Der LRV Algorithmus hält das Dokument mit dem geringsten relativen Wert für den geeignetsten Ersetzungskandidaten. Dieser relative Wert wird durch die Variable P_r ausgedrückt. Der Wert eines Dokumentes wird natürlich durch viele Faktoren beeinflußt, die aber nicht alle berücksichtigt werden können. Um die Schätzung des relativen Wertes einer Datei zu vereinfachen werden Faktoren, die den Wert nur schwach beeinflussen vernachlässigt, d.h. es werden nur die Parameter berücksichtigt, die einen signifikanten Einfluß auf P_r haben. Die „Erfinder" der LRV-Strategie haben durch verschiedene Untersuchen die folgenden Parameter als die wichtigsten Einflußfaktoren[25] bestimmt:

25. Nähere Erläuterungen siehe Kapitel 3 [RV97].

- Vergangene Zeit seit der letzten Anfrage LA: Dieser Faktor hat einen sehr großen Einfluß auf den Wert eines Dokumentes. Hier besteht jedoch das Problem, daß bei Dokumenten die erst einmal angefragt wurden (neue Dokumente) über den Zeitraum der letzten Anfrage keine Angaben gemacht werden können. Dies ist um so schwerwiegender, weil gerade bei „neuen" Dateien die Wahrscheinlichkeit, daß diese Files nie mehr gebraucht werden um so größer ist. Zur Abhilfe wird als Sekundärstrategie die File-Größe verwendet (siehe auch Punkt drei). Der Einfluß dieses Parameters auf P_r wird als 1 - D(LA)[26] ausgedrückt.
- Anzahl der Anfragen NR : Mit diesem Faktor wird die erhöhte Wahrscheinlichkeit einer erneuten Anfrage ausgedrückt. Je gefragter eine Datei ist, desto wertvoller ist sie natürlich. Somit stellt dieser Parameter einen sehr einflußreichen Faktor dar.
- File-Größe S : Um unter Dokumenten mit nur einer Anfrage einen geeigneten Ersetzungskandidaten auszuwählen, scheint die Größe eines Files der geeignetste Faktor zu sein. S wird hier also nur als Sekundärstrategie verwendet, wenn die beiden obigen Parameter nicht angewendet werden können. Der S Parameter tendiert natürlich dazu, große Dokumente aus dem Speicher zu entfernen. Dadurch wird der Platz für mehrer andere Dateien geschaffen, deren erneute Anfrage wahrscheinlicher ist.

Der Wert P_r (NR, LA, S) wird nun wie folgt berechnet:

$$\mathbf{P_r\,(NR, LA, S)} = \mathbf{P_{NR=1}(NR,S)(1\text{-}D(LA))} \quad \text{wenn NR} = 1$$

$$\mathbf{P_r\,(NR, LA, S)} = \mathbf{P(NR)(1\text{-}D(LA))} \quad \text{für alle anderen Fälle}$$

Es soll noch einmal betont werden, daß für den Fall von NR > 1 der Einfluß von S auf P_r (NR, LA, S) vernachlässigt wird. [RV97][CI97]

Parameter:

- LA_i = vergangene Zeit seit der letzten Anfrage an eine Datei i
- NR_i = Anzahl der Anfragen an eine Datei i
- S_i = File-Größe

26. Nähere Erläuterungen siehe Anhang B [RV97].

Ergebnisse aus Studien:

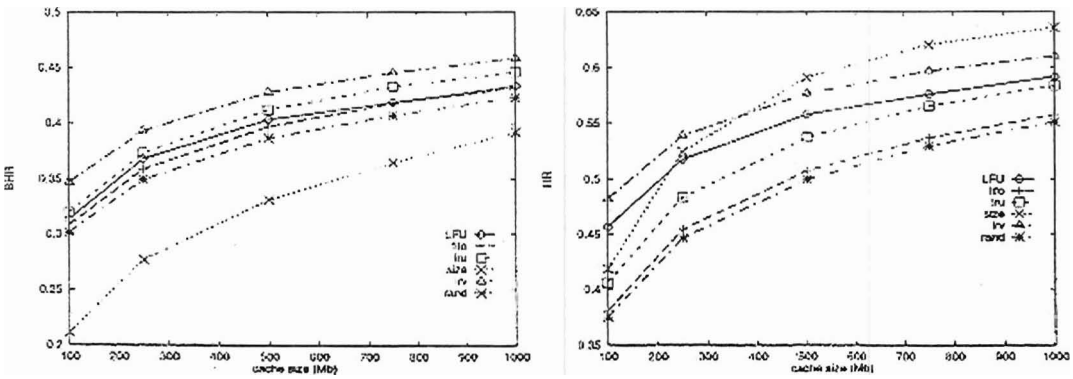


Abb. 38: Byte- und normaler Trefferquotenvergleich der Algorithmen LFU, FIFO, LFU, SIZE, LRV und RAND [RV97]

Die Abbildung 38 stellt die Werte von HR und BHR[27] für verschiedene Algorithmen und Cache-Größen dar. Dabei läßt sich erkennen, daß der LRV Algorithmus eine höhere Byte-Trefferquote einnimmt als alle anderen Algorithmen. Das Gleiche läßt sich anfänglich auch in der anderen Kurve erkennen, bis durch die Erhöhung der Cache-Größe der SIZE Algorithmus die LRV-Strategie übertrifft. Reduziert man allerdings den Umfang des Caches, so gehört der SIZE zu den schlechtesten Strategien, da der Cache durch kleinere Dokumente, die nicht ersetzt werden können blockiert wird. [RV97]

In der Abbildungen 32 wird der LRV Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Der Wert eines Filmes hängt natürlich sehr stark von seiner Popularität ab, die durch die Anzahl der Anfragen ausgedrückt wird. Allerdings darf dies nicht der einzige Einflußfaktor auf den Wert sein. Es müssen auch noch solche Faktoren wie Ladezeit oder Bandbreitenbedarf berücksichtigt werden. Die Idee, die Größe eines Filmes als Sekundärstrategie zu verwenden erscheint recht sinnvoll. Kann man sich nicht entscheiden welches unter den potentiellen Videos (alle haben den gleichen Primärwert) ersetzt werden soll, so wird man den Film wählen, der den meisten Speicherplatz blockiert.

5.5.21 LAT (Latency Estimation Algorithm)

Beschreibung:

Dieser Algorithmus schätzt die benötigte Download-Zeit eines Files, vom Web-Server in den Cache und ersetzt die Files mit der geringsten Download-Zeit. Würde man diese Zeit für jedes Objekt speichern, so würde dies viel zu viel Speicherplatz kosten. Deshalb ermittelt man nur die Zeit für jeden Server der kontaktiert wird. Dies hat außerdem den Vorteil, daß nun Prognosen der Download-Zeit für noch nie geladene Objekte des gleichen Servers gemacht werden können.

Die Download-Zeit setzt sich aus der Verbindungsaufbauzeit zu einem Server, sowie aus der Übertragungszeit des Dokumentes zusammen. Dabei ergibt sich die Übertragungszeit aus der Summe von Verbindungszeit und File-Größe dividiert durch die Bandbreite (Bytes pro Zeiteinheit). Somit erhält man eine Schätzung der Verbindungszeit und Bandbreite von Web-Servern, die in der Vergangenheit kontaktiert wurden.

---

27. Byte Hit-Rate

Für jeden Server j speichert der Cache die geschätzte Verzögerungszeit für die Verbindung zu einem Server ($clat_j$) und die geschätzte Bandbreite der Verbindung (in Bytes/Sekunde), genannt cbw_j. Immer wenn eine neues Dokument vom einem Server empfangen wird, wird gleichzeitig auch die Verbindungsaufbauzeit und die Bandbreite für speziell dieses File gemessen. Diese Werte drückt man durch folgende Variablen aus: s_{clat} und s_{cbw} . Die vorherigen Berechnungen der jeweiligen Werte des Servers werden nun mit folgenden Gleichungen aktualisiert:

$$\mathbf{clat_j = (1\text{-}ALPHA)clat_j + ALPHA\ s_{clat}}$$

$$\mathbf{cbw_j = (1\text{-}ALPHA)cbw_j + ALPHA\ s_{cbw}}$$

ALPHA stellt hier eine glättende Konstante dar mit dem Wert 1/8. Bezeichnet man den Server des Files i mit ser(i) und S_i als die File-Größe, läßt sich die kleinste Download-Zeit mit nachfolgenden Formel berechnen:

$$\mathbf{d_i = clat_{ser(i)} + S_i / cbw_{ser(i)}}$$

Das Dokument mit dem kleinsten d_i wird für den Ersetzungsvorgang ausgewählt. [WA97][WOO96]

Parameter:

- $clat_j$ = Zeit für den Verbindungsaufbau zum Server j
- cbw_j = Bandbreite zum Server j
- s_{clat} = Zeit für den Verbindungsaufbau eines Objektes i vom Server j
- s_{cbw} = Bandbreite eines Objektes i zum Server j
- d_i = kleinste Download-Zeit
- ser(i) = Server des Dokumentes i
- S_i = File-Größe
- ALPHA = glättende Konstante

Implementierungsmöglichkeiten:

- Einstellung der Konstanten ALPHA

Ergebnisse aus Studien:

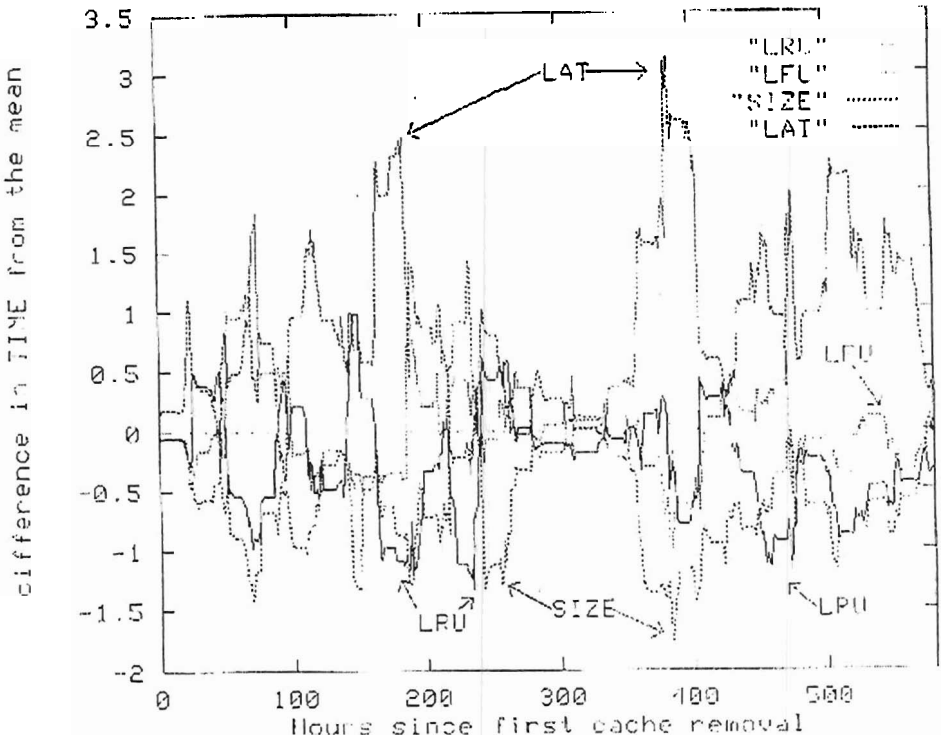


Abb. 39: Veränderung der Download-Zeit für die Algorithmen LRU, LFU, SIZE und LAT [WA97]

Time bedeutet hier die geschätzte Zeit, die der Nutzer auf das Herunterladen seines Dokumentes warten muß (Download-Zeit). Die Mittellinie (mean) stellt die durchschnittliche Downloadzeit dar.

Der Time-Wert für den LAT Algorithmus liegt immer über der mittleren Linie und ist somit schlechter als der Durchschnitt. SIZE dagegen hat immer die kleinste Download-Zeit und wird nur gelegentlich vom LRU Algorithmus übertroffen. [WA97]

Bewertung:

Die benötigte download-Zeit eines Filmes ist beim Einsatz von VoD-Systemen eine sehr wichtige Größe. Der Anbieter möchte jedem Kunden den gewünschten Film, so schnell als möglich zur Verfügung stellen. Deshalb wird man beliebte Filme, die schnell heruntergeladen werden können eher vorübergehend löschen, als Videos deren download-Zeit sehr hoch ist. Die unterschiedlichen download-Zeiten hängen hier allerdings nicht so sehr von der Größe der einzelnen Filme ab, sondern von der Entfernung zum Nutzer. Zwischen dem Kunden und der Zentralbibliothek sind mehrere Server zwischengeschaltet, in deren Cache der gewünschte Film eventuell gespeichert ist. Ist dies der Fall, so ist das Laden von einem Server schneller als von der Zentrale. Diese Tatsache wird in der Variable $clat_j$ berücksichtigt. Als gut ist hier auch die Berücksichtigung der Bandbreite zu bewerten. Allerdings ist diese in der Praxis sehr schwer zu berechnen.

Leider wird eine andere wichtige Einflußgröße beim LAT Algorithmus überhaupt nicht berücksichtigt, nämlich die Anzahl der Anfragen für einen bestimmten Film. Dies wird jedoch im nächsten Algorithmus nachgeholt.

5.5.22 Wooster/Abrams Algorithmus[28]

Beschreibung:

Der Wooster/Abrams Algorithmus stellt eine Erweiterung der LAT-Strategie dar. Hier spielt nun nicht nur die download-Zeit, die Bandbreite und die File-Größe, sondern auch die Anzahl der Anfragen für ein Objekt i eine Rolle. Als zu ersetzendes Dokument i wird jenes mit dem kleinsten Wert folgender Formel ausgewählt:

$$\textbf{Wert} = ((clat_{ser(i)} + W_B / cbw_{ser(i)}) (NR_i ** W_N)) / S_i$$ [29]

W_B und W_N sind Konstanten, welche die Variablen $cbw_{ser(i)}$ und NR_i nach ihrer Bedeutsamkeit gewichten. Ein Dokument wird nicht ersetzt, wenn der obige Ausdruck groß ist. Dies tritt bei einer langen Verbindungszeit zu einem Server (großes $clat_{ser(i)}$), niedriger Bandbreite (kleines $cbw_{ser(i)}$), häufigen Anfragen eines Objektes in der Vergangenheit (großes NR_i), sowie einer kleinen File-Größe (kleines S_i) ein. Die Konstanten haben dabei folgenden Einfluß: Wird W_N nahe Null gewählt, so wird die Dokumentengröße S_i stärker gewichtet als die Anzahl der Anfragen (NR_i). Wächst dagegen W_B, so nimmt $clat_{ser(i)}$ gegenüber $cbw_{ser(i)}$ an Bedeutung zu. [WA97][WOO96]

Server	*clat*	*cbw*	*nref*	*size*	*hyb*	Result
A.any file	10s	1KB/s	x	y	$\frac{(10-10)\cdot x^{B}}{y}$	Retained
B.any file	1s	1KB/s	x	y	$\frac{(1-10)\cdot x^{B}}{y}$	Removed

Tab. 9: Vergleich der Veränderungen von clat im Wooster/Abrams Algorithmus [WOO96]

Server	*clat*	*cbw*	*nref*	*size*	*hyb*	Result
A.any file	1s	**10KB/s**	x	y	$\frac{(1-1)\cdot x^{B}}{y}$	Removed
B.any file	1s	**1KB/s**	x	y	$\frac{(1-10)\cdot x^{B}}{y}$	Retained

Tab. 10: Vergleich der Veränderungen von cbw im Wooster/Abrams Algorithmus [WOO96]

Server	*clat*	*cbw*	*nref*	*size*	*hyb*	Result
A.file 1	1s	10KB/s	5	y	$\frac{(1-1)\cdot 5^{B}}{y}$	Retained
A.file 2	1s	10KB/s	3	y	$\frac{(1-1)\cdot 3^{B}}{y}$	Removed

Tab. 11: Vergleich der Veränderungen von NR_i im Wooster/Abrams Algorithmus [WOO96]

28. Wird von den Autoren als HYB Algorithmus bezeichnet.
29. Die Abkürzung ** bedeutet, daß sich W_N im Exponenten befindet.

Die Beispiele in den Tabellen 9, 10 und 11 illustrieren welche Dokumente behalten und welche ersetzt werden würden, bei unterschiedlichen Konditionen.

Server	*clat*	*cbw*	*nref*	*size*	*hyb*	Result
A.file 3	1s	10KB/s	5	50KB	$\frac{(1+1)\cdot 5^{2}}{50}$	See text
A.file 4	1s	10KB/s	3	30KB	$\frac{(1+1)\cdot 3^{2}}{30}$	See text

Tab. 12: Vergleich der Veränderungen von NR_i und S_i im Wooster/Abrams Algorithmus [WOO96]

Zur Tabelle 12: Wenn $W_N > 1$ liegt die Betonung auf NR_i und damit wird File 4 ersetzt, wenn $0<W_N<1$ liegt die Betonung auf S_i und File 3 wird ersetzt.[WOO96]

Parameter:

- $clat_{ser(i)}$ = Zeit für den Verbindungsaufbau zum Server j
- $cbw_{ser(i)}$ = Bandbreite zum Server j
- ser(i) = Server des Dokumentes i
- NR_i = Anzahl der Anfragen an ein Dokument i
- S_i = File-Größe
- W_B = Konstante
- W_N = Konstante

Implementierungsmöglichkeiten:

- Wahl der Konstanten W_B und W_N

Ergebnisse aus Studien:

Time bedeutet hier die geschätzte Zeit, die der Nutzer auf das Herunterladen seines Dokumentes warten muß (Download-Zeit). Die Mittellinie (mean) stellt die durchschnittliche Downloadzeit dar. Der Wooster/Abrams Algorithmus hat eine sehr kleine Downloadzeit und wird nur gelegentlich vom LFU Algorithmus übertroffen. Dagegen sind die Time-Werte für LRU und SIZE sehr schlecht. [WA97]

In den Abbildungen 41 und 42 wird der Wooster/Abrams Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Der Nachteil der LAT-Strategie, nämlich das nicht berücksichtigen der Anzahl der Anfragen für einen bestimmten Film, wird beim Wooster/Abrams Algorithmus kompensiert. Es läßt sich sogar eine Gewichtung der Faktoren implementieren. Aufgrund dieser Gegebenheiten sollte der Wooster/Abrams Algorithmus in einer Simulation genauer untersucht werden.

5.5.23 MIX

Beschreibung:

Bei diesem Algorithmus werden nun vier Faktoren berücksichtigt. Nämlich die File-Größe (S_i), die Anzahl der Anfragen seit der Speicherung des Dokumentes i (NR_i), die vergangene Zeit seit der letzten Referenz eines Files i (LA) und die Download-Verzögerung des letzten Abrufes des Dokumentes i (lat_i). Dabei ist es leicht einzusehen, daß es besser ist ein Dokument zu behalten, wenn eine Wiedergewinnung sehr kostspielig ist (lat_i ist groß) oder das Objekt sehr oft nachgefragt wird (NR_i ist groß).

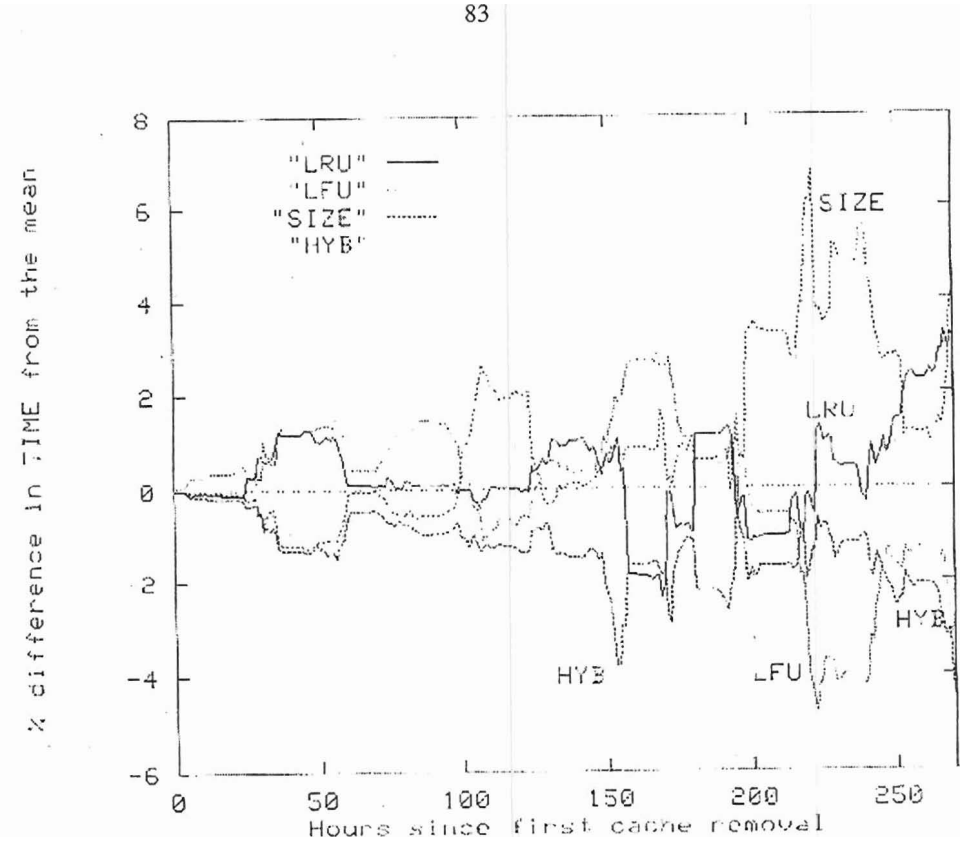


Abb. 40: Veränderung der Download-Zeit für die Algorithmen LRU, LFU, SIZE und HYB[1] [WA97]

1. Der HYB Algorithmus entspricht dem Wooster/Abrams Algorithmus

Dagegen werden diejenigen Files zuerst ersetzt, die lange nicht mehr benötigt wurden (LA ist groß) oder die sehr groß sind (S_i ist groß). Diese Anforderungen werden von einer Vielzahl von Funktionen erfüllt. Das Ziel ist es, eine gutes Verhältnis zwischen diesen verschiedenen Parametern zu finden. Hier wird folgende Kostenfunktion vorgeschlagen:

$$\textbf{Wert} = (lat_i^{r1} * NR_i^{r2}) / (LA^{r3} * S_i^{r4})$$

Dabei ersetzt der MIX- Algorithmus diejenigen Cache-Dokumente mit den geringsten Kosten.

Das Einstellen der Konstanten r_i (i = 1,...,4) ist keine leichte Aufgabe. Die Autoren [30] schlagen nach mehreren Experimenten folgende Werte vor: r_i =1 für i = 2,3,4 und r_1 = 0,1. Diese führen zu einem guten Leistungsverhältnis. [NLN97]

Parameter:

- NR_i = Anzahl der Anfragen an ein Dokument i
- S_i = File-Größe
- LA = vergangene Zeit seit der letzten Anfrage des Dokumentes i
- lat_i = Download-Verzögerung beim letzten Abruf des Dokumentes i
- r_i = Konstanten (i = 1,...,4)

30. Nicolas Niclausse, Zhen Liu, Philippe Nain in [NLN97]

Implementierungsmöglichkeiten:

- Wahl der Konstanten r_i

Ergebnisse aus Studien:

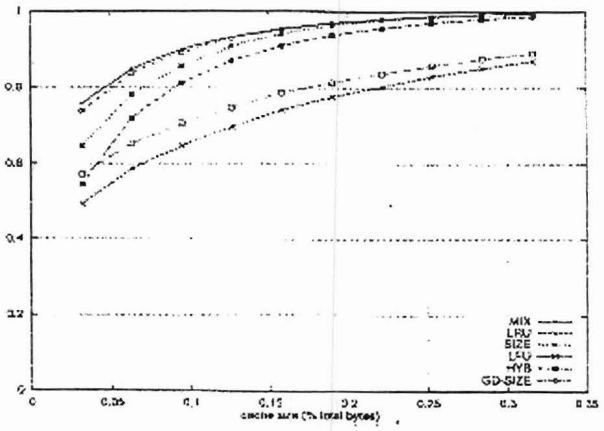


Abb. 41: Trefferquotenvergleich der Algorithmen MIX, LRU, SIZE, LFU, HYB[1] und GD-SIZE [NLN97]

1. entspricht dem Wooster/Abrams Algorithmus

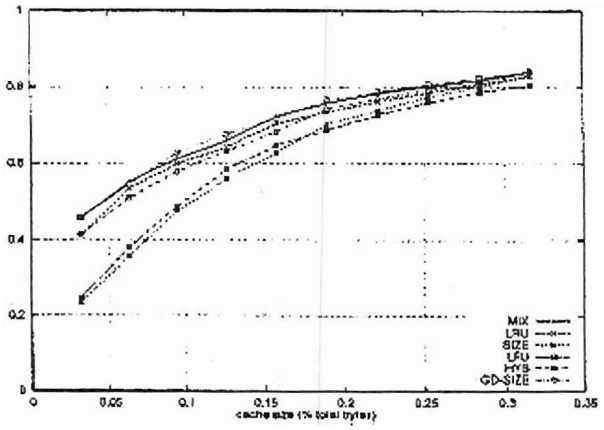


Abb. 42: Byte-Trefferquotenvergleich der Algorithmen MIX, LRU, SIZE, LFU, HYB[1] und GD-SIZE [NLN97]

1. siehe Fußnote 1

Bewertung:

Der MIX stellt einen weiterer interessanter Algorithmus für Videoverteilungssysteme dar. Zwei wichtige Einflußfaktoren, nämlich Ladezeit (bzw. Download-Verzögerung) und Anzahl der Anfragen finden sich in dieser Strategie wieder. Faktoren wie Größe und Zeitpunkt der letzten Anfrage stellen zwar für Videos weniger wichtige Faktoren dar, ihr Einfluß läßt sich hier aber durch die Einführung von Konstanten mit kleinen Gewichtungsfaktoren entsprechend reduzieren.

5.5.24 VALUE

Beschreibung:

Der VALUE Algorithmus stellt eine Erweiterung der WLFU-Strategie dar und versucht dessen Nachteile zu kompensieren. Dazu wird zunächst in die Gleichung von WLFU der benötigte Zeitfaktor integriert.

$$\mathbf{VALUE(T) = NR_i\,(T) / (S_i * T)}$$

T stellt den aktuellen Zeitpunkt und NR_i (T) die Anzahl der Anfragen seit dem Zeitpunkt T dar. Es läßt sich leicht erkennen, daß diese Gleichung ein Äquivalent zum **Original**[31] darstellt. Wann immer man zwei Dokumentenwerte miteinander vergleicht, wird schließlich derselbe Zeitwert (der aktuelle Zeitpunkt) verwendet. Die neue Formel hat trotzdem einen Sinn, da das Verhältnis von Anzahl der Anfrage zum aktuellen Zeitpunkt (NR_i / T) genau der Anfragerate eines Dokumentes entspricht. So läßt sich die Gleichung umschreiben in:

$$\mathbf{VALUE(T) = NR_i\,(T) / (S_i * T) = 1 / (arg\ inter\ NR_i * S_i)}$$

Wobei

$$\mathbf{arg\ inter\ NR_i = T / N_i\,(T)}$$

die durchschnittliche Zeit zwischen den Dokumentenanfragen seit dem Zeitpunkt T darstellt.

Neben dem aktuellen Zeitpunkt T soll noch ein weiterer Zeitfaktor eingeführt werden, nämlich T_{last} als Zeitpunkt der letzten Anfrage an das Dokument i. Damit stellt der Ausdruck T - T_{last} die vergangene Zeit seit der letzten Anfrage dar. Die erweiterte Gleichung lautet nun

$$\mathbf{VALUE(T) = 1/((arg\ inter\ NR_i(T_{last}) + T - T_{last}) * S_i)}$$

Diese Gleichung enthält sowohl die File-Größe S_i, die Anzahl der Anfragen NR_i sowie die vergangene Zeit seit der letzten Anfrage T - T_{last}.

Zusätzlich zu den oben angeführten Faktoren können auch noch Dokumentenänderungen berücksichtigt werden. Gerade im Internet werden Dateien häufig geändert. Sind solche Dateien im Cache gespeichert, so muß ein sogenannter Update vorgenommen werden, um die Dokumente auf den neuesten Stand zu bringen. Um die Berücksichtigung von Dokumentenänderungen in die Formel zu integrieren, bedarf es einiger Modifikationen. Zunächst ändert man arg inter NR_i (T_{last}) zu arg inter NR_i (T_{first}, T_{last}). Dadurch berechnet sich die durchschnittliche Anfragerate nun zwischen der ersten und letzten Anfrage an eine Datei. Ganz zu Anfang wird $T_{first} = 0$ gesetzt. Wird nun ein Dokument zum Zeitpunkt T geändert, so erhalten die beiden Variablen T_{first} und T_{last} den Wert T. Der Anfragenzähler dagegen wird auf eins gesetzt. Somit berücksichtigt der arg inter NR_i nun nicht nur die Referenzhäufigkeit, sondern auch die Anzahl der Modifikationen von Dokumenten.

$$\mathbf{VALUE\ (T) = 1 / ((((T_{last} - T_{first}) / NR_i(T_{first}, T_{last})) + T - T_{last}) * S_i\ )}$$

Die Files werden nun anhand ihres Wertes zur Ersetzung ausgewählt, d.h. Dokumente mit einem kleineren Value Werten werden als erstes ersetzt.

Bei dieser Strategie ist zu beachten, daß sich die Werte der Dokumente mit der Zeit ändern, d.h. die Anfangsbeziehung zwischen zwei Dokumenten ist nicht konstant. Außerdem muß noch erwähnt wer-

31. Gleichung der WLFU-Strategie

den, daß nur die Dokumente potentielle Ersetzungskandidaten darstellen, deren Wert geringer ist als der des neuhinzukommenden Files. Die Komplexität der Strategie wird dadurch leider nur wenig gemindert.

Zum besseren Verständnis dient noch folgendes Beispiel:

Dokument A hat eine Größe von 6 K und wurde nur einmal zum Zeitpunkt 1 frequentiert. Das Dokument B mit der Größe von 5 K wurde zweimal angefragt, nämlich zu den Zeiten 2 und 4. T_{first} wird wie oben erwähnt am Anfang auf Null gesetzt.

Zu jedem Zeitpunkt T > 4 ergeben sich damit folgende Werte:

$$VALUE(A) = 1 / 6T$$

$$VALUE(B) = 1 / 5 (T + 2)$$

Nun werden die Values zu zwei unterschiedlichen Zeiten miteinander verglichen:

1. T = 5: VALUE (A) = 1/30 , VALUE(B) = 1/35

Zum diesem Zeitpunkt würde Dokument B ersetzt werden.

2. 2. T = 11: VALUE(A) = 1/66 , VALUE(B) = 1/65

Zum diesem Zeitpunkt würde Dokument A ersetzt werden.

Dies bedeutet, daß mit der Zeit das Dokument B wertvoller wird als das Dokument A und zwar herrührend durch die kleiner File-Größe von B. [TAT97]

Parameter:

- NR_i (T) = Anzahl der Referenzen einer Datei i ab dem Zeitpunkt T
- S_i = File-Größe
- T = aktueller Zeitpunkt
- T_{last} = Zeitpunkt seit der letzten Anfrage an eine Datei i
- T_{first} = Zeitpunkt der ersten Anfrage an eine Datei i
- $T - T_{last}$ = Zeitraum seit der letzten Anfrage an eine Datei i (entspricht LA_i)

Bewertung:

Für Video-on-Demand-Systemen ist es überaus sinnvoll, wenn nur solche Filme im Cache gespeichert werden, die wertvoller als die schon im Cache vorhandenen Videos sind. Dieses Prinzip scheint auch der VALUE Algorithmus anzuwenden. Allerdings wird nicht beschrieben, wie man den Wert eines neuen Filmes überhaupt ermitteln soll. Falls der Film schon einmal im Cache gespeichert wurde, könnte man natürlich die alten Werte für die Berechnung verwenden. Durch den vorhandenen Zeitunterschied wird sich ein effizienter Vergleich als sehr schwierig gestalten.

Das bei diesem Algorithmus auch Modifikationen berücksichtigt werden, ist für WWW-Seiten sehr nützlich. Filme allerdings durchlaufen in der Regel keine Änderungen, sondern bleiben in ihrem Inhalt konstant. Die Berücksichtigung von Modifikationen führt somit zu keinerlei Leistungsverbesserungen.

5.5.25 RBC (Ressource Based Caching Algorithm)

Beschreibung:

Bei dem RBC Algorithmus wird zunächst einmal entschieden, ob ein Dokument überhaupt gecacht werden soll. Diese Entscheidung basiert auf dem vom File benötigten Ressourcen (Speicherplatz und Bandbreite) und seinem entsprechenden caching gain. Der caching gain einer Datei wird als sein Bei-

trag zur gesamten Cache-Leistung definiert. Er berechnet sich bei gegebener Bandbreite b_i und einer geschätzten Anzahl von konkurrierenden Lesern r_i[32] wie folgt:

$$g_i = r_i * b_i$$

Um nun einen geeigneten Kandidaten für die Ersetzung zu finden, ordnet der RBC die angeforderten Files und die im Cache befindlichen Dokumente nach verschiedenen Kriterien in einer bestimmten Reihenfolge an. Den Wert eines Files innerhalb dieser Reihe bezeichnet man als goodness. Die Kriterien die diese Rangfolge bestimmen bestehen aus caching gain g_i, Speicherplatzbedarf s_i, Bandbreite b_i sowie dem cache state (U_s , U_b). Der cache state ist definiert als ein Paar (U_s , U_b), wobei U_s und U_b die Verwendung von Speicherplatz und Bandbreite darstellen. Die genaue Berechnung der beiden Wert erfolgt mit den angeführten Gleichungen:

$$U_s = \frac{1}{S}\sum_{E_i \in \mathcal{E}} \hat{s}_i; \qquad U_b = \frac{1}{B}\sum_{E_i \in \mathcal{E}} \hat{b}_i; \quad 0 \le U_s, U_b \le 1.$$

Dabei wird S als die Größe des Cache im Server und B als seine Bandbreite definiert.

Um sicherzustellen, daß Speicherplatz und Bandbreite effektiv genutzt werden, kann das Dokument je nach Möglichkeit vollständig oder in Blöcken übertragen werden. So kann man verhindern, daß entweder Speicherplatz oder Bandbreite überlastet ist während die andere Ressource nicht voll ausgenutzt wird. In Abbildung 43 sind die möglichen Beziehungen von U_s und U_b dargestellt. Im Bereich des dunklen diagonalen Streifen befindet man sich in einer „gerechten" Verteilung von Speicherplatz und Bandbreite.

Die Dateien im Cache werden nun in aufsteigender Reihenfolge nach ihrem goodness Wert geordnet. Ausschließlich Files mit einem geringeren goodness Wert, als dem des angeforderten Dokumentes stellen potentielle Ersetzungskandidaten dar. Bei der Auswahl beginnt man zunächst bei der Datei mit dem schlechtesten goodness Wert und arbeitet sich langsam nach oben, bis genügend Speicherplatz für das neue Dokument zur Verfügung steht.

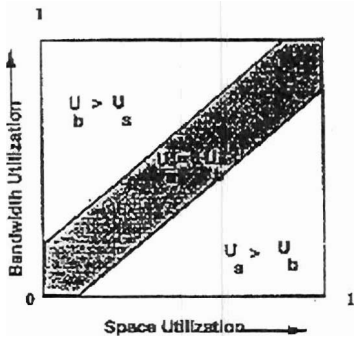


Abb. 43: Cache state [TVD+96]

Die Berechnung des goodness Wertes hängt davon ab, ob der Cache unconstrained, space constrained, bandwidth constrained oder bandwidth and space constrained ist.

1. Unconstrained: Ein Cache ist unbeschränkt, wenn der freie Speicherplatz und die verfügbare Bandbreite die gestellten Anforderungen des zu cachenden Dokumentes übersteigt. In diesem Fall wird die Datei einfach im Cache gespeichert.
2. Space Constrained: Ein Cache ist space constrained, wenn die verfügbare Bandbreite für die

32. Nähere Erläuterungen zur Berechnung von r_i siehe Seite 5ff [TVD+97].

Unterbringung einer neuen Datei ausreicht, aber der freie Speicherplatz nicht genügt. In diesem Fall wird jeder neuen Datei ein goodness Wert G_i zugeordnet, der den Gewinn pro Speichereinheit angibt (gain density). Somit erhält man für einen caching gain g_i und dem Speicherplatzbedarf s_i den folgenden goodness Wert:

$G_i = g_i / s_i$

3. Bandwidth Constrained: Ein Cache ist bandwidth constrained, wenn der freie Speicherplatz für die Unterbringung einer neuen Datei ausreicht, aber die verfügbare Bandbreite nicht genügt. In diesem Fall, ist die Ressource auf die man Rücksicht nehmen muß der Bandbreitenbedarf einer Datei. Der goodness Wert einer neuen Datei wird durch den cache gain pro Bandbreiteneinheit angegeben.

   $G_i = g_i / b_i$

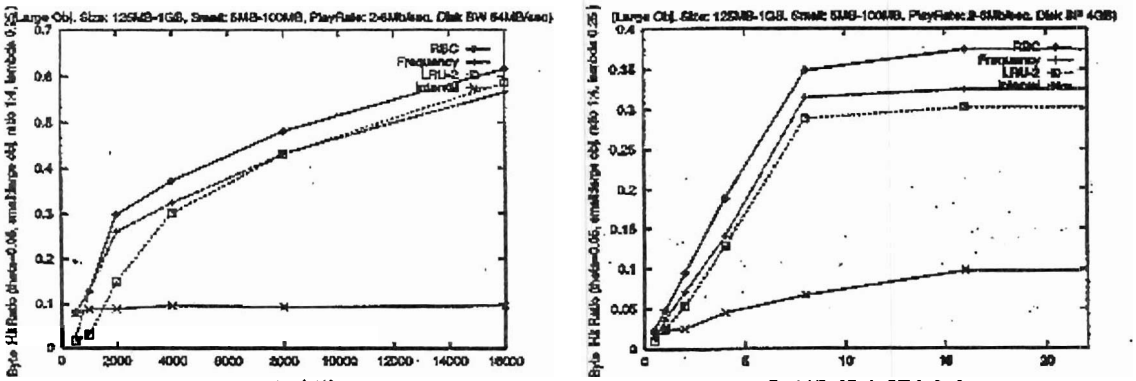
4. Bandwidth and Space Constrained: Ein Cache ist bandwidth and space constrained wenn weder Speicherplatz noch Bandbreite für die Speicherung einer neuen Datei ausreichen. In diesem Fall werden die Dateien im Cache in zwei Listen eingeordnet. In einer s-Liste werden die Files nach ihrem Gewinn pro Speichereinheit ($G_i = g_i / s_i$) und in einer b-Liste nach ihrem Gewinn pro Bandbreiteneinheit ($G_i = g_i / b_i$) angeordnet. Der Algorithmus entfernt aus beiden Listen Dateien bis genügende Ressourcen für die neuen Dokumente freigestellt sind. Wenn S_{free} und B_{free} den zur Zeit verfügbaren Speicherplatz und Bandbreite darstellen, so ist zu entscheiden welche Liste bei jedem Schritt benutzt wird. RBC untersucht dazu das Verhältnis von freiem Speicherplatz zu besetzten Speicherplatz (S_{free} / s_i) und von freier Bandbreite zu besetzter Bandbreite (B_{free} / b_i). Ist S_{free} / s_i kleiner als B_{free} / b_i so wird die Datei am oberen Ende der s-Liste zur Ersetzung ausgewählt, im anderen Fall wird der Kandidat vom oberen Ende der b-Liste gewonnen. Nachdem das Ersetzungsopfer ausgewählt wurde, müssen die Werte für S_{free} und B_{free} neu berechnet werden. Danach wird der Vorgang der Listenauswahl wiederholt. [TVD+96] [TVD+97]

Der RBC Algorithmus wird auch als **IBSC** Algorithmus (Integrated Bandwidth and Space Constrained Algorithm) bezeichnet.

Parameter:

- s_i = Speicherplatzbedarf eines Dokumentes i
- b_i = Bandbreite eines Dokumentes i
- g_i = caching gain eines Dokumentes i
- (U_s , U_b) = cache state
- r_i = konkurrierenden Lesern
- G_i = goodness Wert eines Dokumentes i

Ergebnisse aus Studien:



(a) Varying Cache Size (b) Varying Bandwidth

Abb. 44: Byte-Trefferquotenvergleich der Algorithmen RBC, Frequency[1], LRU-2 und Interval [TVD+97]

1. Frequency = z.B. LFU, Interval = z.B. Static Caching

Bewertung:

Beim RBC Algorithmus wird nicht jede aufgerufene Datei sofort gespeichert, sondern anhand ihres goodness Wertes überlegt, ob das Dokument überhaupt in den Cache geladen werden soll. Dies Überlegung sollte bei einem geeigneten Algorithmus für VoD-Systeme auf jeden Fall auch gemacht werden. Es hat keinen Sinn Filme im Cache zu speichern, von denen man mit großer Sicherheit weiß, das sie sehr lange nicht mehr aufgerufen werden (z.B. spezielle Dokumentarfilme, Kunstfilme usw.). Schließlich wird durch jede Neuspeicherung ein anderer Filme aus dem Cache vertrieben, der dann wieder kosten- und zeitaufwendig in den Cache zurückgeholt werden muß.

Leider hängt der goodness Wert bei dieser Strategie nur vom Speicher- und Bandbreitenbedarf ab. Für Videoverteilungssysteme müssen auf jeden Fall die Anzahl der Anfragen (Popularität) und die Ladezeit mit berücksichtigt werden.

Trotzdem sollte der RBC Algorithmus (eventuell mit leichten Modifikationen) in weiteren Simulationen auf seine Eignung untersucht werden.

6 Kooperation der Server

6.1 Cooperative Caching

Cooperative Caching wurde zunächst für netzwerkfähige Dateisysteme entwickelt. Innerhalb eines Dateisystems gibt es drei Speichertypen, nämlich die lokalen Caches der Clients, die Caches des Servers und die Festplatte des Servers[33]. Benötigt nun eine User[34] ein Dokument, so kann er es von einem dieser drei Speicher erhalten. Dabei ist es für ihn natürlich am günstigsten, das gefragte Dokument vom eigenen Cache zu laden. Die nächste Möglichkeit liegt im Server Cache und erst dann, wenn es auch dort nicht zu finden ist, wird die Anfrage an die Festplatte des Servers geschickt. Die Caches der anderen Clients, die auch mit dem Server verbunden sind konnten vom User bisher nicht genutzt werden. Durch das Prinzip des Cooperative Caching wird nun eine vierte Stufe in der Speicherhierarchie

33. server disk
34. User und Clients werden hier begrifflich gleichgesetzt

eingeführt. Jeder Client kann nun auf die Caches der anderen Clients zugreifen. Dabei gibt es verschiedene Möglichkeiten ein solches System zu implementieren.

Cooperative Caching versucht nun die Leistung eines herkömmlichen netzwerkfähigen Dateisystems zu verbessern, indem der Inhalt der Client Caches koordiniert wird und die Anfragen eines Users auch aus anderen Client Caches befriedigt werden können.

Dabei führt die Anwendung des Cooperative Caching zu mehreren Vorteilen:

1. *Verbesserung der Leistung:* Die globale Trefferquote wird verbessert und die Anfragen direkt an die Festplatte des Servers reduziert. Dabei behalten die einzelnen Client Caches ihren jeweiligen Speicherinhalt überwiegend bei, so daß die eigene Hit-Rate aufrechterhalten wird.
2. *Geringere Belastung des Servers:* Der Datenverkehr zwischen Server und Clients wird entlastet, da es oft genügt dem Client mitzuteilen, wo sich die gesuchte Datei aufhält. Solche Mitteilungen benötigen natürlich viel weniger Leitungskapazitäten, als eine Übertragung einer gesamten Datei.
3. *Kostenvorteil:* Die Errichtung eines Systems, welches auf Cooperative Caching beruht ist billiger als die Verwendung von extrem großen Server Caches.

Cooperative Caching führt also eine vierte Stufe in die Cache-Hierarchie eines Dateisystems ein. Einzelne Dokumente können nun nicht nur im lokalen Speicher, im Server oder auf der Festplatte des Servers gefunden werden, sondern auch in den Caches anderer User. Mittels sogenannter *Cooperative Caching Strategien*[35] kann diese vierte Stufe auf verschiedene Arten verwaltet werden. [DWA+94]

Natürlich läßt sich das Prinzip des Cooperative Caching auch in anderen, ähnlichen Systemen implementieren. Das World Wide Web z.B. stellt seinerseits ein Dokumentenverteilungssystem basierend auf dem Client/Server Modell dar und ist somit ebenso, wie ein verteiltes Dateisystem für den Einsatz von Cooperative Caching geeignet. In einem solchen System, wie das World Wide Web, besteht aber aus Datenschutzgründen keine Möglichkeit einer Koordination der Client Caches. Da es im WWW nicht nur den in Dateisystemen üblichen einen Server gibt, sondern mehrere, können nun die Server untereinander koordiniert werden. Eine mögliche Variante des Cooperative Caching mittels mehrere Server wird in der Abbildung 20 dargestellt. [MLB95]

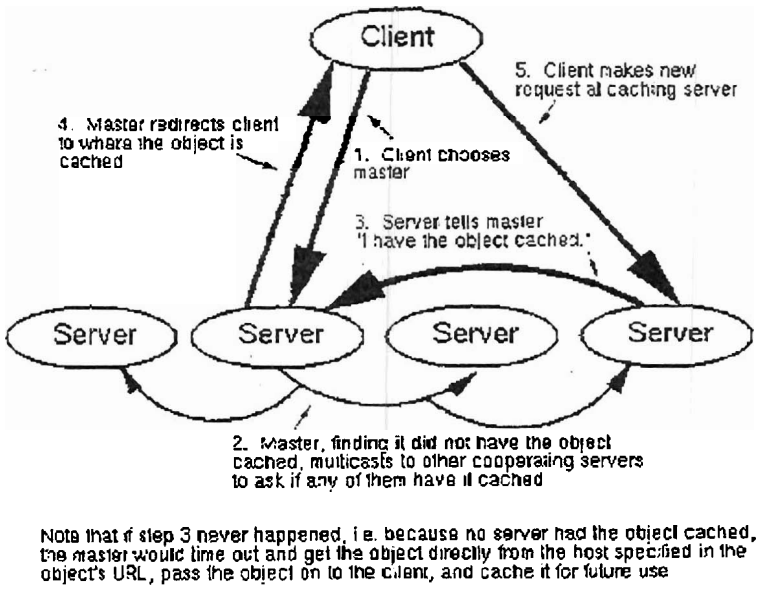


Abb. 45: Illustration eines verwendeten Protokolls beim Cooperative Caching [MLB95]

35. siehe Kapitel 7.3

Eine Möglichkeit Cooperative Caching mit Non-Cooperative Caching zu vergleichen, liegt in der Messung der Trefferquote. In der Tabelle 13 wird dies getan und es ist deutlich zu sehen, daß sich schon die Kooperation zwischen zwei Servern lohnt. [KS98]

Loc.	Co-operation with				
	none	S	M	B	full
S	20.90	–	22.68	25.79	26.57
M	25.09	26.14	–	30.03	30.49
B	32.69	33.47	33.97	–	34.64

(a) Hit rates for number of bytes.

Loc.	Co-operation with				
	none	S	M	B	full
S	29.72	–	32.70	35.02	36.16
M	37.63	39.32	–	42.10	42.94
B	69.90	70.49	70.75	–	71.21

(b) Hit rates for number of requests.

Tab. 13: Trefferquotenvergleich für drei Proxy Servers bei unterschiedlichen Kombinationen der Kooperation [KS98]

Die Symbole B, M und S stehen für die drei Proxy Server und bedeuten „Big" , „Medium" und „Small". Als „Full" Kooperation wird die Kooperation eines Servers mit den anderen beiden Servern bezeichnet. [KS98]

6.2 Möglichkeiten des Einsatzes in Videoverteilungssystemen

In diesem Kapitel sollen zunächst noch keine allgemeinen Unzulänglichkeiten und Annahmen aufgezählt werden, wie es bei den bisherigen entsprechenden Kapiteln geschehen ist, sondern es soll überlegt werden, inwiefern Cooperative Caching für Videoverteilungssysteme geeignet ist.

Zunächst einmal muß geprüft werden, ob die Systemarchitektur eines Videoverteilungssystems die Implementierungvoraussetzungen von Cooperative Caching erfüllt. In dem Kapitel 3.2 wird ein mögliches Modell eines VoD-Systems erläutert und dargestellt. Es handelt sich also eindeutig um ein verteiltes System mit vielen Clients, mehreren Servern und einer Hauptbibliothek. Cooperative Caching ist somit prinzipiell möglich.

Als nächstes wird überlegt, welche Caches miteinander kommunizieren sollen. Dabei sind die Clients und die Server die möglichen Kandidaten. Die Clients allerdings speichern höchstens einen Film, nämlich den, den sie sich gerade ansehen. Meist wird es sogar so sein, daß der Film nicht als ganzes vom Server an den Client Cache übertragen wird, sondern eine Echtzeitübertragung vom Server an den entsprechenden Client stattfindet und der Client höchstens kurzfristig einzelne Blöcke puffert. Somit bleiben als Kooperationspartner nur noch die einzelnen Server des Systems übrig. Dies ist auch viel sinnvoller, da die Server jeweils mehrere Videos in ihrem Cache gespeichert haben. Die Festplatte des Servers, die in Kapitel 7.1 erwähnt wird, soll hier der Bibliothek entsprechen, an die nur solche Anfragen gesendet werden sollen, die sonst nicht erfüllt werden können. Aber auch weit entfernte Server könnten als die oben erwähnte Festplatte definiert werden.

Wenn man nun das Cooperative Caching Prinzip in ein Videoverteilungssystem implementiert, muß man entscheiden, welche Server miteinander kommunizieren (oder kooperieren) sollen. Vorteilhaft ist dabei möglichst nahe beieinander liegende Server zu wählen, um die Übertragungswege möglichst kurz zu halten. Dies muß aber je nach System und Anforderung vom Implementierer festgelegt werden.

Am Ende dieses Kapitels sollen nun noch die bei der Bewertung gemachten Annahmen erläutert werden.

Wenn bei den folgenden Strategien von Client Caches die Rede ist, erfolgt die Bewertung trotzdem immer unter der Annahme, daß nicht die einzelnen Client Caches, sondern die (vorher bestimmten) *Servern miteinander kooperieren*. Dabei muß für einige Algorithmen auch ein *„übergeordneter" Server festgelegt* werden, der Informationen an andere weitergibt, herausfindet ob es sich bei einem Dokument um ein Duplikat handelt usw.

Des weiteren muß man sich vergegenwärtigen, daß die folgenden Algorithmen nur dann reibungslos funktionieren, wenn alle Dateien die gleiche Größe haben. Ist dies nämlich nicht der Fall, so ist der Austausch von zwei Dokumenten problematisch. Aber auch bei unterschiedlichen *Dateien-Größen* lassen sich die folgenden Cooperative Caching Algorithmen trotzdem anwenden. Dazu muß dann z.B. für eine große Datei, die in einen anderen Cache geschoben werden soll, mehrere kleinere Dokumente aus dem neuen Cache entfernt (bzw. auch verschoben) werden. Da Filme in der Regel alle sehr groß sind, wird es höchstens passieren, daß zwei Filme für einen anderen Filme gelöscht (bzw. verschoben) werden müssen. Dies ist durchaus noch vertretbar, wenn man dafür einen dringend benötigten Film erhält.

Die Cooperative Caching Algorithmen sind immer in Verbindung mit den in den Caches intern angewendeten Ersetzungsalgorithmen zu sehen. Es können also durchaus verschiedene Algorithmen im Innern eines Systems existieren. Der verwendete Cooperative Caching Algorithmus verwaltet dagegen den globalen Teil des Systems, indem er Regeln aufstellt wie die einzelnen Caches untereinander kommunizieren, Dateien verschicken usw.

6.3 Strategien

6.3.1 Direct Client Cooperation

Beschreibung:

Dieser sehr einfache Cooperative Caching Algorithmus erlaubt aktiven Clients ungenutzte User-Speicher als eigenen Zwischenspeicher zu verwenden. Dabei wird der Überschuß des lokalen Caches vom aktiven User direkt an den ungenutzten Cache weitergereicht. Der aktive Client kann dann auf diesen entfernten Cache zugreifen, um seine Anfragen zu befriedigen. Wird dieser Cache nun wieder vom eigentlichen Besitzer genutzt, so wird die Verbindung zum vorherigen User unterbrochen. Das System muß nun Kriterien definieren, um aktive und nicht-aktive Clients festzulegen, und es muß einen Mechanismus zur Verfügung stellen, damit ungenutzte Caches lokalisiert werden können.

Der Direct Client Cooperation Algorithmus ist sehr ansprechend, da er unkompliziert ist. Man kann ihn ohne Modifikationen des Servers implementieren. Die Server müssen nur erkennen, daß ein Client einen vorübergehend vergrößerten Cache zur Verfügung hat. Ein Nachteil entsteht durch den Mangel an Serverkoordination. Aktive User können keinen Nutzen aus dem Inhalt der Speicher von anderen Nutzern ziehen. Jede Anfrage eines Nutzers muß an den Server Cache oder gar die Festplatte übertragen werden, wenn die gewünschte Datei nicht im eigenen Cache vorhanden ist. Dabei kann es sein, daß die angefragte Datei schon in einem Cache eines anderen Nutzers gespeichert ist und eine Anfrage an die Festplatte somit unnötig wäre. [DWA+94]

Parameter:

- Ersetzungsalgorithmen für Client Caches

Implementierungsmöglichkeiten:

- Kriterium zur Definition aktiver und nicht-aktiver Clients
- Wahl der Ersetzungsalgorithmen für die Client Caches

Ergebnisse aus Studien:

In der Abbildung 47 wird der Direct Client Cooperation Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Der Direct Client Cooperation Algorithmus wird in Video-on-Demand-Systemen praktisch leider nicht anwendbar sein. Der Fall, daß ein Server ungenutzt bleibt tritt kaum ein. Außerdem wird hier der Inhalt der anderen Server überhaupt nicht genutzt und somit der Hauptvorteil des Cooperative Caching in Videoverteilungssysteme nicht wahrgenommen.

6.3.2 Greedy Forwarding

Beschreibung:

Der Greedy Forwarding Algorithmus stellt eine andere recht einfache Cooperative Caching Strategie dar. Dabei wird der Inhalt aller Caches eines Systems als eine globale Ressource betrachtet, aus der die Anfragen von Usern befriedigt werden können. Allerdings versucht der Algorithmus nicht den Inhalt der verschiedenen Caches miteinander zu koordinieren. Jeder Client verwaltet seinen Cache nach eigenem Gutdünken, ohne den Inhalt der anderen Speicher zu beachten oder gar den potentiellen Bedarf anderer Clients zu berücksichtigen. Wenn ein User die gesuchte Datei nicht in seinem lokalen Cache findet, so wird er den übergeordneten Server nach diesem Dokument ersuchen. Hat der Server die gewünschte Datei in seinem Cache, so schickt er diese an den entsprechenden User. Ansonsten wird der Server eine Dateienliste, welche den Inhalt aller Client Caches einschließt, zu Rate ziehen. In dem Fall, daß sich im Cache irgendeinen Users die angefragte Datei befindet, übermittelt der Server die ursprüngliche Anfrage an den entsprechenden Client. Sobald dieser die Anfrage erhalten hat schickt er die Datei direkt an den User, der die Anfrage gestartet hat. Das Dokument wird also nie zurück über den Server gesendet, denn dies würde zu unnötigen Verzögerungen und Belastungen führen. Wenn nun kein Client die gesuchte Datei in seinem Cache gespeichert hat, muß die Nachfrage über die Festplatte des Servers befriedigt werden.

Der Vorteil dieser Strategie liegt darin, daß die User die volle Kontrolle über ihren lokalen Caches behalten und darüber hinaus Nutzen aus den anderen Caches ziehen können. Außerdem entsteht keine zu hohe Belastung des Servers. Trotzdem gibt er auch hier Probleme. Durch die fehlende Koordination unter den Clients werden unnötige Dateienduplikate in den Caches gespeichert. [DWA+94]

Parameter:

- Ersetzungsalgorithmen für Client Caches

Implementierungsmöglichkeiten:

- Wahl der Ersetzungsalgorithmen für lokal verwaltete Caches

Ergebnisse aus Studien:

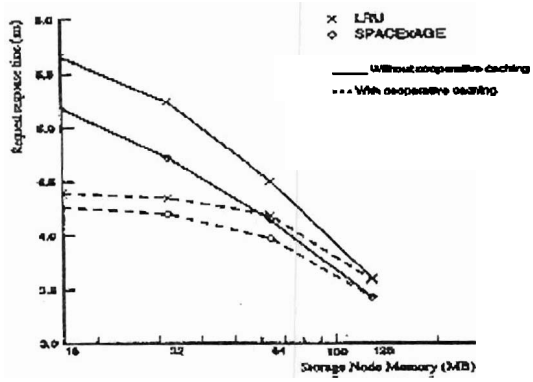


Abb. 46: Vergleich der Antwortzeiten der Algorithmen LRU und SPACExAGE mit und ohne Cooperation Caching [RED95]

In der Abbildung 46 wird als Cooperative Caching-Strategie der Greedy-Forwarding benutzt. Man erkennt hier sehr deutlich, daß der Einsatz des Greedy-Forwarding die Antwortzeiten der beiden Algorithmen deutlich verbessert.

In der Abbildung 47 wird der Greedy Forwarding Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Der Greedy Forwarding Algorithmus stellt für Videoverteilungssysteme durchaus eine geeignete Strategie dar. Die einzelnen Server können Filme untereinander austauschen und vermeiden somit die viel längeren Übertragungswege zur Bibliothek. Jeder Server kann seinen Cache mittels einer Ersetzungsstrategie nach eigenem Ermessen verwalten, allerdings haben die einzelnen Server auch keinen Einfluß auf den Inhalt der anderen Server Caches. Vor allem bei Filmen, die in einem bestimmten Bereich des Systems nur einmal vorkommen besteht die Gefahr, daß sie gelöscht werden. Benötigt dann ein anderer Server dieses Video, muß dieses wieder über die weit entfernte Bibliothek geholt werden.

Einen Nachteil hat dieser Algorithmus aber trotzdem. Dadurch, daß nur der übergeordnete Server Kenntnis über den Inhalt der anderen Caches hat, wird der Datenverkehr zwischen diesem Server und den anderen entsprechend hoch sein. Außerdem verliert das System durch die Implementierung von "zentralen" Servern seine Robustheit.

6.3.3 Centrally Coordinated Caching

Beschreibung:

Der Centrally Coordinated Caching Algorithmus fügt nun dem Greedy Forwarding Algorithmus den Aspekt der Koordination hinzu, indem er den Cache des Users unterteilt. Einerseits in einen lokal verwalteten Teil, der nach dem Wunsch des Clients benutzt wird und andererseits in einen global regierten Teil, der durch den Server als eine Erweiterung seines eigenen zentralen Caches betrachtet wird. Wenn nun eine User die gewünschte Datei nicht in seinem lokalen Cache findet, sendet er eine Anfrage an den übergeordneten Server. Dieser überträgt die angeforderte Datei, wenn er diese in seinem eigenen Speicher findet. Ist das Dokument in seinem Speicher nicht vorhanden, überprüft er die Caches der anderen Clients. Lokalisiert der Server die angefragte Datei in einem Client Cache, so übermittelt er die Anfrage des ursprünglichen Users an den entsprechenden Client. Ist die Datei überhaupt nicht vorhanden muß die Anfrage von der Festplatte des Servers befriedigt werden.

Der Server verwaltet den globalen Teil der Client Caches unter Verwendung der LRU-Strategie. Löscht der Server eine Datei aus seinem eigenen lokalen Cache, um Platz für eine, von der Festplatte neu geladene, Datei zu schaffen, so wird das ersetzte Dokument nun im zentral koordinierten Cache gespeichert. Die Datei, die im globalen Cache am längsten nicht mehr benötigt wurde wird gelöscht.

Der primäre Vorteil dieser Strategie ist die hohe globale Trefferquote, die durch die globale Verwaltung eines Großteils der Speicherressourcen erreicht wird. Der Hauptnachteil besteht in der Reduzierung der Trefferquote der einzelnen Clients, welche durch die Verkleinerung des lokal verwalteten Caches entsteht. Dies bedeutet auch, daß die Anzahl der Serveranfragen durch die Koordination ansteigen wird. [DWA+94]

Parameter:

- LRU-Strategie (für global verwalteten Cache)
- Ersetzungsalgorithmen für lokal verwaltete Caches

Implementierungsmöglichkeiten:

- Feste Einteilung des Caches in einen lokalen und globalen Teil
- Wahl der Ersetzungsalgorithmen für lokal verwaltete Caches

Ergebnisse aus Studien:

In der Abbildung 47 wird der Centrally Coordinated Caching Algorithmus mit anderen Algorithmen verglichen.

Bewertung:

Ein vorher festgelegter übergeordneter Server kann beim Centrally Coordinated Caching Algorithmus Filme, die aus lokalen Caches ausgewiesen werden, in die global verwalteten Caches der anderen Server speichern. Damit können Filme länger im System gehalten werden, die man vielleicht doch noch einmal benötigt. Die Verwendung der LRU-Strategie innerhalb dieses globalen Caches ist bei VoD-Systemen allerdings nicht sinnvoll.

Die statische Einstellung von lokal zu global verwalteten Teil eines Cache ist nachteilig, da eine dynamische Anpassung an die jeweiligen Gegebenheiten somit nicht möglich ist. Es kann aber gut sein, daß ein Server zu bestimmten Zeiten einen großen lokalen Speicherraum benötigt und dagegen zu anderen Zeiten gerne Speicherplatz für den globalen Teil zur Verfügung stellt.

Ein weiterer Kritikpunkt liegt darin, daß nur der übergeordnete Server Kenntnis über den Inhalt der anderen Caches hat. Damit wird der Datenverkehr zwischen diesem Server und den anderen entsprechend hoch sein und die Robustheit des Systems wird verringert. Außerdem wirkt es sich nachteilig aus, daß auch hier auf Filme, die nur einmal in einem bestimmten Bereich des Systems gespeichert sind keine besondere Rücksicht genommen wird.

6.3.4 Hash-Distributed Caching

Beschreibung:

Der Hash-Distributed Caching Algorithmus unterscheidet sich vom Centrally Coordinated Caching Algorithmus dahingegen, daß er den zentral verwalteten Cache basierend auf einen Dateien-Identifizierer unterteilt, d.h. jeder Client managt einen Teil des globalen Caches. Der Server sendet die Dokumente, die aus dem lokal verwalteten Cache ausgewiesen werden zu einem anderen Client, ausgewählt durch den Dateien-Identifizierer. Bei einem lokalen Miss fragt der User nun diesen verteilten Cache an,

indem er seine Anfrage direkt an den zuständigen Client schickt. Dieser Client versorgt den anderen User mit dem entsprechendem Dokument, wenn er die Datei gerade gespeichert hat. In dies nicht der Fall, so sendet er die Anfrage an den Server.

Der Hash-Distributed Caching Algorithmus hat annähernd die gleiche Hit-Rate, wie der Centrally Coordinated Caching Algorithmus[36]. Allerdings besteht bei der Hash-Distributed Caching-Strategie noch ein weiterer Vorteil. Da die meisten Anfragen durch die kooperierenden Caches befriedigt werden, wird der Server entlastet, d.h. der Datenverkehr im Netzwerk sinkt.

Natürlich besteht auch hier wieder die Einteilung zwischen lokal und global verwalteten Cache. Ebenso wird in der Regel der LRU Algorithmus zur Verwaltung des globalen Caches verwendet. [DWA+94]

Parameter:

- LRU-Strategie (für global verwalteten Cache)
- Ersetzungsalgorithmen für lokal verwaltete Caches

Implementierungsmöglichkeiten:

- Feste Einteilung des Caches in einen lokalen und globalen Teil
- Wahl der Ersetzungsalgorithmen für lokal verwaltete Caches

Bewertung:

Im Unterschied zum Centrally Coordinated Caching Algorithmus erfolgen die Anfragen nicht über den übergeordneten Server, sondern die anderen Server wissen zumeist, wo sich der gesucht Film gerade aufhält. Damit wird der Datenverkehr zum „Hauptserver" stark entlastet.

Ansonsten gilt die gleiche Bewertung wie beim Centrally Coordinated Caching Algorithmus.

6.3.5 N-Chance Forwarding

Beschreibung:

Mit dem N-Chance Forwarding Algorithmus läßt sich der Anteil am Cache, den jeder Client selber verwaltet, dynamisch einstellen. Der Größenanteil hängt dabei von der Aktivität des Users ab. Die N-Chance Forwarding Strategie modifiziert den Greedy Forwarding Algorithmus auch dahingegen, daß Dateien die nur in einem einzigen Client Cache gespeichert sind (sogenannte singlets) bevorzugt behandelt werden. Ansonsten arbeitet der N-Chance Forwarding Algorithmus genauso wie der Greedy Forwarding Algorithmus.

Die N-Chance Forwarding Strategie versucht also das Entfernen von singlets aus den Client-Speichern zu vermeiden. Wenn ein Client eine Datei entfernen möchte, muß er zunächst überprüfen, ob von dieser Datei eine Kopie in einem anderen Cache vorhanden ist. Diese Überprüfung erfolgt durch eine Anfrage an den Server. Ist das Dokument ein singlet, so wirft man die Datei nicht einfach weg, sondern der User setzt den rückwärtslaufenden Zähler der entsprechenden Datei auf n. Danach wird die Datei an einen zufälligen User übermittelt und dem Server eine Nachricht übersendet, daß das singlet weitergegeben wurde. Der Client, welcher nun die Datei erhalten hat, speichert das Dokument in seinen global verwalteten Cache. Außerdem wird die Datei in seine LRU-Liste so eingefügt, als ob die Datei gerade frequentiert wurde.

Erreicht das singlet das Ende der LRU-Liste, so wird der Zähler um 1 herabgesetzt und die Datei weitergereicht, es sei denn der Zähler erreicht den Wert Null. In solch einem Fall wird die Datei einfach

36. Simulationsergebnisse auf Seite 5 [DWA+94]

gelöscht. Fragt nun eine User nach einem singlet, so wird der Zähler angehalten und die Datei ganz normal gecacht (der Zähler wird dann natürlich zurückgesetzt), während der Client, der bisher das singlet gespeichert hat die Datei aus seinem Cache entfernt.

Der Parameter n zeigt wie lange es einem singlet erlaubt ist durch verschiedene LRU Listen von Clients durchzulaufen[37] , bevor es gelöscht werden kann. Der Greedy Forwarding Algorithmus wäre z.B. eine einfachere Variante der obigen Strategie mit der Einstellung n = 0.

Der Algorithmus stellt außerdem eine dynamische Einteilung für jede Cache-Zuweisung eines Clients zwischen den lokalen Dateien (Dateien, die aufgrund der Anfrage des Users gespeichert werden) und globalen Dateien (Dateien, die zur Leistungsverbesserung des Gesamtsystems gespeichert werden) bereit. Aktive Clients tendieren dazu, jede globale Datei so schnell wie möglich aus ihrem lokalen Speicher auszuweisen. Untätige Clients dagegen tendieren dazu, globale Dateien anzusammeln und sie für eine lange Zeit im Cache zu halten. Eine Verbesserung der Strategie wäre durch eine Zuweisung der übermittelten singlets zu untätigen Clients möglich, um eine Störung der aktiven User zu vermeiden.

Verschiedene Änderungen der Strategie reduzieren die Menge an Kommunikationsverbindungen zum Server. Bei einem Fehlversuch im Cache kombiniert der User die Nachricht an den Server, indem das neueste „Inhaltsverzeichnis" seines Caches und die Anfrage über die gesuchte Datei zusammen in eine Mitteilung packt. Dabei beinhaltet das neue Verzeichnis, welche Datei der Client aus seinem Cache entfernt und wenn vorhanden, von wo die Datei ursprünglich übermittelt wurde.

Andere Veränderungen führen zu einer Reduzierung der Anfragen an den Server, ob eine Datei ein singlet ist oder nicht. Zunächst kann man davon ausgehen, daß jedes Dokument bei dem der Zähler gesetzt ist ein singlet ist und somit eine Anfrage an der Server unnötig ist. Für Dateien, deren Zähler nicht gesetzt ist muß gewöhnlich eine Nachricht an den Server gesendet werden. Sobald aber einmal ermittelt wurde, daß das Dokument einzigartig ist bedarf es keiner weiteren Nachfrage.

Der Vorteil der Strategie liegt im Austausch von lokalen Dateien der Clients und den gecachten Dateien für alle User des Systems. Durch die Begünstigung von singlets erhält man eine bessere Leistung als beim Greedy Forwarding Algorithmus, da das Entfernen von singlet sehr viel teurer ist, als das Entfernen von Duplikaten. Spätere Anfragen an ein solches Duplikat können durch andere Client Caches befriedigt werden. [DWA+94]

Der N-Chance Forwarding Algorithmus wird oft auch einfach als **N-Chance** Algorithmus bezeichnet.

Parameter:

- LRU-Strategie (für global verwalteten Cache)
- Ersetzungsalgorithmen für lokal verwaltete Caches
- Random-Strategie (Auswahl des neuen Clients für ein singlet)
- n = Anzahl der Weiterübermittlungen

Implementierungsmöglichkeiten:

- Wahl des Faktors n
- Wahl der Ersetzungsalgorithmen für lokal verwaltete Caches

37. daher der Name N-Chance

Ergebnisse aus Studien:

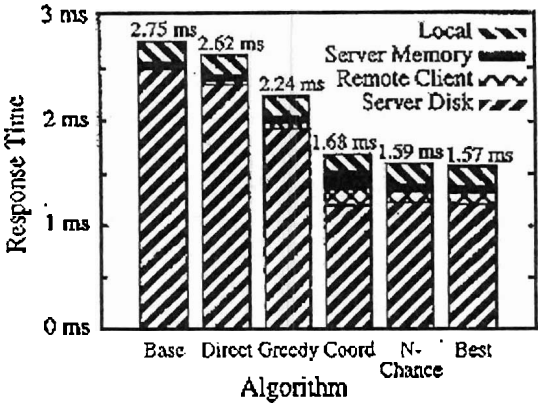


Abb. 47: Durchschnittliche Antwortzeiten der Algorithmen Base Case, Direct Client Cooperation, Greedy Forwarding, Centrally Coordinated Caching, N-Chance Forwarding und Best Case für eine Datei [DWA+94]

Base Case bedeutet hier, daß keine Cooperative Caching stattfindet; Best Case stellt eine optimale Cooperative Caching-Strategie dar. [DWA+94]

In den Abbildungen 50 und 51 wird der N-Chance Forwarding Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Beim N-Chance Forwarding Algorithmus ist positiv zu bewerten, daß die sogenannten singlets besonderen Schutz erhalten. Da das Laden von Filmen aus der Bibliothek sehr kosten- und zeitaufwendig ist, scheint die Idee Unikate nicht sofort zu löschen sehr sinnvoll. Sie werden stattdessen in einem global verwalteten Cache gespeichert, bis wirklich klar ist, daß die jeweiligen Filme für einen längeren Zeitraum nicht mehr benötigt werden.

Durch die dynamische Einteilung eines Caches in Bezug auf seinem global/lokal-Speicherverhältnis kann sich der jeweilige Server an die aktuellen Bedürfnisse anpassen. Wird ein Server kaum benötigt, so wird der globale Teil entsprechend größer, benötigt ein Server dagegen selber seinen Speicherraum, so nimmt der lokal verwaltete Teil zu.

Nachteilig wirkt sich diese Dynamik auf das Verschicken von singlets aus. Da die Auswahl des globalen Caches, in den das singlet gespeichert werden soll, nach dem Random-Prinzip erfolgt, wird häufig der Fall eintreten, daß der Film aus Platzmangel bald wieder weitergeschickt werden muß. Erreicht nämlich das singlet einen Server mit einem sehr kleinen globalen Cache, so erreicht er auch sehr schnell das Ende der internen LRU-Liste. Aufgrund dieses Problems kann man mittels eines Parameters n die Anzahl der Chancen für einen Video einstellen. Allerdings muß noch geklärt werden, welchen Wert n bei einem Einsatz der N-Chance Forwarding-Strategie in VoD-Systemen erhalten soll.

Ein weiterer Kritikpunkt liegt, wie schon beim Centrally Coordinated Caching Algorithmus, in der Verwendung der LRU-Strategie für die Verwaltung des globalen Caches. Außerdem wird bei diesem Algorithmus der Datenverkehr zwischen „Hauptserver" und den anderen Servern noch größer, da bei jedem Film der ausgewiesen werden soll zuvor ermittelt werden muß, ob es sich um ein singlet handelt.

Des weiteren verringert sich durch die Implementierung von "zentralen" Servern die Robustheit des Systems.

6.3.6 Weighted LRU

Beschreibung:

Hier handelt es sich um einen Algorithmus, der seine Ersetzungsentscheidung aufgrund des Nutzen/ Kosten-Verhältnisses einer Datei trifft. Dokumente von denen mehrere Kopien in verschiedenen Client Caches gespeichert sind, haben dementsprechend einen geringen Wert, da die Dateien jederzeit von einem anderen User geholt werden können. Dagegen sind einmalige Dateien besonders wertvoll, da ein Löschen dieser Dateien bedeutet, daß die nächsten Anfrage an ein solches Dokument von der Festplatte des Servers befriedigt werden muß. Dies ist gleichbedeutend mit einem höheren Kosten- und Zeitaufwand, als die Erfüllung einer Anfrage durch einen anderen Client Cache.

Der Weighted LRU versucht somit einen Interessensausgleich zwischen den häufig frequentierten Duplikaten (um den Datenverkehr im Netzwerk zu entlasten) und den wenig genutzten singlets (um Anfragen an die Festplatte zu vermeiden) herzustellen. Trotzdem sind die Antwortzeiten schlechter, als beim einfacher zu implementierenden N-Chance Algorithmus[38].

Natürlich besteht auch hier wieder die Einteilung zwischen lokal und global verwalteten Cache. Ebenso wird in der Regel der LRU Algorithmus zur Verwaltung des globalen Caches verwendet. [DWA+94]

Parameter:

- LRU-Strategie (für global verwalteten Cache)
- Ersetzungsalgorithmen für lokal verwaltete Caches

Implementierungsmöglichkeiten:

- Wahl der Ersetzungsalgorithmen für lokal verwaltete Caches

Bewertung:

Der Weighted LRU versucht die Kosten und den Nutzen, den ein Film verursacht mit den entsprechenden Werten der anderen Filme zu vergleichen. Problematisch ist allerdings, was man unter Nutzen und Kosten versteht und wie sich diese berechnen. Der Implementierungsaufwand scheint hier viel größer, als beim N-Chance Forwarding Algorithmus und trotzdem liefert der Weighted LRU keine besseren Ergebnisse. Deshalb ist er auch für Videoverteilungssysteme kaum geeignet.

6.3.7 GMS (Global Memory Service Algorithm)

Beschreibung:

Der GMS Algorithmus ist der N-Chance-Strategie sehr ähnlich. Auch hier wird das Löschen von singlets vermieden, allerdings weiß man zu jeder Zeit, ob es sich bei einer bestimmten Datei um ein singlet handelt oder nicht. Ein übergeordneter Manager verfolgt die Anzahl der Kopien einer Datei, die sich im System befinden und teilt dem entsprechenden Client mit, wenn eine Dokument zu einem singlet wird. Eine Serveranfrage seitens der User entfällt also. Genau wie beim N-Chance Algorithmus besteht auch hier eine dynamische Einteilung der Client Caches in einen lokalen und einen globalen Teil.

---

38. Simulationsergebnisse auf Seite 5 [DWA+94]

Das Verhältnis von lokalem zu globalem Segment wird durch Anfragen an nicht-vorhandene Dateien (im eigenen lokalen Teil) verändert. Host[39] P führt bei einem Miss die folgenden globalen Ersetzungsalgorithmen aus, die hier durch vier mögliche Fälle beschrieben werden.

- Fall 1: Die gesuchte Datei befindet sich im globalen Cache eines anderen Hosts, Q. Die Datei wird nun mit irgendeinem Dokument aus Ps globalen Cache getauscht[40]. Befindet sich die Datei nun beim Host P, so wird die gesuchte Datei in den lokalen Teil von P gespeichert. Damit wächst die Größe des lokalen Caches von P um 1. Qs lokal/global-Speicherverhältnis bleibt unverändert. Dies wird auch in der Abbildung 48 dargestellt.

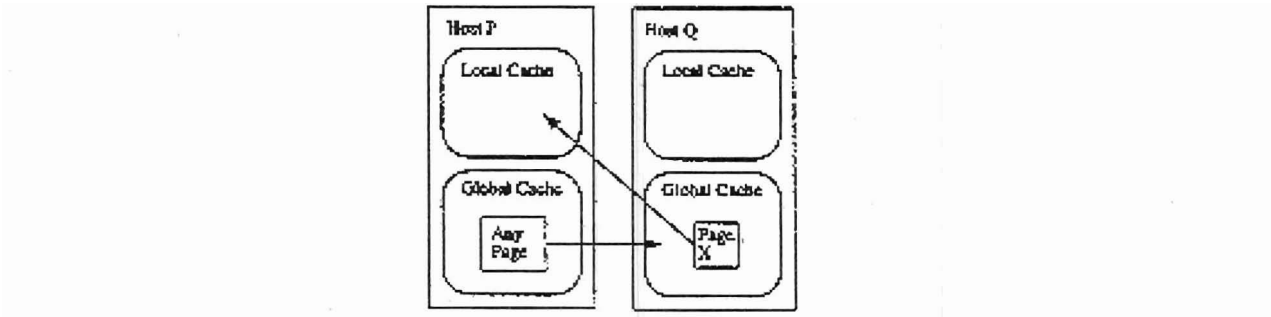


Abb. 48: Globale Ersetzung bei einem Treffer im globalen Cache [FMP+95]

- Fall 2: Die gesuchte Datei befindet sich im globalen Cache des Hosts Q, aber Ps Speicher enthält ausschließlich lokale Blöcke. Dann wird ein lokales Dokument von P (ausgewählt nach dem verwendeten Ersetzungsalgorithmus) mit der gesuchten Datei von Q ausgetauscht. Die Größe des globalen Caches von Q und des lokalen Speichers von P bleibt unverändert.
- Fall 3: Die gesuchte Datei befindet sich auf der Festplatte des Servers. Diese wird in den Speicher von P geladen und dort als lokale Datei behandelt. Um für das neue Dokument Platz zu schaffen wird die älteste Datei des gesamten Systems gelöscht (hier: eine globale Datei aus Host Q) und irgendeine globale Datei von P in den Host Q geschoben. Die Abbildung 49 verdeutlicht die Vorgehensweise noch.

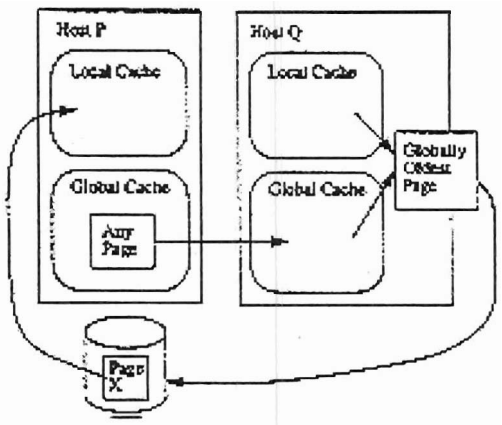


Abb. 49: Globale Ersetzung bei einem Miss im globalen Cache [FMP+95]

39. Hier wird der Begriff Host verwendet, man kann aber auch weiterhin von einem Client (mit einem Cache) sprechen.
40. Random-Strategie

- Fall 4: Die gesuchte Datei befindet sich im lokalen Cache eines anderen Hosts Q. Zunächst wird die Datei kopiert und in den Host P geschrieben, d.h. das Original verbleibt im lokalen Teil von Host Q. Um für das neue Dokument Platz zu schaffen, wird die älteste Datei des gesamten Systems gelöscht (z.B. wieder ein globales Dokument aus Host Q) und irgendeine globale Datei von P in den Host Q geschoben.

Bei diesem Algorithmus besteht leider ein Problem. Im Fall 3 und 4 müssen Dateien gelöscht werden, um Platz für neue Dokumente zu schaffen. Als Auswahlkriterium verwendet man dabei „einfach" die vergangene Zeit seit der letzten Anfrage an eine Datei[41]. Je größer das vom Manager verwaltete System ist, desto schwieriger wird es sein, die jeweils älteste Datei zu bestimmen. Deshalb wird beim GMS nur ein Näherungsverfahren angewendet, um die wahrscheinlich älteste Datei herauszufinden.

Dazu werden zunächst sogenannte „epochs" (Epochen) eingeführt. Jede Epoche hat eine maximale Dauer von T und eine maximale Anzahl an Ersetzungen M innerhalb des Systems. Die Werte von T und M variieren von Epoche zu Epoche, abhängig vom Nutzerverhalten und Speicherinhalt. Eine neue Epoche beginnt jeweils dann, wenn die Dauer T überschritten ist, M Dateien ersetzt wurden oder entdeckt wird, daß die Informationen über das Alter der Dokumente ungenau geworden sind. Im allgemeinen dauert eine Epoche 5-10 Sekunden.

Das System enthält nun Informationen über das Alter der lokalen und globalen Dateien für jeden Client. Am Anfang einer Epoche sendet jeder User die Summe des Alters seiner gesamten Dateien an einen festgelegten initiator node. Aufgrund dieser Informationen kann der initiator node nun ein Gewicht w_i für jeden Client i berechnen. Zusätzlich bestimmt der initiator node noch ein Mindesalter, MinAge. Alle Dokumente die dieses Mindestalter haben werden in der nächsten Epoche ersetzt. Der initiator node sendet die Werte von w_i und den Wert von MinAge zu allen Clients im System. Außerdem wird der User mit den meisten ungenutzten Dateien (größtes w_i) als neuer initiator node für die nächste Epoche definiert.

Während einer Epoche soll z.B. eine Datei aus Host P entfernt werden (Fall 3 und 4). Als erstes muß geprüft werden, ob das dafür ausgewählte Dokument älter als MinAge ist. Ist dies der Fall, so wird sie einfach gelöscht. Ist dem aber nicht so, dann sendet P die Datei zu dem Host, welches den größten Wert w_i hat. In diesem Fall wird die von P ausgewiesene Datei als globales Dokument in den ausgewählten Host geschrieben und die älteste Datei des Hosts wird gelöscht.

Möchte ein Client ein singlet ausweisen, so wird diese Datei nicht an einen zufällig ausgewählten Client geschickt (wie beim N-Chance Algorithmus), sondern auf der Basis des oben beschriebenen Prinzips (größtes w_i) ein geeigneter Client ermittelt. Dabei wird das singlet in den global verwalteten Teil des Clients gespeichert und per LRU-Prinzip verwaltet. Im Gegensatz zum N-Chance erhält ein singlet keine zweite Chance, ist aber durch die bessere Client-Auswahl ausreichend geschützt.[SH96] [FMP+95]

Parameter:

- Ersetzungsalgorithmen für lokal verwaltete Caches
- Random-Strategie (zum Tausch von Dateien)
- MinAge = Mindestalter einer Datei
- T = Maximale Dauer einer Epoche
- M = **Maximale Anzahl an Ersetzungen** innerhalb einer Epoche
- w_i = Gewicht für jeden Client i [42]

41. Anwendung der LRU-Strategie
42. Berechnet aufgrund der Altersdaten der im jeweiligen Client i enthaltenen Dateien.

Implementierungsmöglichkeiten:

- Wahl der Ersetzungsalgorithmen für lokal verwaltete Caches
- Wahl der maximalen Dauer T einer Epoche
- Wahl der maximalen Anzahl an Ersetzungen M innerhalb einer Epoche

Ergebnisse aus Studien:

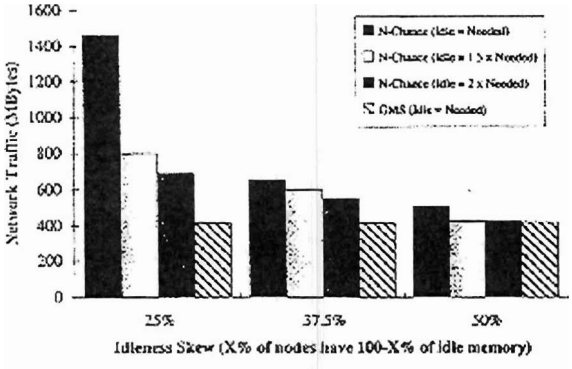


Abb. 50: Netzwerkdatenverkehr bei unterschiedlicher Verteilung des ungenützten Speicherraumes [FMP+95]

In der Abbildung 50 mißt die vertikale Achse den Datenverkehr im Netz und die horizontalen Achse die Verteilung des ungenutzten Speicherraumes. Die aufgetragenen Werte sind dabei wie folgt zu verstehen: (1) 25% der Clients enthalten genau 75% des ungenutzten Speicherraumes, (2) 37,5% der Clients besitzen 62,5% des freien Speichers, und (3) 50% der Clients halten 50% des freien Speicherplatzes. Dadurch ergibt sich im Fall (1) eine schräge Verteilung, die den größten Anteil an freiem Speicherplatz einer kleinen Anzahl von Clients zuordnet, während im Fall (3) der ungenutzte Speicherraum gleichmäßig verteilt ist.

Als vergleichende Strategie zum GMS wird hier der N-Chance Algorithmus verwendet. Der bei dieser Strategie verwendete Parameter n erhält hier den Wert zwei. Die Angaben in den Klammer bei den simulierten Algorithmen haben folgende Bewandtnis: Idle = Needed bedeutet, daß das System genau soviel ungenutzte Dateien enthält, wie benötigt werden, während Idle = 2 x Needed bedeutet, daß doppelt soviel ungenutzte Dokumente im System vorkommen, wie eigentlich benötigt werden. Der GMS Algorithmus wird unter der Bedingung Idle = Needed simuliert.

Anhand der Abbildung 50 läßt sich nun erkennen, daß für (1) der GMS Algorithmus weniger als 1/3 des Datenverkehrs vom N-Chance verursacht. Und auch bei (2) ist der GMS immer noch um 50% besser, als die N-Chance-Strategie. Für diese Überlegenheit von GMS gibt es verschiedene Gründe. Erstens wählt der N-Chance Algorithmus den neuen Client für ein ausgewiesenes singlet nach dem Random-Prinzip aus, was natürlich selten zu einem optimalen Ergebnis führt. In vielen Fällen wird es nötig sein innerhalb kürzester Zeit einen neuen Client zu suchen. Dies führt dann wieder zu einem erneuten Datenverkehr zwischen zwei Clients. Zweitens müssen die Clients für jedes zur Ausweisung vorgesehenes Dokument beim Server nachfragen, ob es sich dabei um ein singlet handelt. Beim GMS dagegen teilt der Server von sich aus mit, wenn eine Datei zu einem singlet wird. Damit läßt sich der Datenverkehr schon um einiges reduzieren. [FMP+95]

In der Abbildung 51 wird der GMS Algorithmus noch mit anderen Algorithmen verglichen.

Bewertung:

Wie beim N-Chance Forwarding Algorithmus werden auch hier singlets besonders geschützt. Vorteilhaft ist weiterhin, daß ein Server keine Anfrage an den übergeordneten Server schicken muß, um zu

erfahren, ob sich bei einem Film um ein Unikat handelt oder nicht. Stattdessen informiert der „Hauptserver“ den entsprechenden Server von sich aus. Dadurch wird der Datenverkehr gegenüber N-Chance reduziert. Leider besteht auch hier das Robustheits-Problem durch die Implementierung von "zentralen" Servern.

Ebenfalls wie bei der N-Chance-Strategie läßt sich das global/lokal-Speicherverhältnis den jeweiligen Gegebenheiten anpassen. Da der Server, in den das ausgewiesene singlet gespeichert werden soll, diesmal nicht nach dem Random-Prinzip ausgesucht wird, ist die oben erwähnte Dynamik hier nicht von Nachteil. Ob allerdings die alternativ verwendete Methode (Näherungsweise LRU über alle globale Caches) für Videos optimal ist, bleibt fraglich. Man muß sich aber auch überlegen, ob ein komplizierterer Algorithmus überhaupt implementierbar und sinnvoll ist.

Will man das Prinzip des Cooperative Caching in einem Video-on-Demand-System implementieren, so ist dafür der GMS Algorithmus einer der geeignetsten Cooperative Caching Algorithmen.

6.3.8 Hint-based Algorithm

Beschreibung:

Bei den bisher beschriebenen Algorithmen wurden die Entscheidungen, welche Datei nun entfernt werden muß, wohin eine ausgewiesen Datei verschoben werden soll usw. mittels einer zentralen Stelle (Manager) getroffen. Dieser Manager wurde also häufig frequentiert und damit stark belastet. Außerdem kam es durch die ständige Kommunikation zwischen dem zentralen Manager und den Client zu Verzögerungen, bis die Anfragen erfüllt werden konnten. Mit dem Hint-based Algorithmus soll nun die Einschaltung eines Managers soweit als möglich vermieden werden, um weitere Verzögerungen zu vermeiden.

Um dies zu verwirklichen wird hier unter anderem das Prinzip des Hint Maintenance eingeführt. Die Hints[43] müssen dabei genau und korrekt registriert werden. Dabei hilft das Konzept des master copy. Die erste Kopie einer Datei, die von irgendeinem Cache gespeichert wird, erhält den Status des master copy. Gibt der Client diese Datei irgendwann weiter, so merken sich beide Clients den neuen Aufenthaltsort des master copy. Das heißt, daß die Hints den wahrscheinlichen Aufenthaltsort des master copy enthalten. Um die einzelnen User durch die zusätzliche Speicherung der Hints nicht zu überlasten, wird das Verschieben anderer Kopien[44] völlig ignoriert.

Als weiteres Prinzip wird beim Hint-based Algorithmus ein Lookup[45] Mechanism kreiert. Dieser funktioniert wie folgt:

1. Tritt bei einem Client nun ein Miss auf, so konsultiert der Client zunächst einmal seine Hint Informationen für diese Datei.
2. Enthält der Hint den wahrscheinlichen Aufenthaltsort, so wird eine Anfrage an den entsprechenden Client gesendet. Ist dies nicht der Fall muß die Anfrage direkt an den Server übermittelt werden.
3. Der Client, der die weitergereicht Anfrage erhalten hat konsultiert nun seinerseits seine Hint Informationen über diese gesuchte Datei und fährt dann mit dem Punkt 2 fort.

Die Idee, die hinter diesem Prinzip steckt ist denkbar einfach. Statt eines übergeordneten allwissenden Managers sollen die einzelnen Clients den Aufenthaltsort der einmal erhaltenen Datei weiterverfolgen. Diese Informationen helfen dann, eine gesuchte Datei wieder zu finden.

Wie bei vielen anderen Cooperative Caching Algorithmen, wird auch bei dieser Strategie zwischen einem lokalen und einem globalen Cache unterschieden. Wird nun eine Datei aus dem lokalen Cache

43. Hint = Hinweis
44. zweite, dritte usw. Kopie
45. Lookup = Nachschlagen

eines Client ausgewiesen, so entscheidet der Cooperative Caching Algorithmus, ob die Datei in den globalen Cache gespeichert wird. Grundsätzlich werden diejenigen Dateien in den globalen Speicher geschrieben, die im System nur einmal vorkommen. Ob dies bei einer Datei der Fall ist, mußte bisher immer der zentrale Manager entscheiden. Bei diesem neuen Algorithmus ist die Entscheidung viel einfacher. Nur master copies werden in den globalen Cache geschrieben, während alle anderen ausgewiesenen Dateien einfach gelöscht werden. Dadurch entfällt für dieses Problem jegliche Anfrage zwischen Clients und Manager.

Ein möglicher Nachteil bei dieser Vorgehensweise liegt darin, daß der Hint-based Algorithmus immer die erste, also die master copy in den globalen Speicher übermittelt. Dies kann in einigen Fällen zu unnötigen Sendungen führen, da es passieren kann, daß die master copy schon in den globalen Cache gespeichert wird, bevor alle anderen Kopie entfernt wurden.

Soll nun eine Datei in den globalen Speicher übermittelt werden, so muß man sich überlegen, welcher der Clients das beste Ziel abgibt. Da jeder Cache eines Users aus einem lokalen und einem globalen Teil besteht, muß aus einem der beiden Segmente eine andere Datei gelöscht werden, um den Platz für die neue Datei zu schaffen. In diesem Fall kann jetzt natürlich auch eine master copy aus dem globalen Cache gelöscht werden. Bei den bisherigen Algorithmen wurde der Ziel-Client vom Manager ausgewählt. Dieser Manager bediente sich dabei meist der Random-Strategie, die natürlich nicht zu einem optimalen Ergebnis führt. Beim Hints-based Algorithmus wird deshalb anders vorgegangen. Jeder Client stellt für sich eine Liste der ältesten Dateien auf, welche das Alter der Dateien enthält, von denen der Client glaubt, daß sie die ältesten der jeweilig anderen Clients sind. Die entsprechenden Informationen erhalten die Clients beim Austausch von Dateien. Sobald nämlich ein solcher Austausch stattfindet, wird auch gleichzeitig mitgeteilt, wie alt die älteste Datei ist, die man zur Zeit im eigenen Cache gespeichert hat. In den Listen der beiden Clients werden dann die entsprechenden Updates vorgenommen. Eine ausgewiesene lokale Datei eines Clients wird also nun in denjenigen globalen Speicher eines anderen Clients geschoben, von dem der erste Client annimmt, daß er die älteste Datei beherbergt. Die soeben beschriebene Vorgehensweise wird als best-guess Ersetzung bezeichnet.

Leider kann es hier auch zu einigen Problemen kommen. Wenn mehrere Clients von einem anderen Client glauben, daß dieser die älteste Datei enthält, so werden alle diese Clients ihre Dateien an den selben Cache schicken. Dies führt dann natürlich zu einer Überlastung des Ziel-Clients. Glücklicherweise ist diese Situation ziemlich unwahrscheinlich, da durch die Dynamik des Systems die einzelnen Clients auch meist unterschiedliche Altersangaben über die Dateien erhalten.

Ein großer Nachteil der best-guess Ersetzung liegt in einer möglichen falschen Entscheidung. Es kann passieren, daß eine Datei zu einem Client gesendet wird, der die älteste Datei überhaupt nicht gespeichert hat. Oder die zu ersetzende Datei kann jünger sein, als die „neue" Datei. Um solche Fehler auszugleichen entwickelte sich die Idee eines Discard Caches. Eine einfache Möglichkeit zu entscheiden, ob eine Fehlentscheidung vorliegt oder nicht wird in der Tabelle 14 gezeigt.

Type of block	Action
Non-master copy	Discard
Old master copy	Discard
Young master copy	Send to discard cache

Tab. 14: Die Regeln eines Discard Cache [SH96]

Es ist offensichtlich, daß non-master copies immer gelöscht werden, weil nur die master copies in den globalen Cache gespeichert werden. Ein Fehler liegt also erst dann vor, wenn die Datei für eine Ersetzung zu jung ist. Ist dies der Fall, so wird die ausgewählte zu ersetzende Datei nicht gelöscht, sondern in den Discard Cache geschoben. Die im Discard Cache befindlichen Dateien werden nach dem LRU-

Prinzip verwaltet. Somit dient der Discard Cache als Zwischenspeicher, um potentielle Ersetzungsfehler auszugleichen. [SH96]

Parameter:

- LRU-Strategie
- Ersetzungsalgorithmen für lokal verwaltete Caches

Implementierungsmöglichkeiten:

- Wahl der Ersetzungsalgorithmen für lokal verwaltete Caches

Ergebnisse aus Studien:

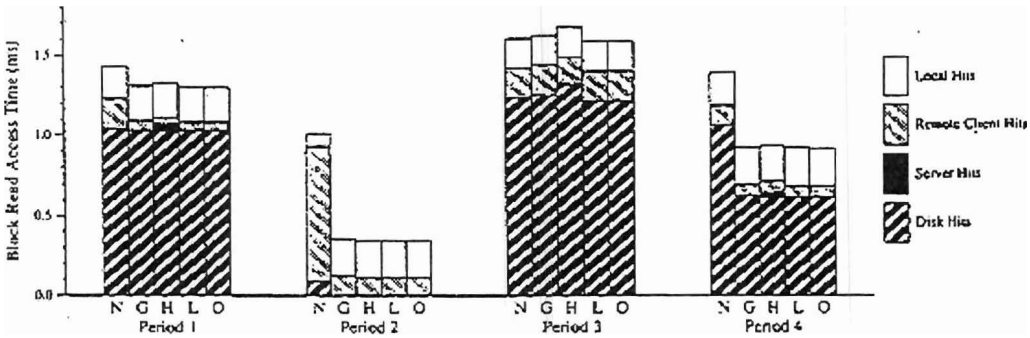


Abb. 51: Vergleich der durchschnittlichen Zugriffszeiten einer Datei der Algorithmen N-Chance (N), GMS (G), Hint-based (H), Global LRU (L) und Optimal (O) [SH96]

Der Optimale[46] und der **Global LRU Algorithmus** stellen zwei ideale Cooperative Caching Strategien dar. Beide bilden eine obere Bewertungsgrenze für den Leistungsvergleich mit anderen Algorithmen. Die zwei erwähnten idealen Strategien unterscheiden sich nur in ihrer internen Ersetzungsstrategie. Der Optimale Algorithmus ersetzt die Datei, deren nächste Anfrage in der am weitesten entfernten Zukunft liegt. Natürlich ist diese Strategie nicht realisierbar, da sie hellseherische Fähigkeiten verlangt. Der Global LRU Algorithmus ersetzt die älteste Datei, die sich im System befindet. Damit nähert sie sich der Optimal-Strategie an und ist dennoch theoretisch realisierbar. Allerdings würde das Auffinden der jeweils ältesten Datei eines Systems einen zu hohen Kosten- und Zeitaufwand verursachen und wird deshalb in der Praxis nicht realisiert.

Die durchschnittlichen Zugriffszeiten für GMS und den Hint-based Algorithmus liegen sehr nahe am idealen Cooperative Caching Algorithmus. Die Leistung der N-Chance-Strategie weicht in der zweiten und vierten Periode stark von den anderen ab. Der Grund liegt in der erhöhten Anfragen nach Dateien innhalb dieser beiden Zeitabschnitte. Dieser Anstieg wirkt sich auf den N-Chance Algorithmus besonders stark aus, da hier noch das Random-Prinzip als Ersetzungsentscheidung angewendet wird. Mit erhöhter Dateienanfrage nimmt auch die Wahrscheinlichkeit zu, daß ein zufällig ausgewähltes Ersetzungsopfer bald wieder von dem Client, der diese Datei zuvor gespeichert hatte benötigt wird. Ist dies der Fall muß sich der Client die Datei von anderen (weiter entfernt liegenden) Clients besorgen, was natürlich zu einer Erhöhung der Zugriffszeiten führt. [SH96]

46. Siehe auch OPT Algorithmus auf Seite 3.

Bewertung:

Der Hauptvorteil liegt beim Hint-based Algorithmus in der Entlastung des übergeordneten Servers. Dies wird dadurch erreicht, indem die Informationen über den Verbleib eines Filmes in allen Servern gespeichert ist und nicht nur im „Hauptserver". Dadurch erhält man noch einen weiteren Nebeneffekt. Sollte der übergeordnete Server einmal ausfallen, so funktioniert der Algorithmus immer noch. Der übergeordnete Server dient hier nur noch als Verbindungsglied zwischen den anderen Servern und der Zentral-Bibliothek. Diese Aufgabe kann jedoch auch jeder andere Server wahrnehmen. Das System gewinnt dadurch an Robustheit.

Als Strategie zur Auswahl eines neuen globalen Caches für eine master copy, wird auch hier eine Näherung zum LRU-Prinzip verwendet. Wie schon beim GMS geschrieben wurde, stellt der LRU Algorithmus nicht gerade die bestmögliche Strategie in Videoverteilungssysteme dar, aber andere Verfahren werden schwer zu implementieren sein.

Neben dem GMS stellt auch der Hint-based Algorithmus einen geeigneten Cooperative Caching Algorithmus für den Einsatz in Videoverteilungssysteme dar. Dabei bestehen zusätzlich noch die oben erwähnten Vorteile.

7 Tabellarische Übersicht

7.1 Erklärungen zur Tabelle

Alter der eingelagerten Dateien:

Die Dateien werden in einer bestimmten Reihenfolge in den Speicher geladen. Eine Strategie könnte nun sein, jeweils die älteste Datei zu ersetzen (FIFO). Dies hätte aber zur Folge, daß häufig benutzte Dateien ständig ein- und wieder ausgeladen werden müssen. Die Fehlerquoute wird dadurch sehr hoch. Von Vorteil ist diese Methode bei sequentiellem Anfrageverhalten (z.B.Stapeldateien).

Werden im Cache nur einzelne Blöcke eines Filmes gespeichert, so könnte man als interne Ersetzungsstrategie sehr gut nach dem Alter der eingelagerten Blöcke vorgehen. Gerade bei Videos muß die Abfolge von gespeicherten Bildern strikt eingehalten werden.

Anzahl der Anfragen:

Dies ist eine der wichtigsten Kennzahlen für Ersetzungsalgorithmen. Man geht dabei von der allgemeinen Beobachtung aus, daß Dokumente die lange nicht mehr aufgerufen wurden auch in nächster Zeit nicht mehr aufgerufen werden. Umgekehrt bleiben Files wahrscheinlich eine längere Zeit unbenutzt, wenn sie zuvor für eine längere Zeit nicht verwendet wurden. Im Normalfall würde man also Dokumente, die in der letzten Zeit häufig frequentiert wurden auf jedem Fall im Speicher behalten.

In der Anwendung von Video-on-Demand hieße das, daß z.B. die Top-Ten der Videofilme auf jeden Fall im Cache vorhanden sein müssen, ebenso wie Filme zu aktuellen und vielgefragten Themen (z.B. Historische Kriminalfilme, Titanic-Filme etc.). Allgemein muß die Popularität eines Filmes geprüft werden.

Alterung:

Alterung bedeutet hier, daß die Anfragen unterschiedlich gewichtet werden. Neue Anfragen erhalten damit eine höhere Gewichtung als ältere Anfragen. Ganz weit zurückliegende Referenzen werden häufig gar nicht mehr berücksichtigt. Mit dieser Maßnahme verhindert man, daß Dokumente, die in der Vergangenheit häufig nachgefragt wurden, zur aktuellen Zeit aber nicht mehr benötigt werden, quasi unlöschbar werden. Außerdem wird durch dieses Vorgehen besser auf das aktuelle Anfragemuster eingegangen.

Gerade in Videoverteilungssystemen ist die Ermittlung des aktuelle Anfragmusters der Nutzer sehr wichtig, denn nur so läßt sich der Cache-Inhalt wirklich an den Kundenwünschen anpassen. Außerdem wäre es für den Anbieter sehr unvorteilhaft, wenn Filme, die mittlerweile nicht mehr gefragt sind trotzdem im Cache gehalten werden, nur weil sie vor kurzer Zeit noch äußerst populär waren.

Zeit seit der letzten Anfrage:

Die Zeit seit der letzten Anfrage steht im starken Zusammenhang mit der gesamten Zahl der Anfragen für ein Dokument. Die Erklärung wurde schon im obigen Abschnitt dargestellt.

Dieser Faktor ist bei Videoverteilungssystemen nicht sehr aussagekräftig. Man könnte ihn aber gut als Sekundärstrategie verwenden, wenn man mehr als einen Film mit dem gleichen Primärwert hat. In solch einem Fall wird dann der Film ausgewiesen, dessen letzte Anfrage am weitesten zurückliegt.

Einheitliche Blockgröße:

Einheitliche Blockgrößen werden bei Seitenersetzungsalgorithmen verwendet. Der Speicher ist dabei in eine bestimmte Anzahl von Seitenrahmen eingeteilt, die alle die gleiche Größe haben und den jeweiligen Prozessen zugeordnet werden.

Will man Filme als eine vollständige Einheit übertragen, so müßte ein Rahmen mindestens so groß sein, wie der Platzbedarf des größten Films. Dies führt aber zu enormen Speicherplatzverschwendungen bei allen anderen Filmen. Eine Alternative liegt in der blockweisen Übertragung eines Filmes. Dies hat jedoch einige andere Nachteile und ist nur in Ausnahmefällen[47] sinnvoll.

Dateigröße:

Da die verfügbare Speichergröße eines Caches begrenzt ist, kann es passieren, daß er durch wenige sehr große Dateien belegt wird. Anstelle einer großen Datei, könnten aber viele kleinere Dokumente gecacht werden. Außerdem ist die Wahrscheinlichkeit, daß eine der vielen kleinen Dateien aufgerufen wird viel größer als die, daß eine große wiederverwendet wird. Auf jeden Fall steigt die Cache Hit Rate und es profitieren mehr Benutzer vom Cache, wenn viele kleine Dateien gespeichert wurden. Legt man diese Betrachtung zu Grunde, ist es sicherlich sinnvoll beim Cache Replacement Algorithmus zuerst die größten Dateien zu löschen.

Für die Anwendung in der Videoverteilung kann man allerdings sagen, daß die Größe der einzelnen Objekte nicht so relevant ist, da Videodateien alle sehr groß sind. Wird der Film gar in Blöcken zerlegt übertragen, so sind alle Blöcke gleich groß und somit eine Auswahl aufgrund der Datengröße unmöglich. Bei einer Sekundärstrategie könnte man allerdings sehr gut die Größe eines Filmes als Auswahlkriterium anwenden.

Kosten der Beschaffung / Erhaltung:

Hier vergleicht man die Kosten für eine Neubeschaffung einer Datei mit den Kosten, die bei einem Cache-Miss entstehenden. Dieser Vergleich lohnt sich vor allem im Internet, wo sehr starke Kostenschwankungen für die Beschaffung von Files bestehen.

Bei Videofilmen könnte man prüfen, ob es rentabler ist den Film in einem direkt übergeordnetem Cache-Server zu belassen oder ihn doch in den anfordernden Cache zu laden. Problematisch ist hier allerdings die Definition der Kosten. Sie beinhalten neben den Leihgebühren, Mietgebühren für Übertragungsleitungen und Kosten für Wartezeiten auch die Kosten für Ladezeit und Bandbreitenbedarf, die in der Tabelle auch noch extra aufgeführt werden.

---

47. siehe Kapitel 3.3

Ladezeit:

Ein weiterer Aspekt sollte nicht unberücksichtigt bleiben. Nicht jede Datei kann gleich schnell beschafft werden. Somit wäre es sinnvoll die Ladezeit zu berücksichtigen, die sich aus der Verbindungsherstellungszeit und der Übertragungszeit zusammensetzt. Hier erscheint es sinnvoll diejenigen Seiten zu löschen, die sehr schnell wieder beschafft werden können.

Da das Laden von Videofilmen sehr lange dauert (vor allem wenn er direkt von der Zentrale geladen wird), ist die Ladezeit eines Filmes ein sehr wichtiges Kriterium bei der Auswahl des zu ersetzenden Videos. Die Ladezeit sollte auf jeden Fall in der „Optimalstrategie" für VoD-Systeme berücksichtigt werden.

Bandbreite:

Jedes Dokument benötigt für seine Übertragung eine bestimmte Bandbreite (Bytes pro Zeiteinheit). Da in einem System leider sehr unterschiedliche Bandbreiten vorhanden sind, kann es an manchen Stellen zu Engpässen kommen. Deshalb müssen Dokumente mit einem sehr hohen Bandbreitenbedarf besonders geschützt werden, da sich eine Wiederbeschaffung unter Umständen als sehr schwierig erweisen kann. Ebenso zu schützen sind Objekte, die von einem Server kommen, dessen Verbindungswege zu anderen Servern eine geringe Bandbreite aufweisen.

Wenn mehrer Filme gleichzeitig übertragen werden sollen, so stellt die Bandbreite einen entscheidenden Faktor dar. Es kann dann passieren, daß aufgrund zu kleiner Bandbreite nicht alle Server beliefert werden können, sondern nur die ersten paar. Ebenso bei der Übertragung mehrerer Filme vom Server zu den verschiedenen Clients. Je niedriger also die Bandbreite einer Übertragungsleitung ist, desto größer ist seine Blockierwahrscheinlichkeit.

Intervallbereich:

Man kann Anfragehäufigkeiten über die gesamte Zeit messen oder aber nur in bestimmten Intervallen (z.B. sieben Tage). Durch Einführung dieser Intervalle ist es möglich, stark gehäufte Referenzen, die schon sehr lange zurückliegen, nicht mehr in die Vorhersage des Anfrageverhaltens in naher Zukunft eingehen zu lassen. Dadurch lassen sich neue Verhaltensmuster des Nutzers besser erkennen.

Da Intervallbereiche sehr geeignet sind, um die Anfragen in einem bestimmten Zeitraum zu erfassen, sind sie für Videoverteilungssystemen überaus vorteilhaft. Die Anfragen für die fünfte Woche sind für den Server-Betreiber schließlich weniger wichtig, als die Anfragen der ersten Woche. Durch die Verwendung von Intervallen läßt sich die aktuelle Popularität eines Filmes viel besser ermitteln und somit auch das zukünftige Anfragemuster besser abschätzen.

Speicherreinigung:

Unter Speicherreinigung versteht man die Änderung des gesamten Cache-Inhaltes, in gewissen Zeitabständen. Üblicherweise verwendet man dabei einen 7-Tage-Rhythmus.

Bei VoD-Systemen wäre es denkbar, den Inhalt des Speichers zweimal am Tag neu zusammenzustellen, um so den Cache-Inhalt dem unterschiedlichen Nachfrageverhalten zu Tages- und Nachtzeiten anzupassen. Problematisch ist allerdings, daß diese Neuzusammenstellung eine sehr zeit- und kostenaufwendige Angelegenheit ist. Die Übertragung der Filme dauert recht lange und blockiert natürlich die Leitungskapazitäten. Der Übergang sollte möglichst gleitend vonstatten gehen und zu Zeiten geringer Anfragen vorgenommen werden.

Prioritäten:

Es besteht die Möglichkeit einzelnen Prozessen, Seiten, Files usw. Prioritäten zuzuordnen, um für den Nutzer wichtige Dokumente vor Ersetzungen zu schützen. .

In VoD-Systemen könnte der Anbieter z.B. die zehn Filme, die in der letzten Woche am häufigsten nachgefragt wurden, für die kommende Woche mit einer hohen Priorität belegen, um sie vor einer Ersetzung zu schützen. Man könnte außerdem Filmtypen, die nur selten nachgefragt werden (z.B. Heimatfilme, Stummfilme usw.) mit einer niedrigen Priorität belegen, um sie bei Speicherbedarf als erstes zu entfernen.

Kooperation:

Damit wird die Kooperation (Cooperative Caching) von Servern verstanden, d.h. der Inhalt der Server Caches wird koordiniert und die Anfragen eines Clients können aus allen in der Umgebung liegenden Servern befriedigt werden.

Die Einführung von Cooperative Caching scheint in Videoverteilungssystemen sehr sinnvoll zu sein. Schließlich können dadurch die viele Anfragen von den naheliegenden Servern erfüllt werden, was natürlich zeit- und kostengünstiger ist als eine Übertragung von der Bibliothek. In der Tabelle wird außerdem noch zwischen zentraler und dezentraler Koordination unterschieden. Zentral bedeutet, daß ein "übergeordneter" Server Managementaufgaben übernimmt und ein Ausfall dieses zentralen Servers zu erheblichen Problemen im System führt. Bei dezentral funktionierende Algorithmen dagegen, kann der "übergeordnete" Server leicht von einem anderen Server ersetzt werden, weil er nur noch als "Übermittlungsserver" dient.

AO/CO:

Die meisten Algorithmen speichern eine neu aufgerufene Datei immer im Cache (Always Overwrite). Allerdings ist dies nicht immer sinnvoll, weil für jedes neue Dokument erst einmal Speicherplatz im Cache geschaffen werden muß. Dazu ist es nötig ein anderes File zu löschen. Wenn aber das gelöschte Dokument für den Nutzer viel „wertvoller" ist, als die neue Datei, so ist diese Ersetzung leistungsmindernd. Somit ist es besser, nur bedingt neue Dokumente im Cache abzulegen (Conditional Overwrite).

Gerade bei Videoverteilungssystemen sollte das Prinzip des Condititonal Overwrite angewendet werden. Es hat keinen Sinn Filme im Cache zu speichern, von denen man mit großer Sicherheit weiß, das sie sehr lange nicht mehr aufgerufen werden (z.B. spezielle Dokumentarfilme, Kunstfilme usw.). Schließlich wird durch jede Neuspeicherung ein andere Filme aus dem Cache vertrieben, der dann wieder kosten- und zeitaufwendig in den Cache zurückgeholt werden muß.

Eignung für VoD-Systeme:

In dieser Spalte erfolgt eine (natürlich subjektive) Bewertung über die Eignung der jeweiligen Strategie in Videoverteilungssystemen. Es werden drei Klassen gebildet, die folgende Eigenschaften besitzen:

1. Diese Algorithmen scheinen überaus geeignet, um in VoD-Systemen eingesetzt zu werden. Allerdings müssen auch hier Modifikationen vorgenommen werden, um den speziellen Systemanforderungen gerecht zu werden.
2. Der Ansatz dieser Algorithmen ist sehr interessant, die Strategie an sich aber nicht geeignet. Trotzdem sollte man sich diese Algorithmen näher ansehen, um aus den verwendeten Prinzipien Ideen für eine „optimale" Strategie zu entwickeln.
3. Die letzte und leider auch größte Gruppe umfaßt alle Algorithmen, die für Video-on-Demand-Systemen völlig ungeeignet sind. Sie wurden in dieser Arbeit trotzdem aufgeführt, da sie oft als Vergleich für andere Strategien dienen und die Studienarbeit eine möglichst umfassende Übersicht über die zur Zeit existierenden Caching-Strategien geben soll.

8.2 Tabellarischer Vergleich der Algorithmen

Tabelle 15: Parameter der beschriebenen Algorithmen Teil 1

Algorithmus	Alter der eingelagerten Dateien	Anzahl der Anfragen	Alterung	Zeit seit der letzten Anfrage	Einheitliche Blockgröße	Dateigröße	Kosten der Beschaffung/Erhaltung	Ladezeit
Adaptive LRU	Nein	Nein	Nein	Ja	Nein	Ja, Files werden nur bis zu einer bestimmten Größe geladen	Nein	Nein
AVI-MRU	Nein	Nein	Nein	Ja	Nein	Nein	Nein	Nein
BIST	Nein	Ja, durch IRG-Modell	Nein	Ja, durch IRG-Modell	Ja	Nein	Nein	Nein
Centrally Coordinated Caching	*[a]	*	*	Ja	*	*	*	*
CLOCK	Ja	Ja, aber es kann immer nur eine Anfrage berücksichtigt werden	Ja, weil nur die letzte Anfrage gespeichert wird	Ja, indirekt	Ja	Nein	Nein	Nein
DGCLOCK	Ja	Ja	Ja, indirekt	Ja, indirekt	Ja	Nein	Nein	Nein

Tabelle 15: Parameter der beschriebenen Algorithmen Teil 1

Algorithmus	Alter der eingelagerten Dateien	Anzahl der Anfragen	Alterung	Zeit seit der letzten Anfrage	Einheitliche Bockgröße	Dateigröße	Kosten der Beschaffung/Erhaltung	Ladezeit
Direct Client Cooperation	*	*	*	*	*	*	*	*
FBR	Nein	Ja	Ja	Ja, Sekundärstrategie	Nein	Nein	Nein	Nein
FIFO	Ja	Nein	Nein	Nein	Ja	Nein	Nein	Nein
GCLOCK	Ja	Ja	Ja, indirekt	Ja, indirekt	Ja	Nein	Nein	Nein
GD	Nein	Nein	Nein	Nein	Nein	Nein	Ja, Beschaffungskosten	Nein
GD-SIZE	Nein	Nein	Nein	Nein	Nein	Ja	Ja, Beschaffungskosten	Nein
GMS	*	*	*	Ja	*	*	*	*
Greedy Forwarding	*	*	*	*	*	*	*	*
Hash-Distributed Caching	*	*	*	Ja	*	*	*	*

Tabelle 15: Parameter der beschriebenen Algorithmen Teil 1

Algorithmus	Alter der eingelagerten Dateien	Anzahl der Anfragen	Alterung	Zeit seit der letzten Anfrage	Einheitliche Blockgröße	Dateigröße	Kosten der Beschaffung/Erhaltung	Ladezeit
HIST	Nein	Ja, durch IRG-Modell (durch h begrenzt)	Ja, nur die letzten h Anfragen werden berücksichtigt	Ja, durch IRG-Modell (durch h begrenzt)	Ja	Nein	Nein	Nein
Hint-based Algorithm	*	*	*	Ja	*	*	*	*
IRG-k	Nein	Ja, durch IRG-Modell	Nein	Ja, durch IRG-Modell	Ja	Nein	Nein	Nein
LAT	Nein	Nein	Nein	Nein	Nein	Ja	Nein	Ja
LEC	Nein	Ja	Nein	Nein	Nein	Nein	Ja, Kosten der Neubeschaffung	Nein
LECS	Nein	Ja	Nein	Nein	Nein	Ja	Ja, Kosten der Neubeschaffung	Nein
LEN	Nein	Ja	Nein	Nein	Nein	Ja	Nein	Nein
LFU	Nein	Ja	Nein	Nein	Ja	Nein	Nein	Nein

Tabelle 15: Parameter der beschriebenen Algorithmen Teil 1

Al rith-mus	Alter der eingelager-n Dateien	Anzahl der Anfragen	Alterung	Zeit seit der letzten Anfrage	ınheit-liche Bock-größe	Dateigröße	Kosten der Beschaf-fung/Erhal-tung	Ladezeit
LNC-A	Nein	, aber nur e letzten Anfragen	Ja	Ja, indirekt durch Zeitfaktor	Nein	Ja	in	Ja
LNC-R	Nein	, aber nur e letzten Anfragen	Ja	Ja, indirekt durch Zeitfaktor	Nein	Ja	Nein	Ja
LNC-RA	Nein	Ja, aber nur die letzten K Anfragen	Ja	Ja, indirekt durch Zeitfaktor	Nein	Ja	Nein	Ja
LNC-R-W3	Nein	Formel zur Durch-schnitts-berechnung	Ja	Ja, indirekt durch Zeitfaktor	Nein	Ja Bevorzu-gı ng von kl inen Fi es	Nein	Ja
LRD	Ja	Ja	Ja	Nein	Ja	Nein	Nein	Nein
LRU	Nein	Nein	Nein	Ja	Ja	Nein	Nein	Nein
LRU-k	Nein	Nein	Nein	Ja, aber nur die letzten K Anfragen	Ja	Nein	Nein	Nein

Tabelle 15: Parameter der beschriebenen Algorithmen Teil 1

Algorithmus	Alter der eingelagerten Dateien	Anzahl der Anfragen	Alterung	Zeit seit der letzten Anfrage	Einheitliche Bockgröße	Dateigröße	Kosten der Beschaffung/Erhaltung	Ladezeit
LRU-k-TH	Nein	Nein	Nein	Ja, aber nur die letzten K Anfragen	Nein	Ja, Files werden nur bis zu einer bestimmten Größe geladen	Nein	Nein
LRU-MIN	Nein	Nein	Nein	Ja	Nein	Ja, große Files werden zuerst gelöscht	Nein	Nein
LRU-SIZE	Nein	Nein	Nein	Ja, als Sekundärstrategie	Nein	Ja	Nein	Nein
LRU-THOLD	Nein	Nein	Nein	Ja	Nein	Ja, Files werden nur bis zu einer bestimmten Größe geladen	Nein	Nein
LRV	Nein	Ja	Nein	Ja	Nein	Ja	Nein	Nein

Tabelle 15: Parameter der beschriebenen Algorithmen Teil

Algorithmus	Alter der eingelagerten Dateien	Anzahl der Anfragen	Alterung	t seit der zten frage	Einheitliche Blockgröße	Dateigröße	Kosten der Beschaffung/Erhaltung	Ladezeit
MIX	Nein	Ja, mit Gewichtung	Nein	Ja, mit Gewichtung	Nein	Ja, mit Gewichtung	Nein	Ja, mit Gewichtung
MRU	Nein	Nein	Nein	Ja	Ja	Nein	Nein	Nein
N-Chance Forwarding	*	*	*	Ja	*	*	*	*
OPT	Nein	Nein	Nein	Nein	Ja	Nein	Nein	Nein
PFF	Nein	Nein	Nein	Ja	Ja	Nein	Nein	Nein
Pittkow/ Recker	Nein	Nein	Nein	Ja	Nein	Ja, kleine Files werden bevorzugt	Nein	Nein
Priority-Hints	Nein	Nein	Nein	Ja	Ja	Nein	Nein	Nein
Priority-LRU	Nein	Nein	Nein	Ja	Ja	Nein	Nein	Nein
PSS	Nein	Nein	Nein	Ja	Nein	Ja	Nein	Nein
RAND	Nein	Nein	Nein	Nein	Ja	Nein	Nein	Nein

Tabelle 15: Parameter der beschriebenen Algorithmen Teil 1

Algorithmus	Alter der eingelagerten Dateien	Anzahl der Anfragen	Alterung	Zeit seit d letzt n An ge	Einheitliche Bock größe	Dateigröße	Kosten d Beschaffung/Erhaltung	Ladezeit
RBC	Nein	Nein	Nein	Nein	Nein	Ja, Speicherplatzbedarf	Nein	Nein
SIZE	Nein	Nein	Nein	Nein	Nein	Ja	Nein	Nein
ZE-MRU	Nein	Nein	Nein	Ja	Nein	Ja	Nein	Nein
SLRU	Nein	Nein	Nein	Ja	Nein	Ja	Nein	Nein
SPACE xAGE	Nein	Nein	Nein	Ja	Nein	Ja	Nein	Nein
Static Caching	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein
STP**y	Nein	Nein	Nein	Ja, mit Faktor y besonders gewichtet	Nein	Ja	Nein	Nein
STWCS	Nein	Nein	Nein	Ja	Nein	Ja	Nein	Ja
STWS	Nein	Nein	Nein	Ja	Nein	Ja	Nein	Nein
2Q	Ja	Nein	Nein	Ja	Ja	Nein	Nein	Nein

Tabelle 15: Parameter der beschriebenen Algorithmen Teil 1

Algorithmus	Alter der eingelagerten Dateien	Anzahl der Anfragen	Alterung	Zeit seit der letzten Anfrage	Einheitliche Blockgröße	Dateigröße	Kosten der Beschaffung/Erhal-	Ladezeit
VALUE	Nein	Ja	Ja, indirekt	Ja	Nein	Ja	Nein	Nein
Weighted LRU	*	*	*	Ja	*	*	*	*
WIRG-k	Nein	Ja, durch IRG-Modell	Nein	Ja, durch IRG-Modell	Ja	Nein	Nein	Nein
WLFU	Nein	Ja	Nein	Nein	Nein	Ja	Nein	Nein
Wooster/ Abrams	Nein	Ja	Nein	Nein	Nein	Ja	Nein	Ja
WS	Nein	Nein	Nein	Nein	Ja	Nein	Nein	Nein

a. Eine Cooperative Caching-Strategie kann nicht alleine stehen, sondern muß mit einem weiteren Algorithmus kombiniert werden. Deshalb kann zu den verwendeten Parametern meistens keine Aussage gemacht werden, da diese von der verwendeten internen Ersetzungsstrategie

Tabelle 16: Parameter der beschriebenen Algorithmen Teil 2

Algorithmus	Bandbreite	Intervallbereich	Speicherreinigung	Prioritäten	Kooperation	AO/CO	Eignung für VoD-Systeme
Adaptive LRU	Nein	Nein	Nein	Nein	Nein	CO, Files dürfen threshold nicht überschreiten	3
AVI-MRU	Nein	Nein	Nein	Ja, indirekt durch Bildung von Gruppen	Nein	AO	3
BIST	Nein	Nein	Nein	Nein	Nein	AO	3
Centrally Coordinated Caching	*[a]	*	*	*	Ja, zentral	*	2
CLOCK	Nein	Nein	Nein	Nein	Nein	AO	3
DGCLOCK	Nein	Nein	Nein	Nein	Nein	AO	3
Direct Client Cooperation	*	*	*	*	Ja, dezentral	*	3
FBR	Nein	Nein	Nein	Nein	Nein	AO	3

Tabelle 16: Parameter der beschriebenen Algorithmen Teil 2

Algorithmus	Bandbreite	Intervallbereich	Speicherreinigung	Prioritäten	Kooperation	AO/CO	Eignung für VoD-Systeme
FIFO	Nein	Nein	Nein	Nein	Nein	AO	3
GCLOCK	Nein	Nein	Nein	Nein	Nein	AO	3
GD	Nein	Nein	Nein	Nein	Nein	AO	3
GD-SIZE	Nein	Nein	Nein	Nein	Nein	AO	3
GMS	*	*	*	*	Ja, zentral	*	1
Greedy Forwarding	*	*	*	*	Ja, zentral	*	2
Hash-Distributed Caching	*	*	*	*	Ja, zentral	*	2
HIST	Nein	Nein	Nein	Nein	Nein	AO	3
Hint-based Algorithm	*	*	*	*	Ja, dezentral	*	1
IRG-k	Nein	Nein	Nein	Nein	Nein	AO	1
LAT	Ja	Nein	Nein	Nein	Nein	AO	2
LEC	Nein	Nein	Nein	Nein	Nein	AO	2

Tabelle 16: Parameter der beschriebenen Algorithmen Teil 2

Algorithmus	Bandbreite	Intervallbereich	Speicherreinigung	Prioritäten	Kooperation	AO/CO	Eignung für VoD-Systeme
LECS	Nein	Nein	Nein	Nein	Nein	AO	2
LEN	Nein	Nein	Nein	Nein	Nein	AO	3
LFU	Nein	Nein	Nein	Nein	Nein	AO	3
LNC-A	Nein	Nein	Nein	Nein	Nein	CO	2
LNC-R	Nein	Nein	Nein	Nein	Nein	AO	2
LNC-RA	Nein	Nein	Nein	Nein	Nein	CO	1
LNC-R-W3	Nein	Nein	Nein	Nein	Nein	AO	3
LRD	Nein	Ja	Nein	Nein	Nein	AO	2
LRU	Nein	Nein	Nein	Nein	Nein	AO	3
LRU-k	Nein	Nein	Nein	Nein	Nein	AO	3
LRU-k-TH	Nein	Nein	Nein	Nein	Nein	CO, Files dürfen threshold nicht überschreiten	3
LRU-MIN	Nein	Nein	Nein	Nein	Nein	AO	3

Tabelle 16: Parameter der beschriebenen Algorithmen Teil 2

Algorithmus	Bandbreite	Intervallbereich	Speicherreinigung	Prioritäten	Kooperation	AO/CO	Eignung für VoD-Systeme
LRU-SIZE	Nein	Nein	Nein	Nein	Nein	AO	3
LRU-THOLD	Nein	Nein	Nein	Nein	Nein	CO, Files dürfen threshold nicht überschreiten	3
LRV	Nein	Nein	Nein	Nein	Nein	AO	2
MIX	Nein	Nein	Nein	Nein	Nein	AO	2
MRU	Nein	Nein	Nein	Nein	Nein	AO	3
N-Chance Forwarding	*	*	*	*	Ja, zentral	*	2
OPT	Nein	Nein	Nein	Nein	Nein	AO	keine Wertung[b]
PFF	Nein	Nein	Nein	Nein	Nein	AO	3
Pittkow/ Recker	Nein	Nein	Ja	Nein	Nein	AO	2
Priority-Hints	Nein	Nein	Nein	Ja	Nein	AO	2

Tabelle 16: Parameter der beschriebenen Algorithmen Teil 2

Algorithmus	Bandbreite	Intervallbereich	Speicherreinigung	Prioritäten	Kooperation	AO/CO	Eignung für VoD-Systeme
Priority-LRU	Nein	Nein	Nein	Ja	Nein	AO	
PSS	Nein	Nein	Nein	Nein	Nein	AO	3
RAND	Nein	Nein	Nein	Nein	Nein	AO	3
RBC	Ja	Nein	Nein	Nein	Nein	CO	1
SIZE	Nein	Nein	Nein	Nein	Nein	AO	3
SIZE-MRU	Nein	Nein	Nein	Nein	Nein	AO	3
SLRU	Nein	Nein	Nein	Nein	Nein	AO	3
SPACE xAGE	Nein	Nein	Nein	Nein	Nein	AO	3
Static Caching	Nein	Nein	Ja	Nein	Nein	CO, nur zu bestimmten Zeiten werden neue Files gespeichert	2
STP*	Nein	Nein	Nein	Nein	Nein	AO	3
STW	Nein	Nein	Nein	Nein	Nein	AO	3

Tabelle 16: Parameter der beschriebenen Algorithmen Teil 2

Algorithmus	Bandbreite	Intervallbereich	Speicherreinigung	Prioritäten	Kooperation	AO/CO	Eignung für VoD-Systeme
STWS	Nein	Nein	Nein	Nein	Nein	AO	3
2Q	Nein	Nein	Nein	Nein	Nein	AO	3
VALUE	Nein	Nein	Nein	Nein	Nein	CO	2
Weighted LRU	*	*	*	*	Ja, zentral	*	3
WIRG-k	Nein	Ja	Nein	Nein	Nein	AO	2
WLFU	Nein	Nein	Nein	Nein	Nein	AO	3
Wooster/ Abrams	Ja	Nein	Nein	Nein	Nein	AO	1
WS	Nein	Ja	Nein	Nein	Nein	AO	2

a. Eine Cooperative Caching-Strategie kann nicht alleine stehen, sondern muß mit einem weiteren Algorithmus kombiniert werden. Deshalb kann zu den verwendeten Parametern meistens keine Aussage gemacht werden, da diese von der verwendeten internen Ersetzungsstrategie

b. Der OPT Algorithmus stellt eine nicht-realisierbare Strategie dar.

8 Ausblick

Zuerst einmal muß klargestellt werden, daß diese Arbeit nicht den Anspruch auf Vollständigkeit erhebt. Gerade die Entwicklung im World Wide Web steht erst am Anfang und hier wird es noch viele neue Entwicklungen und Veränderungen geben, die dann natürlich auch zu neuen Caching-Strategien führen. Trotz allem wird hier jedoch eine recht aktuelle Übersicht geliefert, die Hilfestellung und Anregung für das Entwickeln und Implementieren neuer Algorithmen geben soll. Die Arbeit wurde dabei so allgemein wie möglich gehalten, um sie auch für andere Einsatzgebiete verwendbar zu machen.

Vom Grundkonzept stellt sie allerdings eine Grundlagenarbeit für weitere Arbeiten auf dem Themengebiet Videoverteilungssysteme dar. In Simulationen sollen die hier aufgeführten und für geeignet gehaltenen Algorithmen untersucht und notwendige Änderungen vorgenommen werden, um so einen „optimalen" Ersetzungsalgorithmus für VoD-Systeme zu entwickeln.

Grundsätzlich kann man hier schon sagen, daß dieser „optimale" Algorithmus auf jeden Fall mit mehreren Parametern arbeiten muß, d.h. er wird entsprechend komplex. Dadurch wird die Entscheidungsfindung relativ lange dauern, was allerdings bei VoD-Systemen nicht so sehr ins Gewicht fällt. Da ein Server nur eine kleine Anzahl von Filmen in seinem Cache speichern kann, muß jede Änderung sorgfältig überlegt werden. Schließlich verursacht das Übertragen von Filmen einen hohen Zeit- und Kostenaufwand.

Des weiteren sollte auf jeden Fall das Prinzip des Cooperative Caching angewendet werden, denn nur so kann ein funktionierendes Video-on-Demand-System überhaupt zur Verfügung gestellt werden. Ein System, in dem jeder Server isoliert arbeitet ist zu langsam, zu teuer und zu unzuverlässig. Kunden die solche Erfahrungen machen werden binnen kürzester Zeit wieder zu den Videotheken übergehen.

9 Stichwortverzeichnis

10 Literaturverzeichnis

[ASA+95] Marc Abrams, Charles R. Standbridge, Ghaleb Abdulla, Stephen Williams, Edward A. Fox, Caching Proxies: Limitation and Potentials, Proceedings of the 4th International World Wide Web Conference, Boston, pp.119-133, December 1995, oder <URL:http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html>

[AWY+96] Charu C. Aggarwal, Joel L. Wolf, Philip S. Yu, Marina Epelman, On Caching Policies for Web Objects, IBM Research Report, RC 20619, August 1996

[BH96] J. Bolot, P. Hoschka, Performance Engineering of World Wide Web: Application to Dimensioning and Cache Design, Proceedings of the 5th International World Wide Web Conference, Paris, May 1996, oder <URL:http://www.w3j.com/3/s3.bolot.html>

[BM96] Giuseppe Bianchi, Riccardo Melen, Performance and Dimensioning of a Hierarchical Video Storage Network for Interactive Video Services, EET European Transaction on Telecommunications and Related Technology, pp.349-358, July/August 1996

[BM97] Guiseppe Bianchi, Riccardo Melen, The Role of Local Storage in Supporting Video Retrieval Services on ATM Networks, IEEE/ACM Transactions of Networking, pp.882-892, December 1997

[CI97] Pei Cao, Sandy Irani, Cost-Aware WWW Proxy Caching Algorithms, Technical Report, Dept. of Computer Sciences, University of Wisconsin-Madison, CS-TR-97-1343, 1997

[CO76] R.G.Casey, I.M.Osman, Replacement algorithms for storage management in relational data bases, The Computer Journal, pp.306-314, November 1976

[DEN96] Thorten Dencker, Hierarchical Internet Object Cache, <URL:http://trumpf-3.rz.uni-mannheim.de/www/sem96w/td/index2.html>, 1996

[DOM98] Claudius Dommel, Entwurf und Realisierung eines Caching-Mechanismus zur Kommunikation mit mobilen Endgeräten, Studienarbeit, Darmstadt, Nr. KOM-S-0013, 1998

[DR90] M.V. Devarakonda, J.T. Robinson, Data cache management using frequeny-bases replacement, ACM SIGMETRICS Performance Evaluation Review, pp.134-142, May 1990

[DWA+94] R.Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky, Jim Zelenka, Cooperative Caching: Using Remote Client Memory to Improve File System Performance, Technical Report, University of California at Berkeley, CSD-94-844, 1994

[EFF81] Wolfgang Effelsberg, Systempufferverwaltung in Datenbanksystemen, Dissertation, Fachbereich Informatik, Technische Hochschule Darmstadt, Deutschland 1981

[EH84] Wolfgang Effelsberg, Theo Haerder, Principles of Database Buffer Management, ACM Transactions on Database Systems, pp.560-595, December 1984

[FCL92] Michael J. Franklin, Michael J. Carey, Miron Livny, Global Memory Management in Client-Server DBMS Architectures, Proceedings of the International Conference on VLDB 1992, pp.596-609, August 1992

[FMP+95] Michael J. Feeley, William E. Morgan, Frederic H. Pighin,Anna R. Karlin, Henry M Levy, Implementing Global Memory Management in a Workstation Cluster, Proceedings of the 15th Symposium on Operating Systems Principles, pp.201-212, December 1995

[GBW97] Carsten Griwodz, Michael Bär, Lars C. Wolf, Long-term Movie Popularity Models in Video-on-Demand Systems, ACM International Multimedia Conference, pp.349-357, November 1997

[JCL89] Rajiv Jauhari, Michael J. Carey Miron Livny, Priority in DBMS Resource Scheduling, Proceedings of the International Conference on VLDB 1989, pp.397-410, 1989

[JCL90] Rajiv Jauhari, Michael J. Carey Miron Livny, Priority-Hits: An Algorithm for Priority-Bases Buffer Management, Proceedings of the International Conference on VLDB 1990, pp.708-721, 1990

[JS94] Theodore Johnson, Dennis Shasha, 2Q: A Low Overhead High Performance Buffer Replacement Algorithm ,Proceedings of the International Conference on VLDB 1994, pp.439-450, 1994

[KS98] P. Krishnan, Binay Sugla, Utility of co-operating Web proxy caches, Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, April 1998, oder <URL:http://www7.conf.au/programme/fullpapers/1892/com1892.html>

[MAF93] Silvano Maffeis, Cache Management Algorithms for Flexible Filesystems, ACM SIGMETRICS Performance Evaluation Review, pp.19-25, December 1993

[MAR96] Evangelos P. Markatos, Main memory caching of Web documents, Proceedings of the 5th International World Wide Web Conference, Paris, May 1996, oder <URL:http:// www.csi.forth.gr/proj/arch-vlsi/papers/WWW5/Overview.html>

[MGS+70] R.L. Mattson, J. Gecsei, D.R. Slutz, I.L.Traiger, Evaluation techniques for storage hierarchies, IBM System Journal, pp.78-101, March 1970

[MLB95] Radhika Malpani, Jacob Lorch, David Berger, Making World Wide Web Caching Server Cooperate, Proceedings of the 4th International World Wide Web Conference, Boston, 1995, oder <URL:http://www.w3.org/Conferences/WWW4/Papers/59/>

[NLN97] Nicolas Niclausse, Zhen Liu, Philippe Nain, A New Efficient Caching Policy for the World Wide Web, <URL:http://www.inria.fr/mistral/personnel/Nicolas.Niclausse/ articles/wisp98>, 1997

[OOW93] Elizabeth J. O'Neil, Patrick E. O'Neil, Gerhard Weikum, The LRU-K page replacement algorithm for database disk buffering, ACM SIGMOD Record, pp.297-306, June 1993, oder <URL:http:// paris.cs.uni-sb.de/public_html/papers/LRU-k_reports.ps.Z>

[PD96] Thomas Partl, Adam Dingle, A Comparison of WWW Caching Algorithm Efficiency, <URL: http://cache.kaist.ac.kr/links/a_comparison_of.ps>, 1996

[PG95] Vidyadhar Phalke, Bhaskarpillai Gopinath, An Inter-Reference Gap Model for Temporal Locality in Program Behaviour, Proceedings Joint International Conference on Measurement & Modeling of Computer Systems, pp.291-300, 1995

[PG97] Vidyadhar Phalke, Bhaskarpillai Gopinath, Compression-Based Program Characterization for Improvement Cache Memory Performance, IEEE Transactions on Computers, pp.1174-1185, November 1997

[PR94] James E. Pitkow, Margaret M. Recker, A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns, Proceedings of the 2nd International World Wide Web Conference, Chicago, USA, October 1994, oder <URL: http://gita.srce.hr/ caching/algorithm.html>

[RED95] Narasimha Reddy, Evaluation of caching strategies for an internet server, Technical Report, Texas A&M University, 1995

[RL96] Benjamin Reed, Darrell D.E Long, Analysis of Caching Algorithms for Distributed File Systems, ACM SIGOPS Operational Systems Review, pp.12-17, July 1996

[RV97] Luigi Rizzo, Lorenzo Vivisano, Replacement policies for a proxy cache, Technical Report, Dept. of Computer Science, University College London, LR-96-731, 1997

[SH96] Prasenjit Sarkar, John Hartman, Efficient Cooperative Caching using Hints, Proceedings of the 2nd Usenix Symposium an Operating Systems Design and Implementation OPSDI´96, oder ACM SIGOPS Operating Systems Review, pp.35-46, October 1996

[SMI81] Alan Jay Smith, Long Term File Migration: Development and Evaluation of Algorithm, Communication of ACM, pp.521-532, August 1981

[SSV96] Peter Scheuermann, Junho Shim, Radek Vingralek, WATCHMAN: A Data Warehouse Intelligent Cache Manager, Proceedings of the International Conference on VLDB 1996, pp.51-62, !996

[SSV97] Peter Scheuermann, Junho Shim, Radek Vingralek, A Case For Delay-Conscious Caching of Web Documents, Proceedings of the 6th International World Wide Web Conference, Santa Clara, California, 1997, oder <URL: http://www6.nttlabs.com/HyperNews/get/PAPER20.html>

[TAN90] Andrew S. Tanenbaum, Betriebssysteme Entwurf und Realisierung Teil 1:Lehrbuch,Carl Hanser Verlag München Wien, 1990

[OOW93] Elizabeth J. O´Neil, Patrick E. O´Neil, Gerhard Weikum, The LRU-K page replacement algorithm for database disk buffering, ACM SIGMOD Record, pp.297-306, June 1993, oder <URL:http:// paris.cs.uni-sb.de/public_html/papers/LRU-k_reports.ps.Z>

[PD96] Thomas Partl, Adam Dingle, A Comparison of WWW Caching Algorithm Efficiency, <URL: http://cache.kaist.ac.kr/links/a_comparison_of.ps>, 1996

[PG95] Vidyadhar Phalke, Bhaskarpillai Gopinath, An Inter-Reference Gap Model for Temporal Locality in Program Behaviour, Proceedings Joint International Conference on Measurement & Modeling of Computer Systems, pp.291-300, 1995

[PG97] Vidyadhar Phalke, Bhaskarpillai Gopinath, Compression-Based Program Characterization for Improvement Cache Memory Performance, IEEE Transactions on Computers, pp.1174-1185, November 1997

[PR94] James E. Pitkow, Margaret M. Recker, A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns, Proceedings of the 2nd International World Wide Web Conference, Chicago, USA, October 1994, oder <URL: http://gita.srce.hr/caching/algorithm.html>

[RED95] Narasimha Reddy, Evaluation of caching strategies for an internet server, Technical Report, Texas A&M University, 1995

[RL96] Benjamin Reed, Darrell D.E Long, Analysis of Caching Algorithms for Distributed File Systems, ACM SIGOPS Operational Systems Review, pp.12-17, July 1996

[RV97] Luigi Rizzo, Lorenzo Vivisano, Replacement policies for a proxy cache, Technical Report, Dept. of Computer Science, University College London, LR-96-731, 1997

[SH96] Prasenjit Sarkar, John Hartman, Efficient Cooperative Caching using Hints, Proceedings of the 2nd Usenix Symposium an Operating Systems Design and Implementation OPSDI´96, oder ACM SIGOPS Operating Systems Review, pp.35-46, October 1996

[SMI81] Alan Jay Smith, Long Term File Migration: Development and Evaluation of Algorithm, Communication of ACM, pp.521-532, August 1981

[SSV96] Peter Scheuermann, Junho Shim, Radek Vingralek, WATCHMAN: A Data Warehouse Intelligent Cache Manager, Proceedings of the International Conference on VLDB 1996, pp.51-62, !996

[SSV97] Peter Scheuermann, Junho Shim, Radek Vingralek, A Case For Delay-Conscious Caching of Web Documents, Proceedings of the 6th International World Wide Web Conference, Santa Clara, California, 1997, oder <URL: http://www6.nttlabs.com/HyperNews/get/PAPER20.html>

[TAN90] Andrew S. Tanenbaum, Betriebssysteme Entwurf und Realisierung Teil 1:Lehrbuch,Carl Hanser Verlag München Wien, 1990

[TAN92] Andrew S. Tanenbaum, Modern Operating Systems, Prentice-Hall International London, 1992

[TAT97] Igor Tatarinov, Cache Policies for Web Servers,. NDSU Technical Report, NDSU-CSOR-TR-97-05, 1997, oder <URL:http://www.cs.ndsu.nodak.edu/~tatarino/pubs>

[TAT98] Igor Tatarinov, Performance Analysis of Cache Policies for Web Servers, To appear in Proceedings of 9th International Conference on Computing and Information ICCI´98, June 1998, oder NDSU Technical Report, NDSU-CSOR-TR-97-06

[TRS97] Igor Tatarinov, Alex Rousskov, Valery Soloviev, Static Caching in Web Servers, Proceedings of 6th IEEE International Conference of Computer Comm. and Networks IC3N´97, September 1997

[TVD+96] Renu Tewari, Harrick M Vin, Asit Dan, Dinkar Sitaram, Caching in Bandwidth and Space Constrained Hierarchical Hyper-media Servers, Technical Report, Dept. of Computer Sciences, University of Texas at Austin, CS-TR-96-30, 1996

[TVD+97] Renu Tewari, Harrick M Vin, Asit Dan, Dinkar Sitaram, Resource Based Caching for Web Servers, Proceedings of ACM/SPIE Multimedia Computing and Networking MMCN´98, San Jose, pp.191-204, January 1998, oder <URL:http://www.cs.utexas.edu/users/dmcl/allpapers.html>

[WA97] Roland P. Wooster, Marc Abrams, Proxy Caching That Estimates Page Load Delays, Proceedings of the 6th International World Wide Web Conference, Santa Clara, California, 1997, oder <URL: http://www6.nttlabs.com/HyperNews/get/PAPER250.html>

[WEB92] Darryl L. Willick, Derek L. Eager, Richard B. Bunt, Disk Cache Replacement Policies for Network Fileservers, Technical Report, Department of Computa-tional Science, University of Saskatchewan Saskatoon, DR-92-4, October 1992

[WOO96] Roland Wooster, Optimizing Response Time, Rather than Hit Rates, of WWW Proxy Caches, Master of Science Thesis, Computer Science Dept., Virginia Tech., December 1996

11 Anhang A

Nicht alle Algorithmen die während der Erstellung dieser Studienarbeit gefunden wurden, haben für den Einsatz in Videoverteilungssystemen einen Nutzen. Vor allem Seitenersetzungsalgorithmen sind in VoD-Systemen kaum einsetzbar, deshalb wurden in der Arbeit auch nur die bekanntesten und gängigsten genauer beschrieben. In diesem Anhang sind nun weitere Algorithmen und ihre Quellen aufgeführt, um interessierten Lesern den Zugang zu „spezielleren" Strategien zu erleichtern.

Seitenersetzungsalgorithmen:

- CLIMB

  O. Aven, L. Boguslavsky, Y. Kogan, Some results on distribution-free analysis of paging algorithm, IEEE Transactions on Computers, pp.737-745, July 1976

- DBMIN

  Hong-Tai Chou, David J. DeWitt, An Evaluation of Buffer Management Strategies for Relational Database Systems, Proceedings of the International Conference on VLDB 1985, pp.127-141, August 1985

- DMIN

  Robert L. Budzinski, Edward S. Davidson, Wataru Mayeda, Harold S. Stone, An Algorithm for Computing the Optimal Dynamic Allocation in a Virtual memory Computer, IEEE Transactions on Software Engineering, pp.113-120, January 1981

- DS

  Allen Reiter, A Study of Buffer Management Policies for Data Management Systems, Technical Report, Mathematics Research Center, University of Wisconsin-Madison, March 1976

- DWS

  Alan J. Smith, A Modified Working Set Paging Algorithm, IEEE Transactions on Computers, pp.907-914, September 1976

- GLRU

  E.B. Fernandez, T. Lang, C. Wood, Effect of Replacement Algorithms on a Paged Buffer Database System, IBM Journal of Research and Development, pp.185-196, March 1978

- MG-x-y

  Raymond Ng, Chrislos Faloutsos, Timos Sellis, Flexible Buffer Allocation Based on Marginal Gains, ACM SIGMOD Record, pp.387-396, June 1991

- VMS clock

  H.M. Levy, R.H. Eckhouse, Computer Programming and Architecture: The VAX, Digital Press, 1989

- VMOS

  M.H. Fogel, The VMOS paging algorithm, a practical implementation of the working set model, Operating System Review, pp.8-17, January 1974

- VSWS

  Demonico Ferrari, The Variable-Interval Sampled Working Set Policy, IEEE Transactions on Software Engineering, pp.299-305, May 1983

Prepaging:

- DPMIN

  Radha Krishan Arora, R.K. Subramanian, An optimal demand prepaging algorithm, Information Processing Letters, pp.132-136, April 1978

Algorithmus für einen räumlichen Speicher:

- LRD-Manhatten

  Apostolos Papdopoulos, Yannis Manolopoulos, Global Page Replacement in Spatial Databases, Database and Expert Systems Applications DEXA´96, pp.855-864, 1996