Local Latency Reduction on the Web: Compressing, Caching and Prefetching

Seminar „Quality of Internet Services in E-Commerce“
Wintersemester 2000/2001

Axel Beckert, abe@cs.uni-sb.de

Betreuer: Prof. Gerhard Weikum

17. Februar 2000

Titelseite

- Thema des Vortrages: Lokale Maßnahmen um Wartezeiten (inklusive der Reduktion von genutzter Bandbreite) im Web zu reduzieren.

Übersicht

1 Motivation **3**

2 Ansatzpunkte zur Traffic-Reduktion zwischen Client und Proxy **4**

2.1 Komprimierung 5

2.1.1 Δ-Komprimierung 5

2.1.2 HTTP Transfer-Encoding 5

2.1.3 Compilierte (binäre) Versionen von Markup-Sprachen 5

2.2 Proxy-Initiated Prefetching 7

2.2.1 PPM Prefetching Algorithmus 8

2.2.2 Potential verschiedener Konfigurationen des PPM-Algorithmus 10

2.2.3 Potential von Prefetching 11

2.2.4 Aufschlüsselung von *prefetched* nach Dateitypen 12

2.2.5 Aufschlüsselung der Genauigkeit nach Dateitypen 12

2.2.6 Auswirkungen von vereinfachten Implementationen 13

3 Web Proxy Caching **15**

3.1 Cooperative Web Proxy Caching 16

3.2 HTML Macro-Preprocessing to Support Dynamic Document Caching 18

4 Literatur **19**

1 Motivation

Hohe Wartezeiten und niedrige Byteraten im Web können verschiedene Gründe haben:

1. Modem-Verbindung oder andere schmalbandige Verbindung des Clients zum Rest des Internet
2. Temporär ausgefallene Router oder Leitungen und dadurch Überlastungen auf den Ausweichstrecken
3. Überlastete Server

Lokale Gegenmaßnahmen:

1. Datenvolumen der Modem-Verbindung verringern
2. Chaching mittels einem oder mehreren Proxies

Motivation

- Hohe Wartezeiten und niedrige Byteraten im Web können verschiedene Gründe haben:
  1. Modem-Verbindung oder andere schmalbandige Verbindung des Clients zum Rest des Internet
  2. Temporär ausgefallene Router oder Leitungen und dadurch Überlastungen auf den Ausweichstrecken
  3. Überlastete Server

Nachdem uns Johannes letzte Woche gezeigt hat, wie man auf Serverseite Wartezeiten reduziert, wollen wir uns heute mal der Client- bzw. ISP-Seite zuwenden.

- Aus schmalbandigen Client-Anbindungen resultierende Probleme:
  1. Schmalbandige Verbindungen über Telefonleitungen werden meist nach Dauer und selten nach Volumen berechnet
  2. Tröpfeln aus der Leitung kostet (intellektuelle) Zeit und bei Verbindungen über Telefonleitungen auch mehr Geld, selbst wenn der Kunde nichts für die Überlastung irgendwo anders kann

- Lokale Gegenmaßnahmen:
  1. Datenvolumen der Modem-Verbindung verringern

- Gegen die Ursachen von Überlastungen auf dem Server oder irgendwo auf der Strecke zwischen ISP und Server können weder Kunde noch ISP etwas unternehmen. Trotzdem möchte der ISP seinen Benutzern eine gute Qualität bieten.

- Lokale Gegenmaßnahmen zur Symptomverringerung und Qualitätssteigerung:

    1. Chaching mittels einem oder mehreren Proxies

2 Ansatzpunkte zur Traffic-Reduktion zwischen Client und Proxy

- Großer Cache auf Client-Seite
- Filterung von nicht gewünschten Inhalten durch den Benutzer
- Δ-Komprimierung [Mogul, Douglis, Feldmann & Krishnamurthy, 1997]
- Komprimierung von HTML auf Anwendungsebene [Mogul et al., 1997]
- Durch Compilierung komprimierte Markup-Sprachen (z. B. WAP WMLC)
- Client-Initiated Prefetching (z. B. „Webbeschleuniger“)
- Proxy-Initiated Prefetching (Pushing)
- Proxy-Suggested Prefetching via HTTP Request Piggyback

Ansatzpunkte zur Traffic-Reduktion zwischen Client und Proxy

- Großer Cache auf Client-Seite $\Longrightarrow$ Gegebenenfalls durch lokalen Proxy-Cache. (Die meisten der im folgenden genannten Techniken setzen dies voraus, um browserunabhängig zu bleiben.)
- Filterung von nicht gewünschten Inhalten durch den Benutzer $\Longrightarrow$ Keine Selbstzensur, sondern Filtern von Werbebannern, Hintergrundmusik oder -bildern, etc. Beispiel: Der Proxy „JunkBuster“ (verfügbar für Unix & Windows) fängt die Requests von Werbebanner ab, in dem er die URL gegen entsprechende reguläre Ausdrücke matcht.
- Δ-Komprimierung $\Longrightarrow$ „diff“
- Komprimierung von HTML auf Anwendungsebene $\Longrightarrow$ HTTP/1.1 und Content-Negotiation. Übliche Komprimierungsverfahren sind hier „deflate“ und „gzip“.
- Durch Compilierung komprimierte Markup Sprachen $\Longrightarrow$ Tags und (definierten) häufigen anderen Zeichenketten (z. B. `http://www.`)
- Client-Initiated Prefetching $\Longrightarrow$ Webbeschleuniger (auch nur ein lokaler Proxy), wird hier nicht weiter darauf eingegangen. Wieso, sehen wir später.
- Proxy-Initiated Prefetching (Pushing) $\Longrightarrow$ Proxy errät das oder die nächsten Dokumente. Auf Client-Seite sitzt ebenfalls ein Proxy, der als Browser-Cache fungiert und auf die Push-Anfragen des Proxys auf der anderen Seite der Leitung reagiert.

2.1 Komprimierung

2.1.1 $\triangle$-Komprimierung

Braucht sehr großen Cache auf beiden Seiten:

- lokal, da eine ältere Version zum Vergleichen da sein muß und
- remote, da sowohl die aktuelle als auch (ggfs. mehrere) ältere Versionen — gegebenenfalls als „diff“ gespeichert — vorhanden sein müssen.

Reduziert das Übertragungsvolumen geänderter Objekte um durchschnittlich 88% [Mogul et al., 1997] und schafft bis zu 14,6% Verzögerungsverringerung.

2.1.2 HTTP Transfer-Encoding

HTTP/1.1 unterstützt neben der Übertragung komprimiert gespeicherter Dokumente (via *Content Negotiation*) auch die on-the-fly Komprimierung nur für die Übertragung der Daten (HTTP-Header `Transfer-Encoding`). Übliche Komprimierungsverfahren sind hier „deflate“ und „gzip“.

Reduziert das Übertragungsvolumen um durchschnittlich 25% [Mogul et al., 1997].

2.1.3 Compilierte (binäre) Versionen von Markup-Sprachen

Da es sich bei HTML- und XML-Dokumenten um strukturierten Text mit fest definierten Tags und anderen häufigen Zeichenketten (z. B. `http://www.`) handelt, läßt er sich gut in eine strukturierte, binäre Form compilieren.

Aktuelle Anwendung: WAP-Seiten werden in compilierter Form (Compiled Wireless Markup Language, WMLC) übertragen und interpretiert, da WAP-Clients üblicherweise eine sehr schmalbandige Anbindung und nur einen sehr kleinen Cache besitzen.

Komprimierung

- Δ-Komprimierung $\Longrightarrow$ `diff`, braucht sehr großen Cache auf beiden Seiten:

  lokal , da eine ältere Version zum Vergleichen da sein muß und

  remote , da sowohl die aktuelle als auch (ggfs. mehrere) ältere Versionen — gegebenenfalls als `diff` gespeichert — vorhanden sein müssen.

  Reduziert das Übertragungsvolumen geänderter Objekte um durchschnittlich 88% [Mogul et al., 1997] und schafft bis zu 14,6% Verzögerungsverringerung.

- Komprimierung von HTML auf Anwendungsebene $\Longrightarrow$ Da es sich bei HTML um Text mit vielen sich wiederholenden Zeichenketten (insbesondere den Tags) handelt, läßt er sich gut komprimieren.

  HTTP/1.1 unterstützt neben der Übertragung komprimiert gespeicherter Dokumente (via *Content Negotiation*) auch die on-the-fly Komprimierung nur für die Übertragung der Daten (HTTP-Header `Transfer-Encoding`). Übliche Komprimierungsverfahren sind hier „deflate“ und „gzip“.

  - Bietet sich sehr für XML-Dokumente an: Sie haben üblicherweise selbstbeschreibende (und damit längere) Tags und sind außerdem häufig feiner strukturiert.

    Für Webseiten, die massiv Tabellen für's Layout einsetzen gilt dies allerdings auch.

  - Verringerung des Vorteils: Modem-Verbindungen komprimieren auch schon $\Longrightarrow$ Effekt insgesamt geringer.
  - Großteil des Traffics sind (leider) Grafiken (GIF, JPEG), die zum Teil selbst schon hochkomprimiert sind (JPEG, PNG) $\Longrightarrow$ Effekt geringer.
  - Nachteil: Erhöhter CPU-Bedarf (heute aber kaum noch relevant)

  Reduziert das Übertragungsvolumen geänderter Objekte um durchschnittlich 25% [Mogul et al., 1997].

- Durch Compilierung komprimierte Markup Sprachen $\Longrightarrow$ Da es sich bei HTML (und auch in XML definierten Sprachen) um (mehr oder minder) strukturierten Text mit fest definierten Tags und anderen häufigen Zeichenketten (z. B. `http://www.`) handelt, läßt er sich gut in eine strukturierte, binäre Form compilieren.

  - Großer Nachteil: Komprimiert nur das Markup und wenige (vordefinierte) häufig vorkommende Strings wie z. B. `http://www..`
  - Aktuelle Anwendung: WAP-Seiten werden in compilierter Form (Compiled Wireless Markup Language, WMLC) übertragen und interpretiert, da WAP-Clients üblicherweise eine sehr schmalbandige Anbindung und nur einen sehr kleinen Cache besitzen.
  - Durch die kurzen Tags in HTML nicht unbedingt berauschende Komprimierung. Allerdings: Heutige Seiten haben für das Layout viele (oft verschachtelte) Tabellen, womit sich Komprimierung eher wieder lohnt.

- Für XML-Dokumente und HTML-Seiten mit viel Layout-Tabellen gilt wieder dasselbe wie für Komprimierung auf Anwendungsebene.
- Nachteil: Betrifft im Web nur HTML, Großteil des Traffics sind aber (leider) Grafiken (GIF & JPEG)
- Nachteil (vor allem für XML-Dokumente): Die binäre Darstellung muß für jeden DTD neu definiert werden.

2.2 Proxy-Initiated Prefetching [Fan, Cao, Lin & Jacobson, 1999]

Vorteile gegenüber Client-Initiated Prefetching:

- Die Vorhersage basiert auf statistischen Daten über alle Benutzer des Proxys.
- Das angefragte Dokument muß zur Vorhersage nicht geparst werden.
- Prefetching wird nur auf Dokumente angewendet, die der Proxy im Cache hat. Somit wird durch Prefetching ausschließlich im LAN zusätzlicher Traffic erzeugt, nicht aber in WANs.

Es gibt zwei Varianten:

- Prefetching mit Benachrichtigung über Hits im Cache des Browsers (bzw. des lokalen Proxys)
- Prefetching ohne Benachrichtigung über Hits im Cache des Browsers (bzw. des lokalen Proxys)

Proxy-Initiated Prefetching [Fan et al., 1999]

Vorteile gegenüber Client-Initiated Prefetching:

- Der Proxy kann das Verhalten von sehr vielen Benutzern verwenden, um das Verhalten eines Benutzers vorauszusagen.
  - Einerseits kann dies zu einer Verringerung der Genauigkeit führen, da jeder Benutzer sich unterschiedlich verhält.
  - Andererseits kann der Algorithmus so auf eine wesentlich größere Datenbasis zurückgreifen, was die Wahrscheinlichkeit, daß ihm überhaupt Daten zur Verfügung stehen, wesentlich erhöht wird.
  - Außerdem sollte sowieso der Browser-Cache einen Großteil der wiederholten Zugriffe eines Benutzers abfangen. Dies heißt auch, daß die Vorhersagen für Prefetching sich vor allem um Dokumente drehen, die der Benutzer vorher noch nicht betrachtet hat, was bei einer Datenbasis aus den Zugriffen vieler Benutzer wesentlich einfacher ist, als bei der Datenbasis eines Benutzers.
  - Ein weiterer Vorteil einer Datenbasis vieler Benutzer ist, daß dadurch auch wesentlich weiter (eben nicht nur Tiefe 1) in die Zukunft geschaut werden kann.
  - Ebenso können die nicht unwichtigen Co-Referenzen (Beispiel: Suchergebnis einer Suchmaschine) wesentlich leichter anhand statistischer Daten beachtet werden als durch Analysieren der Linkstruktur $\Longrightarrow$**Zeichnung machen!**

- Der Proxy muß deshalb auch nicht das gerade angefragte Dokument parsen, sondern kann auf eine interne, effiziente Datenstruktur zum Berechnen der Vorhersagen zurückgreifen.

- Das Prefetching wird nur auf Dokumente angewendet, die der Proxy im Cache hat, die also auf jeden Fall schnell verfügbar sind und die komplette Bandbreite des Clients ausnutzen können. Außerdem wird so durch Prefetching ausschließlich im LAN zusätzlicher Traffic erzeugt, nicht aber im Rest des Internets.

Es gibt zwei Varianten:

- Prefetching mit Benachrichtigung über Hits im Cache des Browsers (bzw. des lokalen Proxys)

- Prefetching ohne Benachrichtigung über Hits im Cache des Browsers (bzw. des lokalen Proxys)

Unterschied:

- Prefetching mit Benachrichtigung über Browser-Cache-Hits kann bessere Voraussagen treffen als Prefetching ohne, da es (unter der Annahme, daß bei Einsatz eines lokalen Proxys der Browser Cache ganz abgeschaltet ist) wirklich alle Requests mitbekommt.

Weitere mögliche (pragmatische) Einschränkungen beim Prefetching:

- Prefetching nur für Dokumente bis zu einer bestimmten Größe (hier: 50kB).

  Ein Großteil (ca. 95%) der Dokumente im Web ist kleiner als 50kB (Obwohl das Paper von 1999 ist, sind die verwendeten statistischen Daten von 1996.) und mit der Größe des per Prefetching heruntergeladenen Dokumentes steigt auch die Wahrscheinlichkeit, daß der Benutzer ein Dokument anfragt, bevor das Herunterladen beendet ist. (Die Wahrscheinlichkeit, daß es sich bei dem per Prefetching geholten Dokument um das richtige handelt, bleibt dabei konstant.)

- Verwendung einer oberen Schranke für die Anzahl der pro Vorhersage zum Prefetching zugelassenen Dokumente (hier: 8).

  Damit möchte man das exzessive Prefetching während langer Idle-Zeiten und damit das Überfluten des Browser-Caches sowie die Überbelastung des Proxys verhindern.

2.2.1 PPM Prefetching Algorithmus

Der von Fan et al. (1999) vorgeschlagene PPM-Algorithmus (Prediction by Partial Matching) kennt folgende Parameter:

- *PrefixDepth*: Anzahl der vergangenen Zugriffe, die zur Analyse verwendet werden
- *SearchDepth*: Anzahl der Zugriffe, die der Algorithmus versucht, in die Zukunft zu schauen
- *Threshold*: Wahrscheinlichkeit, ab der Prefetching veranlaßt wird

Funktionsweise:

1. Der Algorithmus vergleicht das letzte, die beiden vorletzten, ... und die *PrefixDepth* letzten zugegriffenen Dokumente des Benutzers mit den statistischen Daten eine entsprechende Datenstruktur und berechnet die Wahrscheinlichkeit aller Dokument, die nach den o. g. Sequenzen innerhalb von *SearchDepth* Schritten vorkamen.
2. Unter diesen Dokumenten werden diejenigen herausgesucht, deren Wahrscheinlichkeit über *Threshold* liegt und die kleiner als 50kB sind.
3. Die so gefundenen Dokumente werden absteigend nach der Länge des gematchten Präfixes sortiert, innerhalb einer Präfixlänge nach der Wahrscheinlichkeit.

PPM Prefetching Algorithmus

Der von Fan et al. (1999) vorgeschlagene PPM-Algorithmus (Prediction by Partial Matching) kennt folgende Parameter:

- *PrefixDepth*: Anzahl der vergangenen Zugriffe, die zur Analyse verwendet werden
- *SearchDepth*: Anzahl der Zugriffe, die der Algorithmus versucht, in die Zukunft zu schauen
- *Threshold*: Wahrscheinlichkeit, ab der Prefetching veranlaßt wird

Funktionsweise:

1. Bei jedem Zugriff wird die History-Struktur aktualisiert. Dies wird im folgenden noch näher erläutert.
2. Der Algorithmus vergleicht das letzte, die beiden vorletzten, ... und die *PrefixDepth* letzten zugegriffenen Dokumente des Benutzers mit den statistischen Daten in einer Menge von Bäumen und sammelt die Werte aller Knoten, die innerhalb einer Tiefe von *SearchDepth* Schritten in den entsprechenden Teilbäumen vorkommen.
3. Anschließend werden alle Werte durch die Wert des Knotens, in dessen Teilbaumes sie gefunden wurden, dividiert. Damit bekommt man die Wahrscheinlichkeit jedes Knotens.
4. Beschreiben mehrere Knoten dasselbe Dokument, so werden ihre Werte aufsummiert.

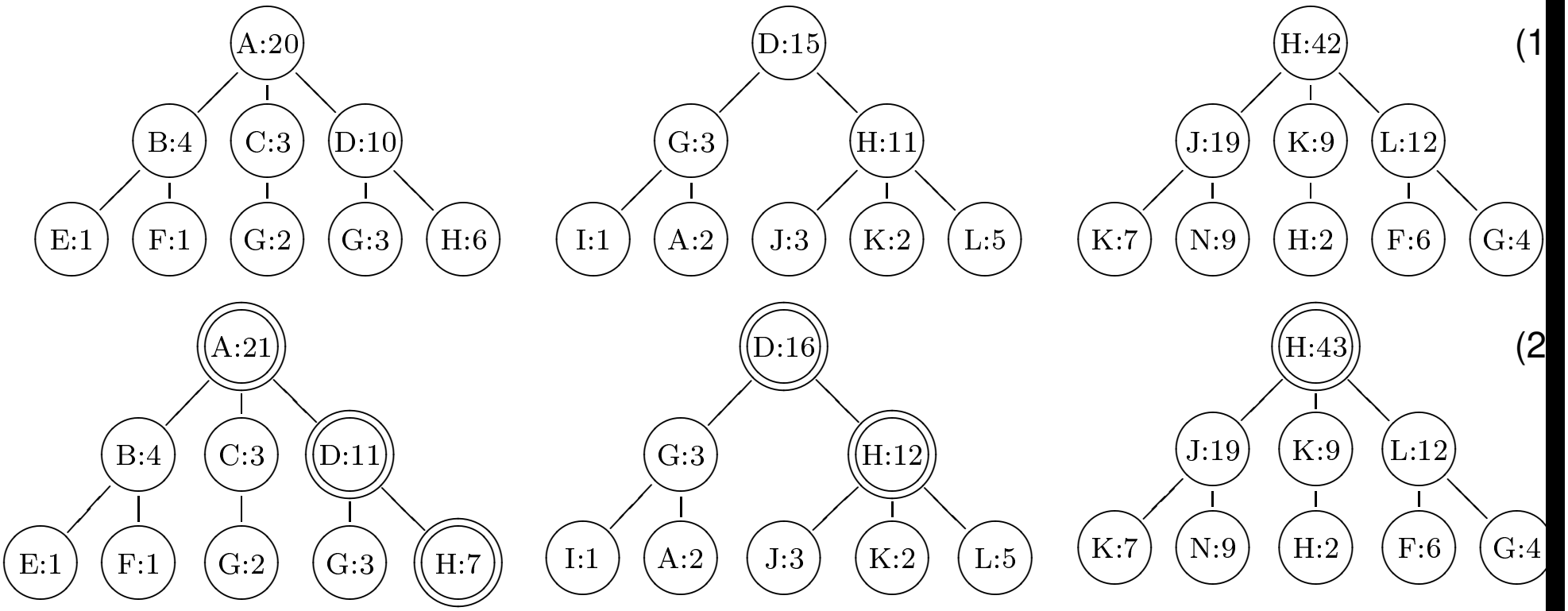
5. Unter diesen Dokumenten werden diejenigen herausgesucht, deren Wahrscheinlichkeit über *Threshold* liegt und die kleiner als 50kB sind.

6. Die so gefundenen Dokumente werden absteigend nach der Länge des gematchten Präfixes sortiert, innerhalb einer Präfixlänge nach der Wahrscheinlichkeit. Davon werden die ersten 8 zum Prefetching vorgeschlagen.

Datenstruktur der für PPM notwendigen History

Für jedes Dokument im Cache existiert ein Baum der Tiefe *K*=*PrefixDepth*+*SearchDepth*, in dem alle *K* Zugriffe nach einem Zugriff auf das entsprechende Dokument gespeichert wird.

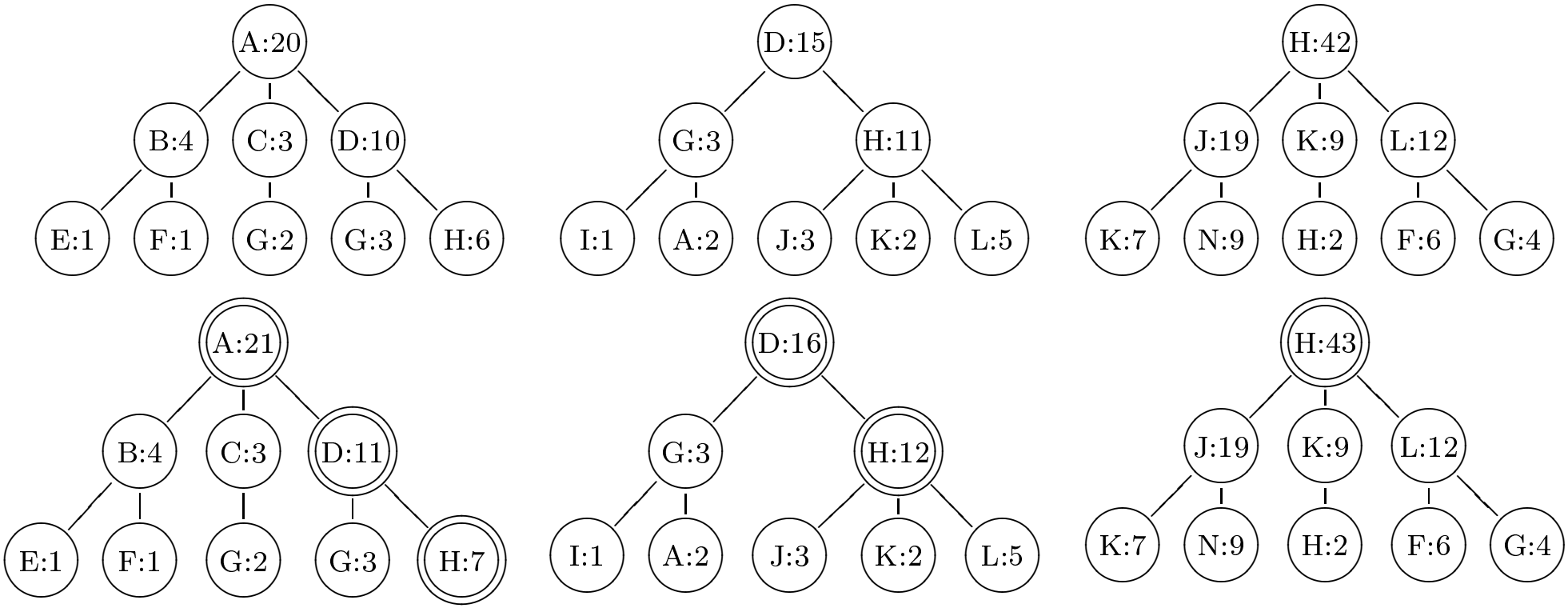
Beispiel: Bäume vor (1) und nach (2) dem Zugriff auf die Dokumentsequenz (*A*, *D*, *H*) bei *K*=3.



Datenstruktur der für PPM notwendigen History

Für jedes Dokument im Cache existiert ein Baum der Tiefe *K*=*PrefixDepth*+*SearchDepth*, in dem alle *K* Zugriffe nach einem Zugriff auf das entsprechende Dokument gespeichert wird.

Beispiel: Bäume vor (1) und nach (2) dem Zugriff auf die Dokumentsequenz (*A*, *D*, *H*) bei *K*=3.



2.2.2 Potential verschiedener Konfigurationen des PPM-Algorithmus

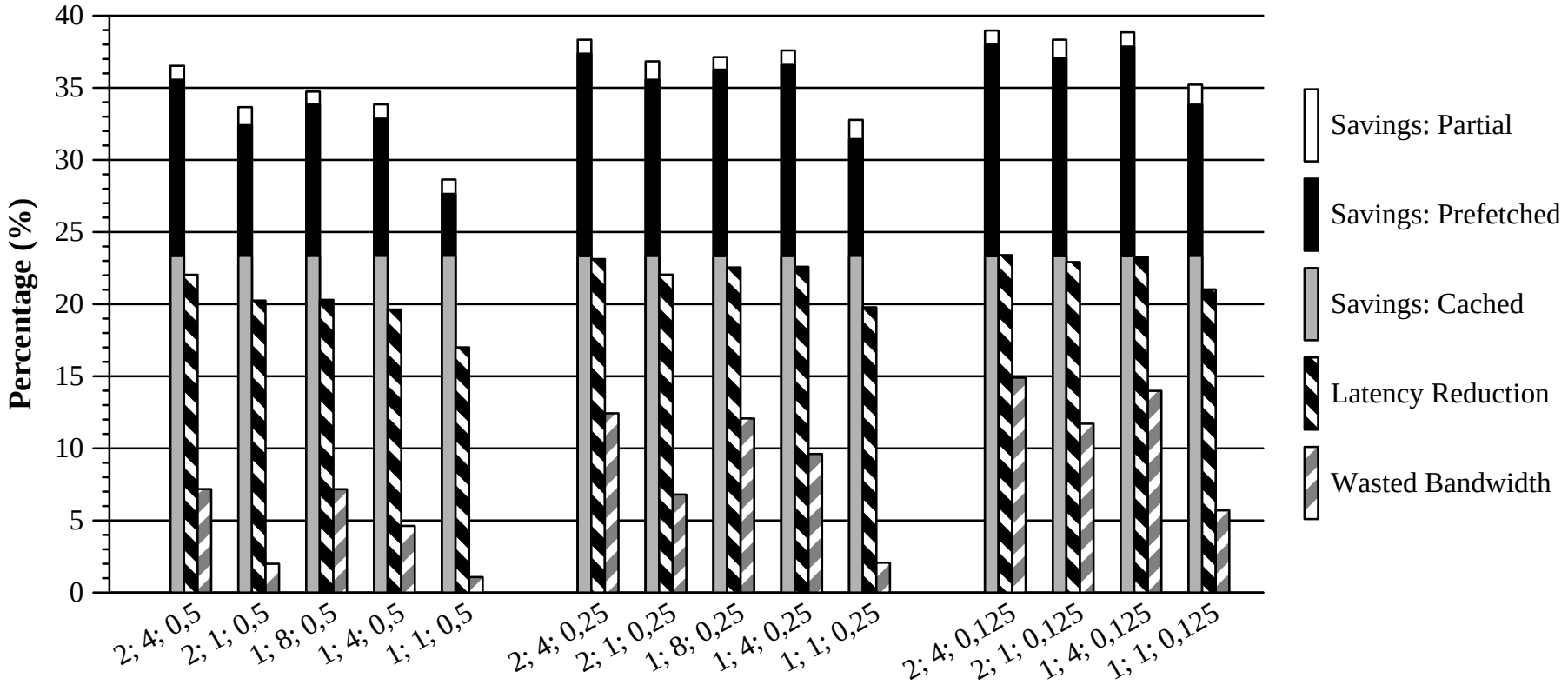


Abbildung 1: Anfrage-Einsparungen, Verzögerungsverringerung und verschwendete Bandbreite des PPM-Algorithmus. Konfigurationen des PPM-Algorithmus dargestellt als Tupel (*Prefix-Depth*, *SearchDepth*, *Threshold*).

Potential verschiedener Konfigurationen des PPM-Algorithmus

Basierend auf Traces über 5 Tage an der University of California, Berkeley wurde die Performance des PPM-Vorhersage-Algorithmus studiert. Es wurden u. a. die Annahmen gemacht, daß jeder Browser einen Cache von 16MB hat und den Least-Recently-Used-Algorithmus verwendet. Weiter wird angenommen, daß der Proxy den Inhalt jedes Browser-Caches kennt und daß bei modifizierten Objekten Prefetching zusammen mit Δ-Codierung verwendet wird.

Es wurden Versuche mit *PrefixDepth* gleich 2 und 1, *SearchDepth* gleich 1, 4 und 8 und *Threshold* gleich 0,5, 0,25 und 0,125 gemacht.

Da die Ergebnisse für *SearchDepth*=8 denen für *SearchDepth*=4 sehr ähnlich waren, werden sie nicht weiter betrachtet.

Abbildung 1: Anfrage-Einsparungen, Verzögerungsverringerung und verschwendete Bandbreite des PPM-Algorithmus. Konfigurationen des PPM-Algorithmus dargestellt als Tupel (*PrefixDepth*, *SearchDepth*, *Threshold*).

Ergebnis:

- Prefetching und Δ-Komprimierung reduzieren die Verzögerung um 17% bis 23,4%.
- Ca. 60% der Anfrage-Ersparnis beruht auf Hits im Browser-Cache, ca. 40% auf Prefetching.

- Insgesamt reduzieren 16MB Browser-Cache und Δ-Komprimierung die Verzögerung um 14,4%, Prefetching sorgt für weitere 9%
- Der zusätzliche erzeugte Traffic reicht von 1% bis 15%

Abhängigkeiten von den Parametern:

***Threshold*:** Allgemein verschwendet ein niedriger *Threshold* mehr Bandbreite als ein hoher. Ist *SearchDepth*=1, verringert ein niedriger *Threshold* aber auch die Verzögerung. Trotzdem scheint für *SearchDepth*$>$1 *Threshold*=0,5 der beste Wert zu sein.

Begründung:

- Bei ansonsten fixen Parametern ergibt ein niedriger *Threshold* mehr mögliche Dokumente.
- Bei *SearchDepth*=1 gibt es nicht sehr viele potentielle Prefetching-Kandidaten und somit ergibt erst ein entsprechend niedriger *Threshold* genügend Kandidaten.

***SearchDepth*:** Die Erhöhung der Suchtiefe verringert die Verzögerung genauso, wie sie mehr verschwendet Bandbreite zur Folge hat. Eine Suchtiefe von 4 scheint sinnvoll, da es darüber kaum noch Änderungen gibt.

***PrefixDepth*:** Die Erhöhung der *PrefixDepth* verringert ebenso die Verzögerung und erhöht die Verschwendung von Bandbreite, jedoch ist dieser Effekt stärker ausgeprägt, wenn der *Threshold* hoch ist. Generell hilft zusätzlicher Kontext.

2.2.3 Potential von Prefetching

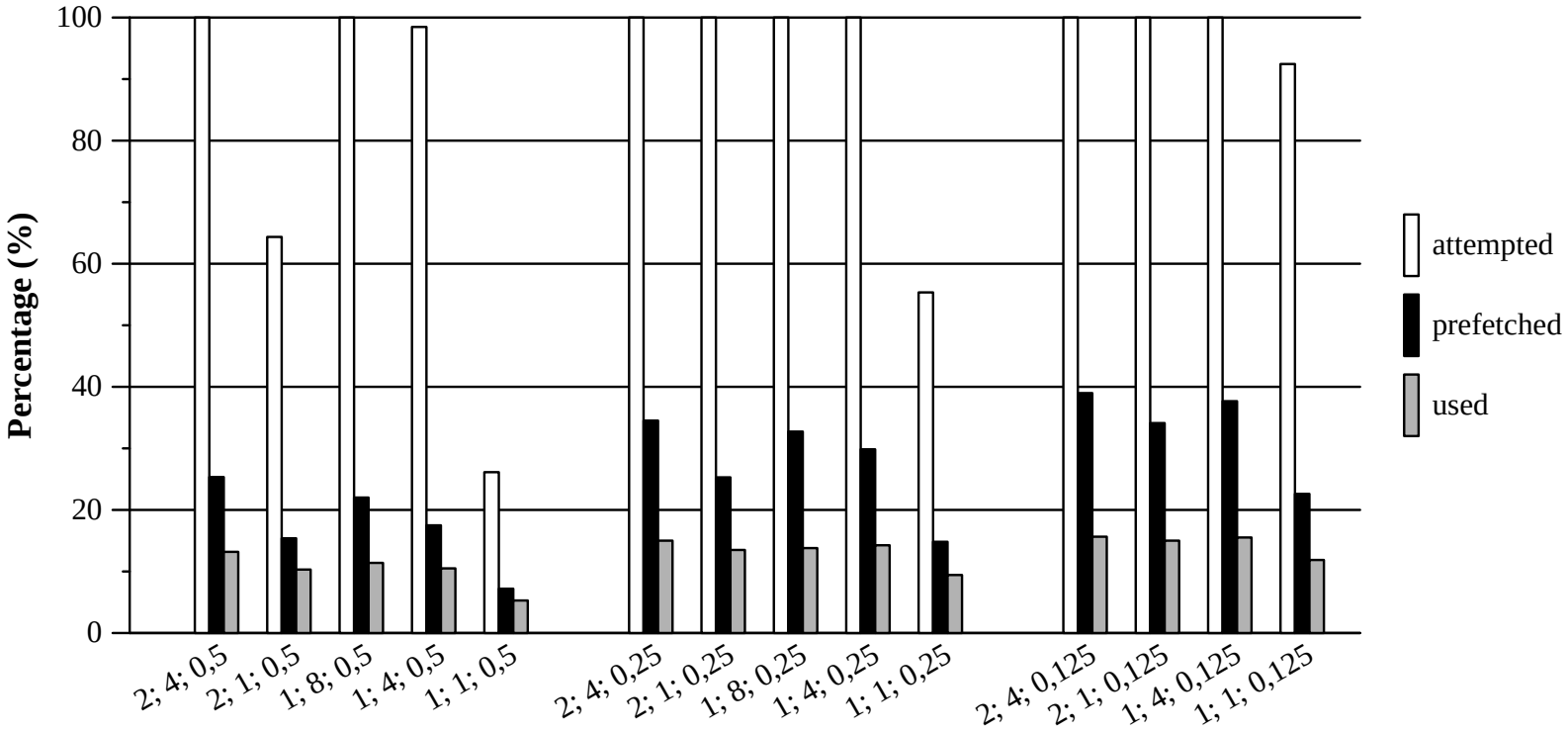


Abbildung 2: Gesamtanzahl an vorgeschlagenen Kandidaten (*attempted*), Anzahl der durchgeführten Prefetching-Vorgänge (*prefetched*), Anzahl der Zugriffe auf Objekte, die per Prefetching in den Browser-Cache kopiert wurden (*used*). Dabei entsprechen 100% der Gesamtanzahl von Anfragen im Trace. Konfigurationen des PPM-Algorithmus dargestellt als Tupel (*PrefixDepth*, *SearchDepth*, *Threshold*).

Potential von Prefetching

Abbildung 2: Gesamtanzahl an vorgeschlagenen Kandidaten (*attempted*), Anzahl der durchgeführten Prefetching-Vorgänge (*prefetched*), Anzahl der Zugriffe auf Objekte, die per Prefetching in den Browser-Cache kopiert wurden (*used*). Dabei entsprechen 100% der Gesamtanzahl von Anfragen im Trace. Konfigurationen des PPM-Algorithmus dargestellt als Tupel (*PrefixDepth*, *SearchDepth*, *Threshold*).

- Aus dem große Unterschied zwischen *attempted* und *prefetched*, daß es meist nicht zum Prefetching aller Kandidaten kam, da dafür meist die Idle-Zeit der Benutzer zu kurz war.
- Das Verhältnis zwischen *used* und *prefetched* ist die Genauigkeit der PPM-Konfiguration. Sie schwankt zwischen 40% (2; 4; 0,125) und 73% (1; 1; 0,5).
- Eine Verringerung des *Threshold* verringert immer die Genauigkeit.
- Es gibt keine direkte Korrelation zwischen der Genauigkeit und der Verzögerungsverringerung.

Empfehlungen:

- Ist Verzögerungsverringerung das höchste Ziel, so ist (2; 4; 0,125) die beste Konfiguration.

- Ist Verzögerungsverringerung bei niedriger Bandbreitenverschwendung das Ziel, so ist (2; 4; 0,5) die beste Wahl.

- Ist die Kapazität des Proxys bezüglicher der History-Struktur beschränkt, so sind (2; 1; 0,25) und (1; 1; 0,125) eine gute Wahl.

2.2.4 Aufschlüsselung von *prefetched* nach Dateitypen

- Ca. 48% GIFs
- Ca. 21% JPEGs
- Ca. 15,5% HTML-Dateien
- Rest andere oder anhand der Traces nicht identifizierbare Dateitypen

Dies legt nahe, daß es sich um eingebettet Grafiken der entsprechenden Webseiten handelte.

2.2.5 Aufschlüsselung der Genauigkeit nach Dateitypen

Ca. 65% GIFs

Ca. 58% JPEGs

Ca. 35% HTML- und andere Dateien

Aufschlüsselung von *prefetched* nach Dateitypen

- Ca. 48% GIFs
- Ca. 21% JPEGs
- Ca. 15,5% HTML-Dateien
- Rest andere oder anhand der Traces nicht identifizierbare Dateitypen

Dies legt nahe, daß es sich um eingebettet Grafiken der entsprechenden Webseiten handelte.
Diese Objekte können laut Fan et al. (1999) auch durch andere Methoden, wie z. B. dem Parsen von HTML-Dokumenten gut gefunden werden.
Allerdings macht dies nur dann Sinn, wenn es sich um eingebette Objekte in für Prefetching vorgesehene HTML-Dokumente handelt, da Browser normalerweise eingebette Objekte sofort nach deren Entdecken anfragen und somit für Prefetching uninteressant machen.

Aufschlüsselung der Genauigkeit nach Dateitypen

Ca. 65% GIFs

Ca. 58% JPEGs

Ca. 35% HTML- und andere Dateien

Da PPM aber (wie leicht klar wird) stark vom Umfang der History abhängt und die oben genannten Werte auf Traces von nur 5 Tagen basieren, sollte sich die Genauigkeit verbessern, sofern man auf Daten von mehreren Wochen oder Monaten zurückgreifen kann.

2.2.6 Auswirkungen von vereinfachten Implementationen

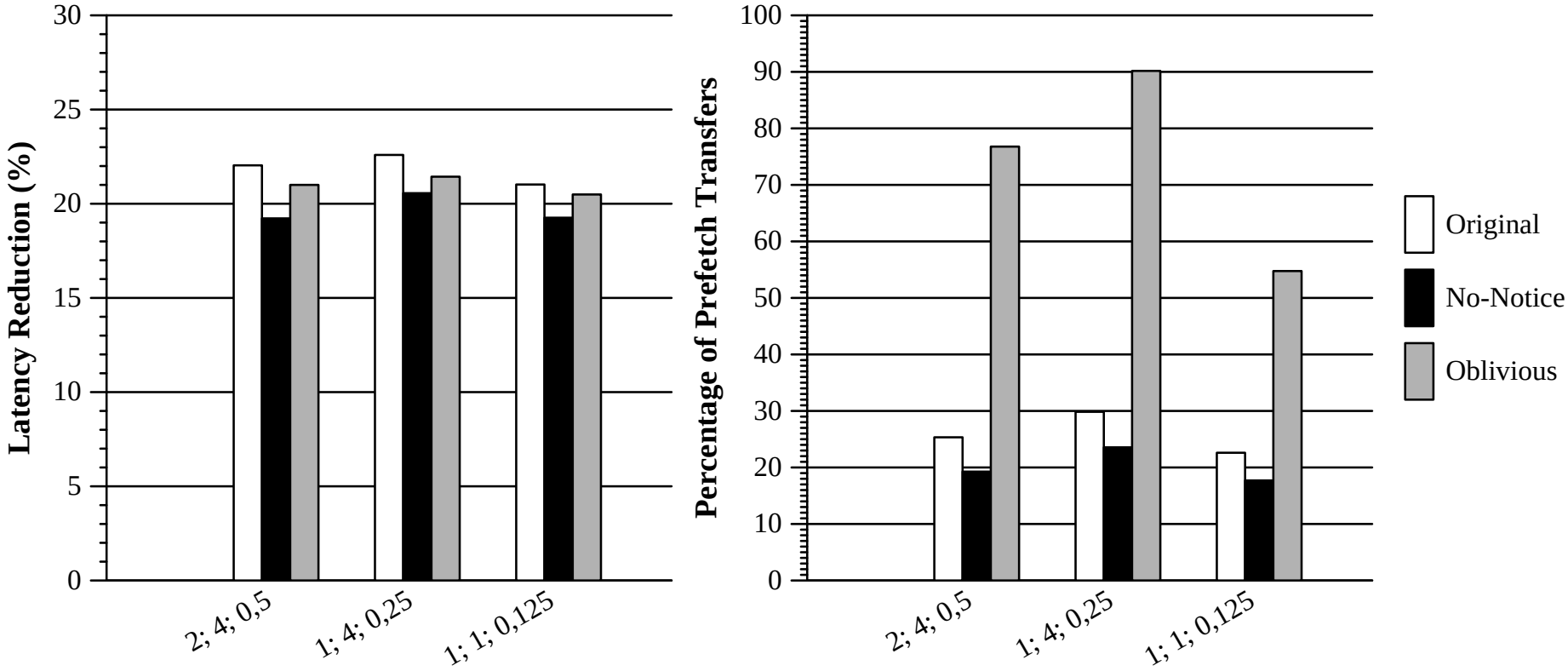


Abbildung 3: Vergleich des ursprünglichen Prefetchings mit Vereinfachungen. Konfigurationen des PPM-Algorithmus dargestellt als Tupel (*PrefixDepth*, *SearchDepth*, *Threshold*).

Auswirkungen von vereinfachten Implementationen (2)

Keine Benachrichtigung über Hits im Browser-Cache („No-Notice“)

Grund: Wesentlich einfacher zu implementieren.

Effekt: Prefetching wird weniger häufig initiiert.

Kein Wissen über Inhalt des Browser-Caches („Oblivious“)

Grund: Um das unnötiges Prefetching von Dokumenten zu verhindern, die bereits im Browser-Cache liegen, sollte der Proxy über den Cache-Inhalt aller seiner Clients Bescheid wissen. Dies ist aber aufgrund des zusätzlichen Speicherverbrauchs als auch von der Implementation her schwierig zu handhaben.

Effekt: Es werden wesentlich mehr Dokumente per Prefetching zum Client übertragen.

Alternative: Proxy-Suggested Prefetching via HTTP Request Piggyback

Auswirkungen von vereinfachten Implementationen

Abbildung 2: Vergleich des ursprünglichen Prefetchings mit Vereinfachungen. Konfigurationen des PPM-Algorithmus dargestellt als Tupel (*PrefixDepth*, *SearchDepth*, *Threshold*).

Keine Benachrichtigung über Hits im Browser-Cache („No-Notice“)

Grund: Wesentlich einfacher zu implementieren.

Effekt: Prefetching wird weniger häufig initiiert.

Keine Wissen über Inhalt des Browser-Caches

Grund: Um das unnötiges Prefetching von Dokumenten zu verhindern, die bereits im Browser-Cache liegen, sollte der Proxy über den Cache-Inhalt aller seiner Clients Bescheid wissen. Dies ist aber aufgrund des zusätzlichen Speicherverbrauchs als auch von der Implementation her schwierig zu handhaben.

Effekt: Es werden wesentlich mehr Dokumente per Prefetching zum Client übertragen. $\Longrightarrow$ Ist also die Verschwendung von Bandbreite kritisch, so muß der vorhersagende Proxy den Inhalt des Browser-Caches kennen.

Um das unnötiges Prefetching von Dokumenten zu verhindern, die bereits im Browser-Cache liegen, sollte der Proxy über den Cache-Inhalt aller seiner Clients Bescheid wissen. Dies

ist aber aufgrund des zusätzlichen Speicherverbrauchs als auch von der Implementation her schwierig zu handhaben.

Alternative: **Proxy-Suggested Prefetching via HTTP Request Piggyback**

Der (Remote-) Proxy teilt dem lokalen Proxy (oder Browser) seine Empfehlungen für Prefetching als Header in der Antwort auf einem HTTP-Request mit (Piggyback = Huckepack), dieser veranlaßt ein Prefetching nur dann, wenn sich das Dokument nicht im lokalen Cache befindet.

3 Web Proxy Caching

Caching mittels Web Proxy reduziert einerseits die Verzögerung von häufig angefragten Daten, andererseits das Datenvolumen der Anbindung des ISP zum Internet. Dazu tragen u. a. folgende HTTP-Header bei:

Pragma: Cache oder No-Cache

ETag: Hash-Wert eines Dokumentes, sollte innerhalb eines Servers eindeutig sein

Expires: Angabe über die Dauer der Gültigkeit eines Dokumentes

Last-Modified: Angabe über Datum der letzten Änderung.

Dynamische Dokumente oder Dokumente, die via *Content Negotiation* ausgesucht werden, machen Probleme beim Caching. Hierbei helfen außerdem noch folgende HTTP-Header:

Vary: z. B. Encoding, Type, Language

Content- & Accept-Type: z. B. text/plain, text/enriched, text/html, text/vnd.wap.wml, application/pdf, etc.

Content- & Accept-Encoding: z. B. gzip

Content- & Accept-Language: z. B. de, en, fr, dk, nl, no

Web Proxy Caching

Caching mittels Web Proxy (z. B. Squid) reduziert einerseits die Verzögerung von häufig angefragten Daten und andererseits das Datenvolumen der Anbindung des ISP zum Internet, da sie meist nur innerhalb eines LANs übertragen werden müssen. Dazu tragen u. a. folgende HTTP-Header bei [Fielding, Gettys, Mogul, Frystyk & Berners-Lee, 1997]

Cache-Control: Cache oder No-Cache

Pragma: Cache oder No-Cache

ETag: Hash-Wert eines Dokumentes, sollte innerhalb eines Servers eindeutig sein

Expires: Angabe über die Dauer der Gültigkeit eines Dokumentes (Datumsangabe, keine Dauer)

Last-Modified: Angabe über Datum der letzten Änderung. Wird bei dynamischen Dokumenten (dazu gelten SSI-Dokumente, die Kopf- und Fußzeile einfügen lassen!) meist *nicht* gesendet, auch wenn vielleicht auf der Webseite ein Datum der letzten Änderung (z. B. sehr leicht zu realisieren mittels SSI) steht!

Dynamische Dokumente oder Dokumente, bei denen der Server das am besten zum Client passende Dokument sendet (*Content Negotiation*), machen Probleme beim Caching. Hierbei helfen außer den o. g. auch noch folgende HTTP-Header:

Vary: z. B. Encoding, Type, Language

Content- & Accept-Type: z. B. text/plain, text/enriched, text/html, text/vnd.wap.wml, application/pdf, application/ms-word, etc.

Content- & Accept-Encoding: z. B. gzip

Content- & Accept-Language: z. B. de, en, fr, dk, nl, no

3.1 Cooperative Web Proxy Caching

Cooperative Web Proxy Caching ist die Zusammenarbeit von zwei oder mehreren Web Proxy Caches. Grundgedanke: Es kann immer noch schneller sein, seinen Cache-Kollegen zu fragen, als den eigentlich angefragten Server.

Wolman et al. (1999) versuchen anhand Auswertung von gleichzeitigen Traces an der *University of Washington* und *Microsoft* die Nützlichkeit von *Cooperative Web Proxy Caching* in der Praxis zu ermitteln. Sie kamen dabei zu folgenden Ergebnissen:

- Cooperative Web Caching ist nur für kleine Organisationen (bis ca. 2500 Nutzer) nützlich und liegt damit in einem Bereich in dem es technisch (aber eben meist nicht politisch oder geographisch) auch ein einzelner, großer Proxy Cache täte.
- Bei großen Organisationen reduziert Cooperative Web Caching die Verzögerung nur marginal: „Unbeliebte Dokumente sind universell unbeliebt.“

Cooperative Web Proxy Caching

Cooperative Web Proxy Caching ist die Zusammenarbeit von zwei oder mehreren Web Proxy Caches, da es immer noch schneller sein kann, seinen Cache-Kollegen zu fragen, als den Server.
[Wolman et al., 1999] versucht anhand Auswertung von gleichzeitigen Traces (Dauer: 1 Woche) an der *University of Washington* und *Microsoft* die Nützlichkeit von *Cooperative Web Proxy Caching* in der Praxis zu ermitteln.
Dabei werden vor allem 2 Dinge analysiert:

1. Die Kooperation lauter kleiner Caches in jedem der ca. 200 Teilorganisationen der *University of Washington* (Institute, Fachbereiche, Verwaltung, etc.) mit jeweils kleinen Populationen (100 bis 1000 Benutzern).

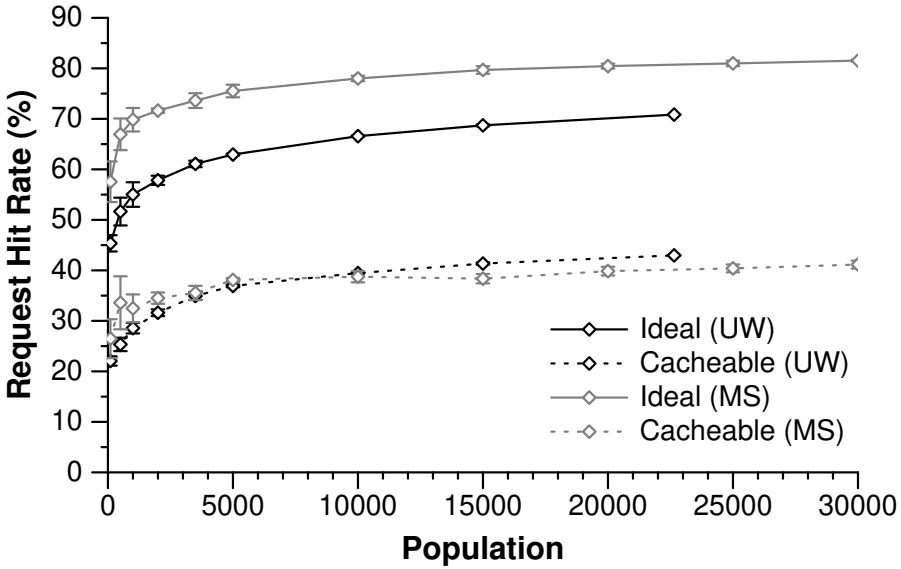
2. Die Kooperation zwei großer Organisationen (eben der *University of Washington* und *Microsoft*) mit jeweils sehr großen Populationen (jeweils mehreren 10 000 Benutzer).

Sie kamen dabei zu folgenden Ergebnissen:

- *Cooperative Web Proxy Caching* ist nur für kleine Organisationen (bis ca. 2500 Nutzer) nützlich und liegt damit (bei den verwendeten Traces) in einem Bereich in dem es technisch (aber eben meist nicht politisch oder geographisch) auch ein einzelner, großer Proxy Cache täte.

- Bei großen Organisationen reduziert *Cooperative Web Proxy Caching* die Verzögerung nur marginal, da es eine große Überschneidung zwischen den beliebtesten Dokumenten in den Organisationen (im Beispiel 33% der ersten 1000 Dokumente) gibt und „unbeliebte Dokumente universell unbeliebt sind“.

Cooperative Web Proxy Caching (2)



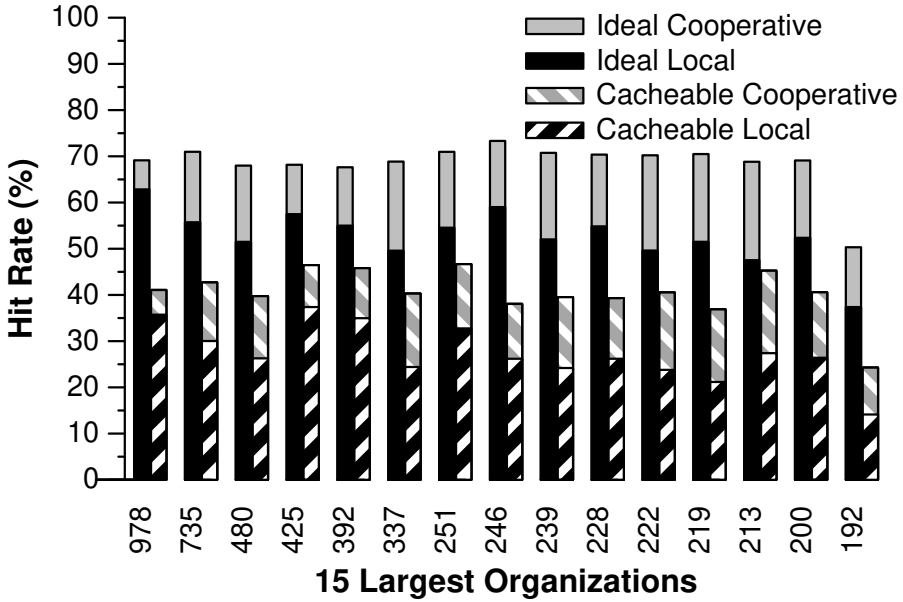


Abbildung 4: Proxy Anfrage-Trefferquoten als Funktion der Größe der Benutzerpopulation sowie lokale und globale Proxy Anfrage-Trefferquoten für die 15 größten Suborganisationen der *University of Washington*

Cooperative Web Proxy Caching (2)

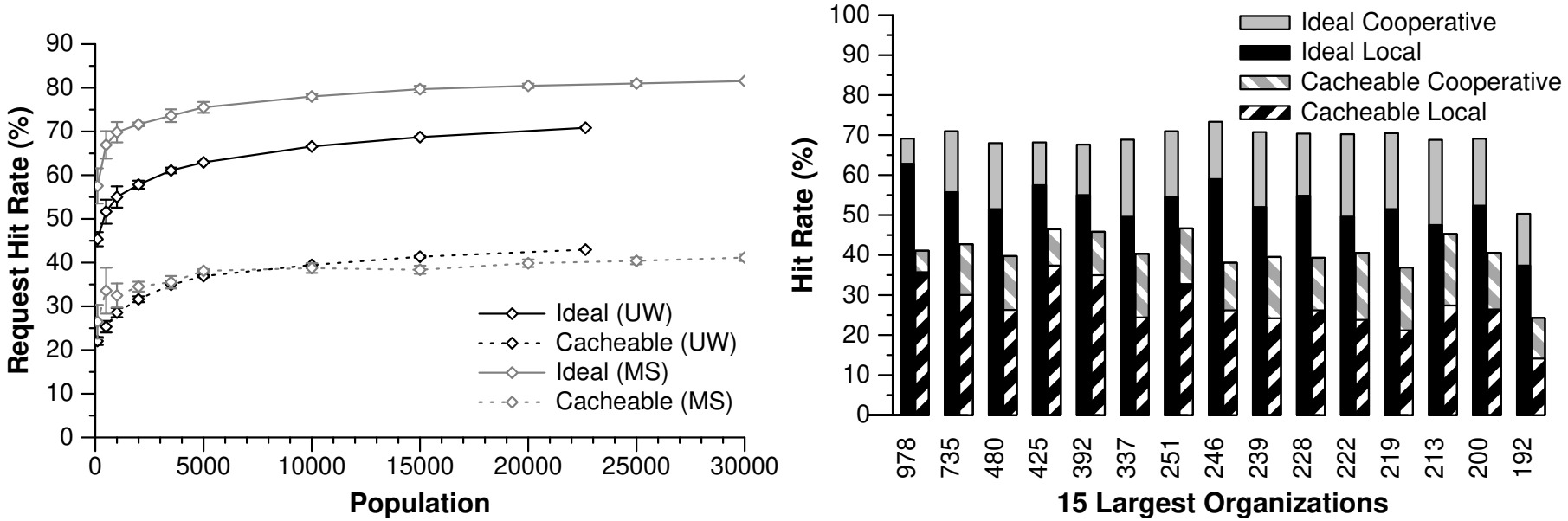


Abbildung 5: Proxy Anfrage-Trefferquoten als Funktion der Größe der Benutzerpopulation sowie lokale und globale Proxy Cache-Trefferquoten für die 15 größten Suborganisationen der *University of Washington*

In der linkem Grafik sieht man schön das „Knie“ bei ca. 2500 Benutzern.
Unterschiede des Benutzerverhaltens an der *University of Washington* und bei *Microsoft*:

- Die schwarze Idealkurve (*Microsoft*) ist um ca. 13% höher als die schwarze (*University of Washington*) $\Longrightarrow$ Das „Document Sharing“ der Benutzer ist bei *Microsoft* wesentlich höher als an der *University of Washington*. $\Longrightarrow$ Die Benutzer bei *Microsoft* verhalten sich wesentlich homogener als die der *University of Washington*, was auch nicht sehr verwunderlich ist.

- Auf der anderen Seite unterscheiden sich die beiden „Cacheable“-Kurven nur sehr wenig, was zusammen mit der vorherigen Beobachtung den Schluß zuläßt, daß die Anfragen bei *Microsoft* wesentlich häufiger (hier 49%) nicht-cachebare Dokumente (hauptsächlich dynamische Seiten) betreffen als an der *University of Washington* (40%). $\Longrightarrow$ Trotz der größeren Benutzeranzahl bei *Microsoft* ist die Cache-Trefferquoten nicht besser als die der *University of Washington*.

Obwohl die Theorie diese Ergebnisse teilweise schon vorausgesagt hatte ($\Longrightarrow$ die Trefferquote eines Cache steigt logarithmisch mit der Anzahl der Anfragen und der Anzahl seiner Benutzer), sind die Ergebnisse bezüglich der Nützlichkeit von *Cooperative Web Proxy Caching* mit Vorsicht zu genießen:

- Wie wir gerade gesehen haben, hängen Cache-Trefferquoten allgemein sehr von der Homogenität der Benutzer sowie ihrem Verhalten ab. Dies läßt vermuten, daß es auch zu anderen Ergebnissen als denen, die hier gezeigt wurden, kommen kann.

- Nach eigener Aussage war [Wolman et al., 1999] die erste praxisnahe Untersuchung von *Cooperative Web Proxy Caching* in großem Umfang. Weiter gingen die Traces nur über eine knappe Woche. Die statistische Signifikanz dieser Ergebnisse ist also nicht sehr hoch.

- Diese Ergebnisse geht von optimalen Bedingungen aus (keine Verwendung des Browser Caches während der Traces, beliebig viel Speicherplatz in allen Proxy Caches). Allerdings wurde ein Unterschied zwischen idealem Caching (alle Dokumente können im Cache gespeichert werden) und Caching von Dokumenten, bei denen Caching möglich ist, gemacht.

3.2 HTML Macro-Preprocessing to Support Dynamic Document Caching

[Douglis, Haro & Rabinovich, 1997]

- Dynamische Dokumente sind abhängig von mindestens einem der folgenden Faktoren:
  - Datum
  - Uhrzeit (z. B. Wettervorhersage, Nachrichten)
  - Cookies (z. B. viele Online-Shops, Slashdot & Co.)
  - Referer-Header
  - lokalen Daten(banken) auf dem Server

  Dementsprechend können sie normalerweise nicht im Cache gespeichert werden.

- In vielen Fällen hilft das Offenlegen der internen Struktur dynamischer Seiten, die statischen Elemente dieser Seiten (Layout von Tabellen, Menüs, Kopf- und Fußzeilen, etc.) zu cachen, so daß nur noch die dynamischen Daten vom Server geholt werden müssen.

HTML Macro-Preprocessing to Support Dynamic Document Caching [Douglis et al., 1997]

- Dynamische Dokumente sind abhängig von mindestens einem der folgenden Faktoren:
  - Datum
  - Uhrzeit (z. B. Wettervorhersage, Nachrichten)
  - Cookies (z. B. viele Online-Shops, benutzerabhängige Portalseiten wie Slashdot, MyOpera, MyYahoo! und wie die ganzen MyWeißgottwas-Seiten alle heißen.$\Longrightarrow$ Dies sind insbesondere viel sehr beliebte Seiten.)
  - Referer-Header
  - lokalen Daten(banken) auf dem Server

  Dementsprechend können sie normalerweise nicht im Cache gespeichert werden.

- In vielen Fällen hilft das Offenlegen der internen Struktur dynamischer Seiten, die statischen Elemente dieser Seiten (Layout von Tabellen, Menüs, Kopf- und Fußzeilen, etc.) zu cachen, so daß nur noch die dynamischen Daten vom Server geholt werden müssen. $\Longrightarrow$ Das Macro-Processing kann dann im Proxy oder im Browser passieren, je nachdem, ob der Browser HPP unterstützt. Ob dies der Fall ist, kann z. B. mit dem HTTP-Header „Accept-Encoding“ übermittelt werden.

- while- und foreach-Schleifen können außerdem zur Komprimierung der statischen Daten beitragen, z. B. bei Auflistungen von Suchergebnissen oder bei Newstickern wie Slashdot.

- Oft werden die Daten von beliebten Seiten bereits im RDF-Format (Netscape-Entwicklung) angeboten, dies dient aber nahezu ausschließlich dazu, um die Kurztitel der aktuellen Artikel auf anderen Seiten in kleinen Boxen anzeigen zu können. Beispiel: In einem Kasten bei Slashdot werden die neusten Programme bei Freshmeat, der Wired-Newsticker, die aktuellsten Linux-Kernelversionen und vieles, vieles mehr angezeigt.

4 Literatur

Douglis, F., Haro, A. & Rabinovich, M. [1997]. „HTML Macro-Preprocessing to Support Dynamic Document Caching.“ In *Proceedings of USENIX‘97.* (Available from http://www.research.att.com/~douglis/)

Fan, L., Cao, P., Lin, W. & Jacobson, Q. [1999]. „Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performances.“ In .

Fielding, R., Gettys, J., Mogul, J. C., Frystyk, H. & Berners-Lee, T. [1997, Jan]. *RFC 2068: Hypertext Transfer Protocol – HTTP/1.1.*

Mogul, J. C., Douglis, F., Feldmann, A. & Krishnamurthy, B. [1997]. „Potential benefits of delta encoding and data compression for http.“ In *Proceedings of ACM SIGCOMM‘97.* (Available from http://www.research.att.com/~douglis/)

Wolman, A., Voelker, G. M., Sharma, N., Cardwell, N., Karlin, A. & Levy, H. M. [1999]. „On The Scale and Performance of Cooperative Web Proxy Caching.“ In *Operating Systems Review* (Vol. 34, Seite 16-31).

Literatur

Hauptquellen waren [Fan et al., 1999] und [Wolman et al., 1999].