WebCompanion: A Friendly Client-Side Web Prefetching Agent

Reinhard P. Klemm

Bell Laboratory, Lucent Technologies
(Distributed Software Research Department)

IEEE TKDE, July/August 1999

Outline

• Introduction
• WebCompanion Features
• Experimental Results
• Summary
Introduction

• Motive
  ▪ Deal with the Web latency and bandwidth issues
  ▪ Web prefetching strategies
    ✓ client-side, proxy-side, server-side, hybrid
  ▪ Greedy prefetching strategy
    ✓ high network overheads and resource consumption
• Goal
  ▪ Reduce the round trip time of accesses to the Web
  ▪ Client-side Java-implemented prefetching agent
    ✓ estimate the round trip times, limit the overhead

• Main Idea
  ▪ Estimate the round trip times of all documents referenced by embedded hyperlinks
  ▪ Prefetch the documents with the longest times first
• Estimated Round-Trip Time-based Prefetching
  ▪ Highly selective prefetching strategy
    ✓ only long retrieval latencies and low resource usage
  ▪ Sophisticated session control scheme
    ✓ adapt to changing network and server conditions
  ▪ Startup prefetching, DNS caching
Introduction

- Performance Gains
  - Average speed-up > 50%
  - Average network byte overhead < 150%
- Architecture
  - HTTP Web proxy

WebCompanion Features

- Server Statistics Cache
  - Linearly weighted averages for individual servers
    - setup time: $t_s$
    - waiting time: $t_w$
    - byte transmission time: $t_b$
    - resource size: $s_r$
    - round trip time: $t_r = t_s + t_w + t_b \times s_r$

\[
an = \frac{1 \times y_1}{n(n+1)/2} + \frac{2 \times y_2}{n(n+1)/2} + \ldots + \frac{n \times y_n}{n(n+1)/2}
\]

\[
a_n = \frac{n-1}{n+1} \times a_{n-1} + \frac{2}{n+1} \times y_n
\]

\[
a_n = \omega \times a_{n-1} + (1-\omega) \times y_n
\]
WebCompanion Features

- Estimation of Round Trip Time (RTT)
  - Identify embedded hyperlinks
  - Examine document cache
    - if found, no prefetching operation is activated
  - Access server statistics cache
    - if not found, fetch the referenced document and update both caches
  - Compare $t_s + t_w$ with $t_s + t_w + t_b *s_r$
    - if not significantly less, store the estimated $t_r$
  - Issue HEAD request to obtain status information
    - if size $s$ is returned, compute and store $t_s + t_w + t_b *s$

WebCompanion Features

- Hyperlink Filters (before prefetching)
  - Protocol filter
    - compressed files, images, executables…
  - Dynamic resource filter
    - the output of CGI scripts
  - Size filter
    - maximum size threshold
  - Time filter
    - maximum/minimum RTT threshold
      - 75% linearly weighted average of previous accesses
      - (min, max): (3000 ms, 20000 ms)
WebCompanion Features

- Session Control
  - Start prefetching
    1. first request arrives ⇒ the beginning a new session
    2. new request arrives ⇒ examine the ongoing session
  - Stop prefetching
    1. restart a new session with a short delay (500 ms)
    - wait for requests to reconsider the end of session

WebCompanion Features

- Implementation
  - Memory cache with compression
    1. 2MB cache with LRU replacement policy
    2. 65.44% cache hit ratio
      - RTT vs. priority to be displaced
  - DNS caching
    1. influence on the pessimistic scenario
  - Parallel prefetching
  - GUI on statistics and cache
Experimental Results

- **WebWatch**
  - Workload generator (browser emulator)
    - URL list with a configure probability distribution
      - no duplicate requests/no cache
    - switch between WebCompanion and direct access
  - Performance measurement
    - pessimistic access pattern
      - simulate user idle time: fixed interval (5 sec)
    - average-case access pattern
      - simulate user idle time: Poisson distribution ($\lambda=10$)
      - randomly select hyperlinks: equal probability (80%)

---

Comparison

- **Comparison**
  - Speedup
  - Percentage of faster accesses
  - Network overhead

![Comparison Between Direct Access and WebCompanion Access: Averages Are Per Web Resource](image)

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Average Direct Access Time (ms)</th>
<th>Average WebCompanion Access Time (ms)</th>
<th>Speedup (%)</th>
<th>% of Accesses Faster with WebCompanion Access</th>
<th>Network Overhead (bytes) %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pessimistic Scenario</td>
<td>6711</td>
<td>6207</td>
<td>7.5</td>
<td>56.1</td>
<td>82.3</td>
</tr>
<tr>
<td>Average Case</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Direct Access First</td>
<td>4501</td>
<td>1939</td>
<td>60.43</td>
<td>80.8</td>
<td>205.22</td>
</tr>
<tr>
<td>2. WebCompanion Access First</td>
<td>2630</td>
<td>1514</td>
<td>42.42</td>
<td>74.3</td>
<td>112.76</td>
</tr>
</tbody>
</table>
Experimental Results

Access Time Differences in Pessimistic Scenario

Experimental Results

Distribution of Access Time Differences in Pessimistic Scenario

DNS caching
Experimental Results

Access Time Differences for Average-case Access Pattern: Direct Access First, Web Companion Access Second

User idle time is longer
Experimental Results

Access Time Differences for Average-Case Access Pattern; WebCompanion Access First, Direct Access Second

Average access time is shorter
Summary

- **Advantage**
  - It can deal with pages that have not been visited
  - It takes the round trip time into consideration
  - It exhibits 7.5% speedup in the pessimistic case
  - It exhibits 50% speedup in the average case

- **Weakness**
  - Average deviation of RTT=54.9%
  - Error-prone decision of a new session
  - Local cache on the browser must be turned off
  - Network and server caching effects cannot be isolated