Trading off Quality for Throughput Using Content Adaptation in Web Servers

Michael Gopshtein
Dror Feitelson
Hebrew University, Jerusalem
CNN on 9/11

• Load grew from 85,000 hits/sec to 229,000 hits/sec in 15 minutes
• Hosting service diverted additional servers: grew from 10 to 52
• Shut down monitoring service
• Reduce content to image, logo, and 1247 bytes of HTML
CNN on 9/11

• Load grew from 85,000 hits/sec to 229,000 hits/sec in 15 minutes
• Hosting service diverted additional servers: grew from 10 to 52
• Shut down monitoring service
• Reduce content to image, logo, and 1247 bytes of HTML
Brewer’s DQ Principle

\[ D \times Q = \text{const.} \]

Data per query \( \rightarrow \) \text{quality}

Queries per second \( \rightarrow \) \text{throughput}
Previous Work

• Switching between versions of a site
  – Create a parallel degraded content tree
    o Remove decoration images
    o Compress remaining images
    o Integrate style files and JavaScript
    o Question of doing this automatically
  – Monitor load conditions
    o E.g. response time or server utilization
  – Switch between versions based on load
    o Simply switch root directory of server
Previous Work

• Reported improvements of
  – Up to factor of 2
  – Up to factor of 7 (for very large files)
  Up to factor of 2 (for < 64 KB)
Previous Work

• Automatic content adaptation to support mobile devices
  – Device-specific CSS or layout
• Becoming increasingly important in recent years
• Different emphasis than in our context
Our Contributions

- Describe full implementation on Apache
  - Normal or optimized mode
  - Triggering mode switch
  - Simple integration using existing facilities
- Detailed experimental evaluation
  - Achieved throughput improvement of 2-4
    (and up to 10 with more extreme adaptation)
  - Speed of mode switch
Our Contributions

• Describe full implementation on Apache
  – Normal or optimized mode
  – Triggering mode switch
  – Simple integration using existing facilities

• Detailed experimental evaluation
  – Achieved throughput improvement of 2-4
    (and up to 10 with more extreme adaptation)
  – Speed of mode switch
Serving Optimized Content

• Optimized content tree mirrored under /opt
• Use Apache mod_rewrite to re-write URL and select appropriate version
• selection rules:
  1. Optimized mode flag is on, implemented as existence of a special file called opt.do
  2. Optimized version of requested file exists and is up to date
Serving Optimized Content

- Mod_rewrite overhead:
Performance Indicators

- CPU utilization
  - 5-second average using sar utility
- Number of incoming TCP connections
  - Available from /proc/net/snmp
- Number of idle Apache processes
  - Available using Apache mod_status
Switching Algorithm

• In normal mode:
  – Keep track of “safe” TCP connection rate
  – If CPU utilization > 85%, switch to optimized

• In optimized mode:
  – Switch to normal if 3 conditions are met:
    1. CPU utilization < 85%
    2. There is at least one idle process
    3. Connection rate is lower than recorded “safe” rate
  – Reduce “safe” rate by 30%
Implementation

- External perl script runs every 5 seconds
- Collects the required statistics
- Creates or removes special file opt.do
- Apache configured with mod_rewrite to look for this file
Our Contributions

• Describe full implementation on Apache
  – Normal or optimized mode
  – Triggering mode switch
  – Simple integration using existing facilities

• Detailed experimental evaluation
  – Achieved throughput improvement of 2-4
    (and up to 10 with more extreme adaptation)
  – Speed of mode switch
Experimental System

- Apache web server with copy of top500 site
  - Configure with important non-default settings
    - Enable keep_alive connections (1000, 10 sec timeout)
    - Enable in-memory caching (4MB)
    - Enable large number of active processes (1500)
- Several client machines with HP LoadRunner
  - Play scripts of HTTP requests simulating users
  - Loop to simulate requests from additional users
  - List of requests can be manipulated to simulate requests in optimized mode
Response Time

- Closed-system scenario
- Optimized version reduces 43% of requests and 22% of bytes
- Normal: 270 ms
- Optimized: 138 ms
- Demonstrates need for enhanced Apache settings
Slashdot Scenario

- All users request the same page
- Open-system with no feedback effects

<table>
<thead>
<tr>
<th></th>
<th>Normal</th>
<th>Moderate opt</th>
<th>Extreme opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP reqs</td>
<td>59</td>
<td>29</td>
<td>3</td>
</tr>
<tr>
<td>KB</td>
<td>322</td>
<td>188</td>
<td>65</td>
</tr>
<tr>
<td>Connections</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Extreme Optimization

- Leave only logo and story image
- Compress them
- Remove CSS and JS, integrate minimal needed parts
Extreme Optimization

- Leave only logo and story image
- Compress them
- Remove CSS and JS, integrate minimal needed parts
Transaction Success Rate

Transaction Success Percentage (regular)

Transaction Success Percentage (optimized)

Percentage (extreme)
Throughput

![Graph showing transaction success percentage for different scripts with optimized and extremely optimized scripts reaching 100%.]
Throughput
Mode Switch

- Load: create step in number of requests
  - From 15 to 85 requests per sec
- Causes near-immediate switch to optimized mode
- And actual drop in CPU utilization and response time
Conclusions

• Technical aspects of content adaptation are easy
  – Monitoring using available hooks
  – Switching with mod_rewrite
• Increase throughput by 2-4 with little degradation
• Increase by up to 10 possible
• Larger increase probably requires other techniques
• Automatic creation of adapted content requires more work
• Need for careful experimental evaluation