Handling Persistent Connections in Overloaded Web Servers

Thiemo Voigt\textsuperscript{a}
Swedish Institute of Computer Science (SICS), Uppsala University

Per Gunningberg
Uppsala University

\textsuperscript{a}This work is partially funded by the national Swedish Real-Time Systems research initiative ARTES (www.artes.uu.se).
Motivation

• web server overload can cause unpredictable service

• web server admission control and service differentiation schemes deployed

• requests accepted/rejected based on information found in HTTP header

• requests rejected when queue lengths exceed thresholds or when requests are not compliant with token bucket based policers
Persistent Connections

(a) no persistent connections

(b) persistent connections
Motivation

- persistent connections reduce client latency and server overhead
- resource demands of future requests on persistent connections unknown when admission decision is made
- first request does not reveal information about amount of work entering the system
- too optimistic $\rightarrow$ overload risk
  too conservative $\rightarrow$ unnecessary rejections

Goal of this work:
avoid uncontrollable overload while maximizing access
Outline

• Approach for dealing with persistent connections
• Kernel-based architecture
• Experiments to evaluate the approach
• Conclusions
Approach

- admission control of first request using token bucket based policers
- no control of following requests on same persistent connection
- during overload: reset “unimportant” connections, but keep alive important connections
- use cookies to code importance of connections
Why Cookies?

- all information to determine importance of connections embedded in cookie.

- established technique, widely used

- can contain long-lasting information, e.g. identify preferred customers.

- no changes to the web server or clients (browsers) necessary.
Cookie Example

Client one

GET (no cookie)

GET (no cookie)

Req. Object

RST

Client two

GET Cookie: val=import.

(1) Req. Object

GET Cookie: val=import.

Req. Object

(2)

Server

(1) normal condition

(2) overload
Kernel-based Architecture

web application

Apache server: non–static content

non–static requests

KHTTPD: static content, HTTP header parsing

HTTP requests

KHTTPD: parsing HTTP header static content, HTTP requests non–static

Apache non–static content

KHTTPD

QoS, overload and persistent connection control module

user space

kernel space

non–static requests

reject accept

URLs cookies

HTTP requests
Kernel-based Architecture

• cookie-based connection control active when CPU-utilization is higher than threshold

• when cookie-based connection control is active:
  “right cookie” $\rightarrow$ keep connection alive
  else $\rightarrow$ abort connection

• approach works as well in middleware or user-space
Experimental Setup

- isolated network, server and hosts connected via switch
- client populations emulated using one traffic generator (S-Client) instance per client population
- client population requests one or more files on same persistent connection (“session”)
- some of the requests carry “right” cookie in HTTP headers (“protected requests”)
Experiments: Overload Control

two emulated client populations:

- (client\_pop\textsubscript{one})’s sessions: 6 requests (last 4 protected), 50 sessions/sec requested

- Bursty population: sessions start with same first request (treated same way as client\_pop\textsubscript{one}’s first request), plus more requests (unprotected) which causes server overload

<table>
<thead>
<tr>
<th>cookie control enabled</th>
<th>throughput non-bursty client pop.</th>
<th>session compl. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>40.3 sess/sec</td>
<td>9.9 sec</td>
</tr>
<tr>
<td>yes</td>
<td>37.8 sess/sec</td>
<td>2.1 sec</td>
</tr>
</tbody>
</table>

Overload protection using cookie control
Experiments: Service Differentiation

three emulated client populations:

- (client_pop_{pref})’s sessions: 6 requests (all reqs. protected)
- (client_pop_{two})’s sessions: 6 requests (last 4 reqs. protected)
- Bursty population causing overload

<table>
<thead>
<tr>
<th>Throughput (sessions/sec)</th>
<th>client_pop_{pref}</th>
<th>client_pop_{two}</th>
</tr>
</thead>
<tbody>
<tr>
<td>overall</td>
<td>25.3</td>
<td>19.4</td>
</tr>
<tr>
<td>during bursts</td>
<td>21.1</td>
<td>9.4</td>
</tr>
<tr>
<td>during non-bursts</td>
<td>28.0</td>
<td>24.8</td>
</tr>
</tbody>
</table>

Service Differentiation using cookie-based control
Limitations

• KHTTPD handles static requests only

• simple string comparison ($strmp$)

• turning cookie-based control on/off can potentially lead to oscillations

• there need to be “unimportant” sessions

• users might disable cookies
Conclusions

- problem of unknown resource demand of persistent connections
- architecture that protects web servers from overload
- importance of persistent connections encoded in cookies
- experiments that show that the approach can prevent uncontrollable server overload and provide service differentiation
# Filter Rules

<table>
<thead>
<tr>
<th>URL</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>noaccess</em></td>
<td>(&lt;\text{drop}) &gt;</td>
</tr>
<tr>
<td>/index.html</td>
<td>(&lt;\text{rate}=50, \text{burst}=5&gt;)</td>
</tr>
<tr>
<td>/cgi-bin/*</td>
<td>(&lt;\text{rate}=10, \text{burst}=2&gt;)</td>
</tr>
</tbody>
</table>

application-level filter rules

(HTTP header-based connection control)

<table>
<thead>
<tr>
<th>cookie</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>val=important</td>
<td>keep-alive</td>
</tr>
<tr>
<td>client=Fred</td>
<td>keep-alive</td>
</tr>
</tbody>
</table>

cookie rules

(cookie-based persistent connection control)
Related Work

• **WebQoS** [1999]: middleware for service differentiation. Cookies used to identify sessions.

• Bhatti *et al.* [2000] use cookies to code durations of sessions

• Cherkasova and Phaal [1999] deploy admission control with aim to allow session completion

• Aron *et al.* [1999] support persistent connections in cluster-based web servers

• Almeida *et al.* [1999]: preferred requests executed as higher priority processes