Approaches to Support Differentiated Quality of Web Services

Ryu Sook Hyun
Email: shryu@postech.ac.kr
DP&NM Lab.
POSTECH
Contents

• Introduction
• Differentiated Web Server
  – General Web Server & Differentiated Web Server
  – Related Work
• Requirements
  – Functional Requirements
  – Non-Functional Requirements
• Design
  – System Architecture
  – Realtime Kernel
  – Classification Approach
  – Priority driven Scheduling Methods
• Implementation
• Performance Evaluation
• Conclusion & Future Work
• References
Introduction (1)

- The WWW is rapidly increasing, with the number of users expected to reach 320 million by the year 2002.
- The diversity of Web applications continues to increase.
  - The quality of service is an increasingly critical issue in Web services.
- QoS requirements
  - Refer to non-functional requirements, such as performance or availability requirements.
  - Deal with how an application or service will behave at run-time.
  - May differ for different invocations of a service, based on factors such as the user or time of delay.
  - This is referred to as a differentiated QoS.
Introduction (2)

• Most Web Servers handle incoming requests on a first-come, first-serve basis.
  → do not provide differentiated QoS.

• How can Web Servers support differentiated QoS?
Differentiated Service Outline

1. Service request
2. Classification
3. Differentiated Service

1. Service request
Internet

3. Differentiated Service

User

User

Internet

Web Server
General Web Server & Differentiated Web Server

• General Web Server
  – Handles incoming requests on a first-come, first-serve basis
  – Premium users cannot be protected from overload in server
  – Do not provide Differentiated QoS

• Differentiated Web Server
  – The incoming requests are classified into different categories
  – Different levels of service are applied to each category
  – The requests of premium users are serviced first
  – Provide Differentiated QoS
## Related Work

<table>
<thead>
<tr>
<th></th>
<th>My work</th>
<th>WebQOS (HP)</th>
<th>Nikolaos’s work</th>
<th>Jussara’s work</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Level Approach</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kernel-Level Approach</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>Portability</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td># of Differentiation Levels</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>URL (Classification)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Client IP (Classification)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>User Private Key (Classification)</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>User Authentication (Classification)</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Functional Requirements

• Classification of Requests
  – The classification methods are as various as can be thought of

• Scheduling
  – Throughput remains constant during peak demand
  – Error rate remains constant during peak demand
Non-Functional Requirements

• Portability
  – Portable on various Web servers and operating systems

• Resource Requirements
  – Must use as little CPU, memory usage as possible
Differentiated Service System Architecture

- Listen Queue
- Classification
- Schedule
- Priority Queues
User-Level Approach (1)

Web Server

PORT 80

Connection & Classification Processes

Priority Queues

Schedule Process

Execution Processes (fork)

Network Interface

OS

Reply Request

Request

Reply
User-Level Approach (2)

Differentiate Module

PORT 80
Connection & Classification Processes

Priority Queues
Schedule Process

PORT 3000
Web Server

Network Interface
OS

Reply Request

Approaches to Support Differentiated Quality of Web Services
Kernel-Level Approach

Web Server

PORT 80

Connection, Classification & Execution Process

Use Real-time Module

Real-time Module

Real-time OS

Network Interface

Reply

Request
Classification Approaches (1)

- Server-based approach
  - Use **content information** in Server
    - URL

![Diagram showing server and files with priority levels]
Classification Approaches (2)

- **Client-based approach**
  - Use *Client Information*
    - Client IP
    - User authentication
    - User Private Key
    - Key Value = Priority Level

![Diagram showing client-based approach with IP, username, and key value.]
Priority-driven Scheduling methods in User-Level Approach

- Strict priority
- Weighted priority
- Shared capacity
- Fixed capacity
- Earliest deadline first

→ We use strict priority scheduling method

→ Strict priority scheduling method is very simple and requires less CPU capacity, memory volume than other priority scheduling methods
## Development Environment

<table>
<thead>
<tr>
<th></th>
<th>User-Level Approach</th>
<th>Kernel-Level Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Linux Kernel 2.2.14</td>
<td>Linux Kernel 2.2.14</td>
</tr>
<tr>
<td>Realtime Kernel</td>
<td>None</td>
<td>Soft Realtime Kernel (Monstavista Realtime Scheduler)</td>
</tr>
<tr>
<td>Web Server</td>
<td>Apache 1.3.12</td>
<td>Apache 1.3.12</td>
</tr>
<tr>
<td>Language</td>
<td>C, PHP</td>
<td>C, PHP</td>
</tr>
</tbody>
</table>
Montavista Realtime Scheduler

- Soft Realtime Kernel Scheduler
- Priority driven Real-time process scheduler
- Priority Levels : 128
- Multilevel-FIFO, Multilevel-Round Robin scheduling method
- Separation between Real-time and Non-Real-time Processes
User-Level Approach Flow

Initial State → Listen Connection → Parse Request Header → Classification → Send Request to Web Server → Apache Web Server
Kernel-Level Approach Flow

Apache Web Server
Performance Metrics

Throughput
(reply rate per request rate)

Performance Results

Response Time
(millisecond)

Error Rate
(errors per second)
Httpperf Example

Httpperf Results

- Request part
  - Request rate (req / s)
  - Request size (B)

- Reply part
  - Reply rate (replies / s)
  - Reply time (ms)
  - Reply size (B)

- Errors part
  - Total number of errors
  - Type of errors (client-timeout, socket-timeout, connection refused, fd-unavailable, ftab-full)
General Measurement Architecture

- Use Httperf (request generator & performance measurement)
- Change of request number per second
Measurement Testbed for General Web Server

- Menam: Pentium III-800, 256MB Ram, Red-hat Linux 6.2
- Kwai: Pentium II-233, 128MB Ram, Wow-Linux 6.2
- Jordan: Pentium 133, 128MB Ram, Mizi-Linux 1.1S
Measurement Testbed for Differentiated Web Server

- Menam: Pentium III-800, 256MB Ram, Red-hat Linux 6.2
- Kwai: Pentium II-233, 128MB Ram, Wow-Linux 6.2
- Jordan: Pentium 133, 128MB Ram, Mizi-Linux 1.1S

Client 1 (Httperf) – High priority

Client 2 (Httperf) – Low priority

Request

Differentiated Web Server
Method of Measurement

- **1 Client (General / Differentiated) Web Server**
  - Using shell script

- **2 Clients (General / Differentiated) Web Server**
  - Using timing synchronization (rdate -s [system])
  - Using crontab to execute shell script periodically
Evaluation Results for General Web Server (Two Clients) (1)

- Throughput

![Throughput Graph](image)
Evaluation Results for General Web Server (Two Clients) (2)

• Throughput Rate
Evaluation Results for General Web Server (Two Clients) (3)

- Response Time

![Response Time Graph](image-url)
Evaluation Results for General Web Server (Two Clients) (4)

- Error Rate

![Graph showing Error Rate (Jordan 133) with error rate on the y-axis and request rate on the x-axis, comparing Kwai and Menam lines.](image)
Evaluation Results for Differentiated Web Server (Two Clients) (1) – Kernel-Level Approach

- Throughput

![Throughput Graph]

The graph shows the throughput (reply rate) in replies per second against the request rate (requests per second) for different levels of differentiation. The graph includes two lines: one for high differentiation and another for low differentiation. The throughput varies with the request rate, reflecting the performance of the differentiated web server.
Evaluation Results for Differentiated Web Server (Two Clients) (2) – Kernel-Level Approach

- Throughput Rate

Throughput Rate (Jordan)

- Throughput Rate (reply rate/request rate)
- Request Rate (requests/s)

![Graph showing throughput rate over request rate](image)
Evaluation Results for Differentiated Web Server (Two Clients) (3) – Kernel-Level Approach

• Response Time

![Graph showing Response Time (Jordan 133)](image)

Response Time (Jordan 133)

- **High**
- **Low**

Response Time (ms):
- 0
- 200
- 400
- 600
- 800
- 1000
- 1200
- 1400

Request Rate (requests/s):
- 10
- 80
- 150
- 220
- 290
- 360
- 430
- 500
- 570
- 640
- 710
Evaluation Results for Differentiated Web Server (Two Clients) (4) – Kernel-Level Approach

- Error Rate

![Error Rate Graph](Error_Rate_Jordan_133.png)
Evaluation Results for Differentiated Web Server (Two Clients) (1) – User-Level Approach

- Throughput

![Throughput Graph](image)
Evaluation Results for Differentiated Web Server (Two Clients) (2) – User-Level Approach

- Throughput Rate

![Throughput Rate Graph](image-url)
Evaluation Results for Differentiated Web Server (Two Clients) (3) – User-Level Approach

• Response Time

Response Time (Jordan 133)

Response Time (ms)

Request Rate (requests/s)

High
Low
Evaluation Results for Differentiated Web Server (Two Clients) (4) – User-Level Approach

- Error Rate

![Error Rate Graph (Jordan 133)](image)

Error Rate (errors/s)

Request Rate (requests/s)

- High
- Low
Conclusion

• Users demand for differentiated quality of Web service, but current Web services cannot satisfy this
• We developed support to satisfy the users needs
• Our scheme categorizes HTTP requests into classes based on classification methods, with the requests of each class handled differently → differentiated QoS is possible
Future Work

- Applying the current work in real Web Services
- A study for Web Server Framework supporting Server QoS
  - Consideration of admission control
  - Consideration of resource management
  - Consideration of disk scheduling
- A study for integrating between Server QoS and Network QoS
  - Unification of QoS parameter
  - Consideration of ToS (Type of Service) field
- Extending current work on Web Server to other Internet Servers (e.g., FTP server, VOD server, RealAudio server, etc.)