Towards Supporting Differentiated Quality of Web Service

Sook-Hyun Ryu, Jae-Young Kim and James Won-Ki Hong
Dept. of Computer Science and Engineering, POSTECH
{shryu, jay, jwkhong}@postech.ac.kr

Abstract — **The exponential rise in the number of Web users has inspired the creation of a diversity of Web applications. Hence, Web Quality of Service (QoS) is an increasingly critical issue in Web services, such as e-commerce, Web hosting, etc. In future, improved QoS will be linked to a fee for service. Customers expect their requests to be served with a quality proportional to the amount charged to their accounts. Because most Web servers currently process requests on a first-come, first-serve basis, they do not provide differentiated QoS. This paper presents two approaches to implement differentiated quality of Web service. In the user-level approach, the Web server is modified to include a classification process, priority queues and a scheduler. However, with this approach, it is difficult to achieve portability. In this paper, a new, portable user-level approach is presented. In the kernel-level approach, a real-time scheduler to support prioritized user requests has been added to the operating system kernel. Prototype implementations for two approaches have been developed.**

Keywords — **Web Quality of Service (QoS), Differentiated QoS, Web Service**

1. Introduction

The World Wide Web (WWW or Web) [10] is rapidly growing and the diversity of Web applications continues to increase, making Web Quality of Service (QoS) a critical issue in Web services [13]. With the term QoS, we refer to non-functional requirements, such as performance or availability requirements. QoS requirements are concerned with how an application or service will behave at run-time. QoS requirements may differ for different invocations of a service, based on diverse factors, such as the user or time of day. This is referred to as differentiated QoS [1, 2].

Today, most Web servers do not provide differentiated QoS. The Apache Web server [9], one of the most widely used Web servers, handles incoming requests on a first-come, first-serve basis. All requests correctly received are eventually handled, regardless of the type of request on the Web server. In this situation, premium users (i.e., those who pay more for higher quality of service) cannot be protected from overload in the Web server. Further, the throughput in a general Web server decreases during a peak period. Consequently, this general Web server does not provide differentiated QoS.

In this paper, we propose two approaches, a user-level and a kernel-level, to provide differentiated QoS using priority-based scheduling. In the user-level approach, the Apache Web server is modified to include a classification process, priority queues and a scheduler. It is difficult to achieve portability when modifying a specific Web server, such as the Apache Web server. Thus, we propose a new user-level approach, which includes the classification and scheduling of user requests. In the kernel-level approach, a Linux kernel [16] is modified by adding a real-time scheduler to support prioritized HTTP requests.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 presents functional and non-functional requirements for design and implementation issues. Section 4 presents the system design architecture and classification approaches. Section 5 describes two prototype implementations. Finally, Section 6 summarizes our work and discusses possible future work.

2. Related Work

A general architecture for Web server QoS was previously outlined by Bhatti and Friedrich in [5]. HP WebQoS is an enhancement to the HP-UX operating environment, providing a unique and advanced platform to assure high service quality for e-services on HP 9000 Enterprise Servers. HP's WebQoS stabilizes service delivery, assuring fast and consistent service quality to customers even under the competitive environment found on the Internet. WebQoS optimizies resources and permits developers to build more cost-effective solutions. WebQoS also enables businesses to prioritize service levels to allow higher service quality for the most important users and applications. The goal was to manage peaks in client request rates and to support differentiated QoS for users. Their solution essentially entails scheduling and admission control to improve the performance of high priority requests. Their architecture includes a management component so that configuration parameters can be remotely set and the server's operation monitored. This closely resembles our work. However, there are several differences. First, changes to the main Apache server code were required for both request classification and scheduling. Our new user-level prototype performs similar tasks without modifying the main Apache code. Second, they do not have the kernel-level approach concept.

Nikolaos [1] presented the design of a QoS architecture which can be added to the Apache Web server to allow the server to provide a differentiated QoS. The QoS module can support changing scheduling algorithms and the values of parameters that are used by the scheduling algorithms. They do not have the kernel-level approach concept, however. This work discusses the services needed to provide a differentiated QoS to clients of a Web site, based on the client's identity and attributes. Nikolaos's work integrated an implementation of the services with the Apache Web server and describes a service that can be used to create algorithms to suit a specific company's goal.

Jussara's work [2] investigated a method to provide a differentiated QoS: priority-based scheduling. The main metric for the QoS is latency in handling the HTTP request to the Web page. This work shows both a user-level and a kernel-level approach. However, the user-level approach can be used only in a specific Web server, and this approach does not support portability. Scheduling policies can be preemptive or non-preemptive. They have implemented preemptive scheduling at the kernel-level, and non-preemptive scheduling at the user-level. Two important aspects of the scheduling policy must be mentioned. First, upon receiving a request, the scheduling policy must decide whether to process the request immediately, or to postpone the execution (sleep policy). Secondly, the scheduling policy must also decide when a postponed request must be allowed to continue (wakeup policy). This work allows a request to continue only in place of a completed request. When a request is completed, the scheduling policy must decide which of the postponed requests, if any, should be selected to execute in its place. If the policy allows lower priority requests to execute in the absence of higher priority requests, the scheduling policy is said to be work-conserving. Otherwise, it is said to be non-work-conserving. A work-conserving policy tries not to allow processes to block while there are waiting requests. In this work, the Sleep and Wakeup policies have been implemented, by using thresholds for the maximum number of requests that can be concurrently handled at each priority level. Thus, a fixed number of slots exist for each priority level, and each incoming request must either occupy a slot (executes), or wait in a queue until it is allowed to execute (blocks).

The related work has been compared and summarized in Table 1.

	Our work	WebQOS (HP)	Nikolaos's work	Jussara's work
User-Level Approach	O	O	O	O
Kernel-Level Approach	O	X	X	O
Portability	O	O	X	X
# of Differentiation Levels	2	3	2	2
URL (Classification)	O	O	O	O
Client IP (Classification)	O	O	O	X
User Private Key (Classification)	O	X	X	X
User Authentication (Classification)	O	X	X	X

Table 1. Comparison of Related Work

3. Requirements

In this section, we discuss requirements that must be considered during design and implementation. Requirements are divided into two parts – functional requirements and non-functional requirements.

3.1 Functional Requirements

In order to design and implement approaches to provide differentiated quality of Web service, certain functional requirements must be considered. The differentiated Web server requires the following functions: classification of requests, scheduling, and execution of requests.

The basic role of a general Web server is processing HTTP [10] packets on a first-come, first-serve basis. The general Web server listens to a signal which indicates that a connection has been made. After accepting the connection, the server must serve the user request. After serving the request, the server awaits the next signal.

Unlike the general Web server, the differentiated Web server must provide differentiated QoS using priority-based scheduling. Therefore, we modify the general Web server into the differentiated Web server to add components that support differentiated QoS.

After accepting the connection, the differentiated Web server classifies the user request. The classification of requests is the first concern in the implementation of a differentiated Web server. The classification methods are as various as can be imagined. For example, in a server based method, we can categorize a URL required by the user. In a client based method, we can classify the user request with the client IP address.

After classification of the user request, the server must save it into a priority queue. A scheduler which employs a specific scheduling method assigns the user request in priority queues. The scheduling method is the second concern in implementing a differentiated Web server. As a specific scheduling method is applied, the throughput remains constant during peak demand. In addition, the error rate also remains constant during peak demand. After scheduling the user request, the server must serve the user request. Finally, this server is ready to receive another connection.

3.2 Non-Functional Requirements

To deliver differentiated QoS, a Web server can be modified to include a classification process, priority queues and a scheduler. Yet portability is difficult to achieve when modifying a specific Web server. A module which supports a differentiated QoS must use a variety of general Web servers, such as the Apache Web server and IIS Web server [17]. In addition, a differentiated Web server must be portable on various operating systems. Portability is the major concern among non-functional requirements.

General Web servers have evolved toward a multi-threaded architecture that either dedicates a separate thread to each incoming connection, or uses a thread pool to handle a set of connections with a smaller number of threads. Since a greater number of threads are used to handle user requests, more CPU capacity and memory usage is needed in general Web servers. Further, this condition is applied to a differentiated Web server equally. Because modules are added to support differentiated QoS, more CPU capacity and memory usage is needed in a differentiated Web server. Most Web servers do not serve every user request during peak demand or run short of CPU capacity and memory volume. If both Web servers have an equal CPU capacity and memory volume, the differentiated Web server must use as little CPU and memory usage as possible. Therefore, resource requirements are a secondary concern in non-functional requirements.

4. Design of Two Approaches

In this section, we discuss two approaches towards differentiated QoS in Web services, and present a differentiated Web server architecture and process structures.

4.1 User-Level Approach

Here, we present a user-level approach to support differentiated quality of Web service. In the user-level approach, the general Web server, such as Apache Web server, is only modified to include components for supporting differentiated QoS. These components consist of connection and classification processes, priority queues, a schedule process, and execution processes, as illustrated in Figure 1. Such components as classification processes, priority queues and the schedule process do not exist in a general Web server.

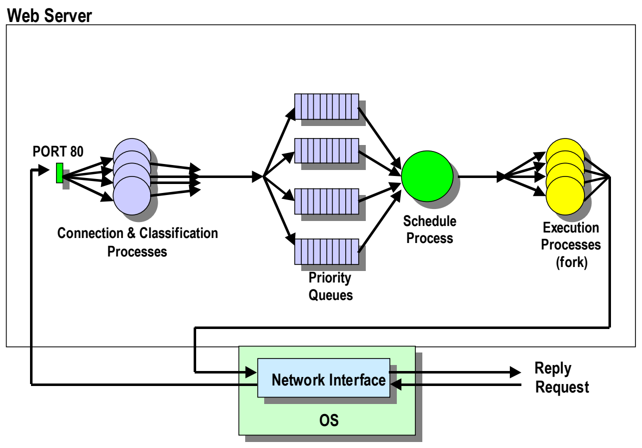

Figure 1. User-Level Approach

First, incoming user requests from the network interface in the operating system are received through port 80 by connection and classification processes. Port 80 is a well-known port for supporting Web [10] service. These processes classify the requests and place the requests on the appropriate priority queues. Several methods are used to classify requests. These classification mechanisms can be divided into two categories, a server-based and a client-based approach. In a later section, we explain the classification method in detail.

The number of priority queues implies the number of differentiation levels. A priority level is assigned for each priority queue. After requests are classified, the server must realize different service levels for each class of requests. This is accomplished by selecting the order of request execution.

A schedule process selects the next request, based on the scheduling policy. For example, requests from the highest priority queue will be processed first. Execution processes forked by the schedule process may be able to execute requests from any class, and will run until completion. Finally, execution results are sent to the user through a network interface.

Since we must modify a general Web server source code to support differentiated QoS in this approach, the system using this approach cannot handle the large variety of general Web servers. Because of this, we propose a new user-level approach to support portability, as follows.

4.2 A New User-Level Approach for Supporting Portability

Since the user-level approach described in Section 4.1 can be used only in a specific Web server, this approach does not support portability. For this reason, we propose a new user-level approach to support portability. The new user-level approach consists of a differentiate module and a general Web server, as illustrated in Figure 2.

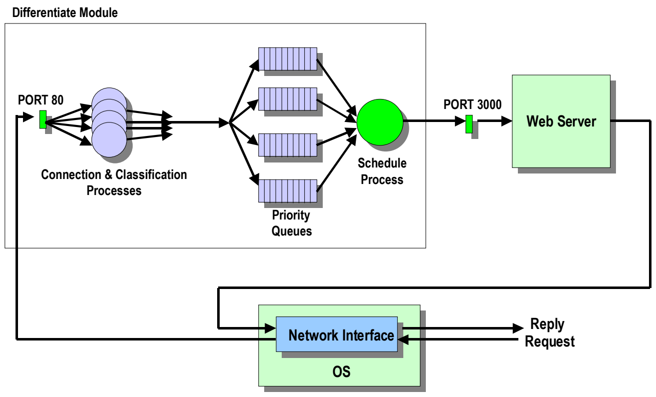

Figure 2. New User-Level Approach

The differentiate module is an independent program for the realization of a differentiated QoS. The components of the differentiate module are similar to that of the user-level approach described in Section 4.1. However, we do not the general Web server, such as the Apache Web server.

First, user requests arriving from the network interface in the operating system are received through port 80 by connection and classification processes in the differentiated module. These processes classify the requests, placing them on the appropriate priority queues in this module. Several ways exist to classify requests. In a later section, we explain the classification method in detail. After requests are classified, the server must realize different service levels for each class of requests. This is done by selecting the order of request execution.

A schedule process selects the next request based on the scheduling policy. A selected request by a schedule process is sent to a general Web server through a specific port, such as 3000. To accomplish this, the Web service port in a general Web server is configured into a specific port number, except for port 80. The selected request is processed in a general Web server. Finally, the execution result is sent to the user through a network interface.

4.3 Kernel-Level Approach

We considered it heuristically advisable to attempt a kernel-level approach, since processes which act directly on the priorities assigned to the HTTP request might be more effective in controlling executions. Therefore, we use direct mapping from the user-level request priority to a kernel-level process priority. The kernel-level approach is based on the instrumentation of both a general Web server modification and an operating system with a real-time module, as illustrated in Figure 3.

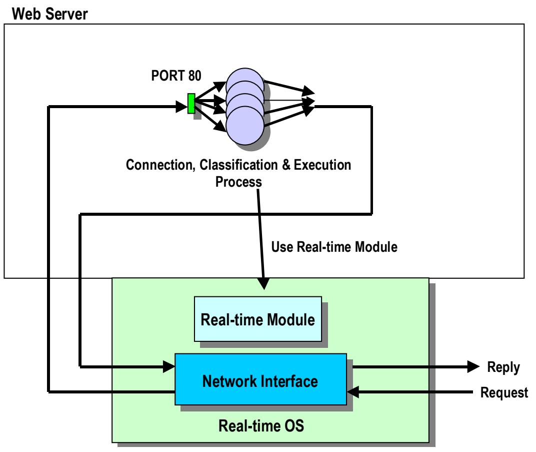

Figure 3. Kernel-Level Approach

This approach to support differentiated QoS consists of a modification of a Web server to support request classification and an operating system with a real-time module. A general Web server is modified to have each HTTP process call the kernel to record the priority of the request currently being handled. The kernel is responsible for mapping this priority into the process priority. The kernel scheduler decides which process should use the CPU next. The kernel must keep track of all processes currently using the priority, along with their current state.

User requests arriving from the network interface in the operating system are first received through port 80 by connection, classification and execution processes, in a modified version of the Web server for supporting request classification. These processes classify the requests, and adjust the priority level by themselves.

The real-time module in an operating system is responsible for the assignment of a request priority level. In general, the real-time module, such as a specific real-time scheduler, can be added into an operating system that lacks a real-time property. After processing user requests, execution results are sent to the user through a network interface.

4.4 Classification Approaches

A key requirement to support differentiated QoS is the ability to identify and classify the incoming requests of each class. Classification mechanisms can be divided into two general categories: a server-based and a client-based. The server-based approach classifies requests according to the contents or destination of the request. However, the client-based approach characterizes requests according to the source of the request.

We regard the URL [10] of the requests as crucial. A URL consists of a protocol header, a server address, a directory name, a file name and an extension. This information can be used to classify the relative importance of the request. Contents can be classified into different priority levels. Destination IP addresses can be used by a server when the server supports co-hosting of multiple destinations (Web sites) on the same node. Another approach to classify user requests is the client-based approach.

The IP address is used to distinguish clients from one another. This method is the simplest to implement. However, the client's IP address can be masked due to proxies or firewalls, so this method has limitations. Further, user authentication with username and password is used to classify a request. Finally, we use a key value to classify user requests. A key value is equal to a specific priority level.

4.5 Priority-driven Scheduling Methods in the User-Level Approach

After the requests are classified according to one of the above-mentioned classification schemes and admitted by the classifier, the server must actually realize different service levels for each class of requests. This is done by selecting the order of request execution. Execution processes are autonomous and select requests to process based on the scheduling policy. The scheduling policy may depend on queue lengths. Execution processes may be able to execute requests from any class. Alternatively, to reserve a capacity for higher-class processes, they may be restricted to executing higher-class traffic. This paper presents several possible policy guidelines.

- Strict priority – This policy schedules all higher-class requests before lower-class requests, even when low-priority requests are waiting.
- Weighted priority – This policy schedules a class based on its weight importance. For example, one class will receive twice as many scheduled requests if its class weight is twice that of another.
- Shared capacity – This policy schedules each class to a set capacity and any unused capacity can be given to another class. The class may also have a minimum reserve capacity that cannot be assigned to another class.
- Fixed capacity – This policy schedules each class to a fixed capacity that cannot be shared with another class.
- Earliest deadline first – This policy schedules based on the deadline for completion of each request. This can be used to provide a guaranteed predicted response time.

In our work, we use a strict priority scheduling method to provide differentiated Web service because it is very simple and requires less CPU capacity and memory volume than other priority scheduling methods. This condition is sufficient for non-functional requirements. If we use other priority scheduling methods, an additional resource management mechanism will be required.

5. Prototype Implementation

5.1 Development Environment

We have implemented both approaches on the development environment, as summarized in Table 2. Obviously, the development environment of the kernel-level approach is similar to that of new user-level approach, except that it uses the Montavista [14] real-time scheduler to support process priority on the operating system kernel level. The Montavista real-time scheduler is a specific scheduler to support a soft real-time kernel.

We have implemented prototypes of these approaches on the Linux kernel 2.2.14 and the Apache Web server 1.3.12. We used C language for programming the prototypes of these approaches. Further, PHP [15] script was used to configure both the classification policy and user information in these prototypes.

	User-Level Approach	Kernel-Level Approach
OS	Linux Kernel 2.2.14	Linux Kernel 2.2.14
Realtime Kernel	None	Soft Realtime Kernel (Monstavista Realtime Scheduler)
Web Server	Apache 1.3.12	Apache 1.3.12
Language	C, PHP	C, PHP

Table 2. Development Environment

5.2 New User-Level Approach Flow

We have implemented a prototype of the new user-level approach with no modification to the Apache Web server 1.3.12, as described earlier. Because a module for differentiating user requests resides in front of a general Web server, such as the Apache Web server, this prototype is portable on various general Web servers and operating systems. This module consists of a request parser, a classifier, and a request scheduler. The operation of this module is a sequence of functional flow, as illustrated in Figure 4.

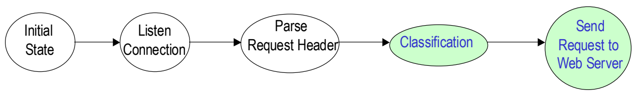

Figure 4. New User-Level Approach Flow

At first, this system resides in the initial state, for socket initialization, and so that the configuration file can be read. After the initialization of the socket, a number of sockets listen to user requests. If user requests enter the differentiation module, the request parser in this module reads the HTTP message. This parser parses the HTTP header and stores the parsing result. The classifier classifies user requests according to classification methods. At this time, the classifier uses the parsing result to classify user requests by placing the requests on appropriate priority queues. The module must actually realize different service levels for each class of requests. This is accomplished by selecting the order of request execution. A scheduler selects the next request based on the scheduling policy. A selected request by a schedule process is sent to the Apache Web server through port 3000. In Figure 4, the shaded functions represent those which are performed outside the Apache Web server.

5.3 Kernel-Level Approach Flow

We have implemented a prototype of the kernel-level approach with modification to the Apache Web server 1.3.12 and the Linux kernel 2.2.14 using the Montavista real-time scheduler, as described earlier. The priority can be set by API. The operation of this prototype is a sequence of functional flow, as illustrated in Figure 5.

As mentioned, the system begins in the initial state, both for socket initialization, and so that the configuration file can be read. After socket initialization, a number of sockets listen to user requests. If the requests come into a modified Apache Web server, the request parser in this server reads the HTTP message, then parses the HTTP header and stores the parsing result. The classifier classifies user requests according to classification methods. At this time, the classifier uses the parsing result to classify user requests, and sets a priority for execution of this process by using the API of the real-time scheduler.

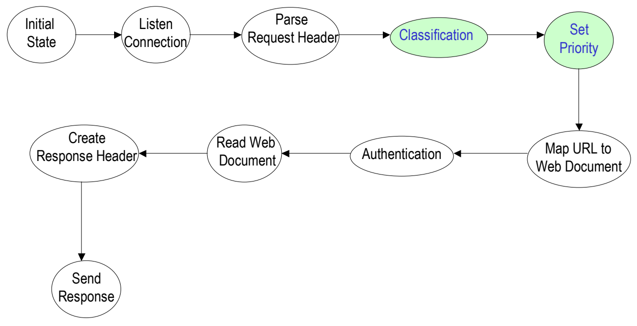

Figure 5. Kernel-Level Approach Flow

The real-time scheduler is responsible for realizing different service levels for each class of requests. The real-time scheduler selects the next execution process based on the scheduling policy. After prioritizing this process, the prototype determines the type of application interface and stores a pointer within the handler. Further, the prototype forces authentication of the user upon the URL, username and password. In the following sequence, this prototype reads Web documents from the file system and creates an HTTP response header. Finally, the execution result is sent to the user through the network interface. In Figure 5, the shaded functions are performed outside the Apache Web server and Linux kernel 2.2.14.

6. Summary and Future Work

In this paper, we presented two approaches to support Web Quality of Service (QoS) which allows a general Web server to provide a differentiated QoS. Our methods categorize HTTP requests into classes based on classification methods, with the requests of each class handled differently by the desired scheduling method.

High priority clients are given better throughput, response time, and slightly better error rates. Supporting portability in that using user-level approach is considered factor as important. Most Web server administrators intend to implement a facility for supporting differentiated service in a variety of Web servers and operating systems.

We are currently working on performance evaluations of our implementations. They include analyzing errors, throughput, and response time of a server which handles one high priority client and one low priority clients with request rates that monotonically increase. This experiment models a situation where there are a various number of clients with subscriptions for high and low quality services. Further, we are making comparative performance evaluations of two approaches supporting differentiated quality of Web services.

More work is needed on scheduling algorithms. Many approaches can be taken to perform the scheduling of requests. The open-queue system of the Web will require complex algorithms to balance processing of requests in each class while maximizing system utilization. Designing a Web server framework to support server QoS is also a complex task. The Web server framework consists of an admission controller, a resource manager, a disk scheduler and a request scheduler. More work is also needed to integrate a server QoS with a network QoS. It is necessary to map their QoS parameters. Finally, we will extend current work on the Web server to other Internet servers, such as the FTP server, the VOD server, the RealAudio server, and so on.

REFERENCES

[1] N. Vasiliou and H. Lutfiyya., "Providing a Differentiated Quality of Service in a World Wide Web Server", Proc. of the Performance and Architecture of Web Servers Workshop, Santa Clara, California USA, June 2000, pp. 14-20.
[2] J. Almeida, M. Dabu, A. Manikutty, and P. Cai., "Providing Differentiated Levels of Service in Web Content Hosting.", Proc. of the Workshop on Internet Server Performance, Madison, Wisconsin USA, March 1998, pp. 91-102.
[3] M. Banatre, V. Issarny, F. Leleu, and B. Charpiot., "Providing Quality of Service over the Web: A Newspaper-based Approach.", Proc. of the Sixth International World Wide Web Conference, Santa Clara, California USA, April 1997, Paper 149-Tec 110.
[4] K. Beyer, M. Livny, and R. Ramakrishnan., "Protecting the Quality of Service of Existing Information Systems.", Proc. of the Cooperative Information Systems, 1998, pp. 74-83.
[5] N. Bhatti and R. Friedrich., "Web Server Support for Tiered Services.", IEEE Network, September/October 1999, pp. 64-71.
[6] R. Pandey, J. Barnes, and R. Olsson., "Supporting Quality of Service in HTTP Servers.", Proc. of the SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Puerto Vallarta, Mexico , June 1998, pp. 247-256.
[7] Y. Bernet at al, "An Architecture for Differentiated Services", IETF, October 1998.
[8] T. Abdelzaher and N. Bhatti, "Web Server QoS Management by Adaptive Content Delivery", Proc. of the 7th International Workshop on Quality of Service, London, England, June 1999, pp. 216-225.
[9] Apache Group, http://www.apache.org.
[10] R. Fielding, J. Getys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", IETF, January 1997.
[11] Richard W. Stevens, *TCP/IP Illustrated: The Protocols, volume 1*, Addison-Wesley, Reading, MA, 1994.
[12] R. Bless, K. Wehrle, "Evaluation of differentiated services using an implementation under Linux", Proc. of the 7th International Workshop on Quality of Service, London, England, June 1999, pp. 97-106.
[13] N. Bhatti, A. Bouch, and A. Kuchinsky, "Integrating User-Perceived Quality into Web Server Design", Proc. of the 9th International World Wide Web Conference, Amsterdam, Netherlands, May 2000, pp. 92-115.
[14] Montavista Software, http://www.montavista.com.
[15] PHP script, http://www.php.net.
[16] Linux Online, http://www.linux.org.
[17] Microsoft TechNet, http://www.microsoft.com/technet/iis.