Arima

X-Switch

A Secure, Multi-User, Language-Independent Web Application Server

Mayumbo Nyirenda* — Hussein Suleman**

* Department of Computer Science
University of Zambia
P.O Box 32379
Zambia
nmayumbo@cs.uct.ac.za

** Department of Computer Science
University of Cape Town
Private Bag X3
Rondebosch 7701
South Africa
hussein@cs.uct.ac.za

ABSTRACT. The need for user interactivity has led to the creation of many technologies for Web applications. These Web applications are usually installed on and accessed through a Web application server. Security is however a problem in such environments and most of the servers and Web applications are thus run as the restricted user 'nobody'.
Performance also is an issue due to process re-initialisation. Many technologies resolve some security and performance shortcomings in limited contexts. This paper presents the X-Switch system: a secure language independent Web application server that is a generalisation of existing technologies. Experimental results show that the X-Switch system can perform comparably to other Web application servers and that a general solution is possible.

RÉSUMÉ. Les applications Web sont généralement installés et par l'intermédiaire d'un serveur d'application Web. La sécurité et la performance est toutefois un problème dans ces environnements et la plupart des serveurs Web et les applications sont gérées comme limité 'personne'.
Ce document présente les X-Switch système: assurer une langue indépendant serveur d'application Web qui est une généralisation de les technologies existantes. Les résultats expérimentaux montrent que une solution générale qui peut réaliser comparable à d'autres serveurs est possible.

KEYWORDS : Web application, scalability, process persistence, modularity

MOTS-CLÉS : Application Web, d'évolutivité, de traiter la persistance, modularité

1. Introduction

The stable growth and popularity of the Internet has inspired the development of component based systems that interface using the World Wide Web e.g., over Web Services. These components can each be implemented using a variety of programming languages, resulting in the desirability of a Web application server that can host multiple implementation languages. In addition, secure multi-user environments, like academic environments, used to host a spectrum of Web application implementation technologies can easily become too complex to manage effectively. It can be argued that this dual need for control and freedom can best be met by a single secure and language-independent Web application server.
To this end, this paper presents the X-Switch system, a Web application server that is language-independent, secure, multi-user and efficient. This system is based on a proof-of-concept prototype [4] that first demonstrated that the different modules can fit together but without much language-neutrality and with performance tests restricted to cases where only a single connection was made at a time. The architecture as presented in this paper overcomes both of these problems and is described and evaluated in the sections that follow.

2. Background

The first Web application technology for dynamic responses was the Common Gateway Interface (CGI) [7]. The Common Gateway Interface (CGI) is a protocol for communication between Web applications and Web servers. The major strength and success of CGI was due to the fact that CGI Web applications could be written in any language and also because CGI was compatible with various Web server architectures. However, CGI starts a new process for each request received and thus an increase in the number of user requests is associated with performance degradation.

2.1. Web application technologies

2.1.1. Persistent processes

To solve the performance degradation problem associated with CGI Web applications, Open Market Inc developed FastCGI, a variation of CGI, having processes that run persistently just like SpeedyCGI [5]. Using persistent interpreters removes the overhead associated with starting a new interpreter every time a new request arrives. Java Servlet technology [1] uses Servlets that work as a type of extension to the Web server functionality and are simply Java objects that implement the Java Servlet API interface. A Java Servlet class is loaded once and then instantiated every time a new request arrives. How-

ever, SpeedyCGI and Servlets focus on a single technology whereas FastCGI puts the burden of persistence on the Web application developer.

2.1.2. Web server APIs

Modern Web servers allow developers to have access to the Web server core functionality and structures through an Application Programming Interface (API) [6]. Web server APIs can be used to develop Web applications as Web server modules. For example, mod_oai [12] is an Apache module that implements the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH). However Web applications developed using Web server APIs do not port to other Web servers since Web server APIs are usually proprietary. Web server API applications are not a desirable solution in a multi-user environment with users of varying user privileges because they make Web server administration complex if not impossible.

2.2. Security in Multi-user Web server environments

To reduce the eventuality of an external attack due to poorly written or intentionally harmful scripts, most Web servers are run in a restricted environment as an unprivileged user without login permissions, such as the user 'nobody'. However, scripts run under the same user are usually restricted to the same isolated subset of the file system. Therefore there is nothing to protect one author's data from other authors' scripts.
A viable solution to this problem is using a wrapper script. The wrapper script performs consistency checks on the CGI script file and the execution environment to prevent potential damage to the script and/or the files in the directory in which the script resides.One such wrapper script is cgiwrap [8] which runs the scripts with the privileges of the owner. Similar wrapper scripts are suEXEC [9], suPHP [10] and sbox [14].
Java Servlet technology uses sandboxing to enforce security. The sandbox acts as a form of script isolation. The isolated script is run in an environment where it can cause least or no damage at all to the host system.

2.3. Server performance optimization techniques

Web servers can serve hundreds or even thousands of connections concurrently. Web server performance is affected by IO and therefore one connection's IO blocking can degrade the performance of the server. There are several architectures for managing concurrency on Web servers. The main levels of classification are event-driven, thread per-connection and hybrid, which utilizes both threads and event-driven approaches. Recent studies by Pariag et al [2] reveal that an event driven server and hybrid based server achieved up to 18% higher throughput than the best implementation of the thread based server. Therefore the event driven architecture is a more viable approach for a server that implements concurrency with minimal disc access.

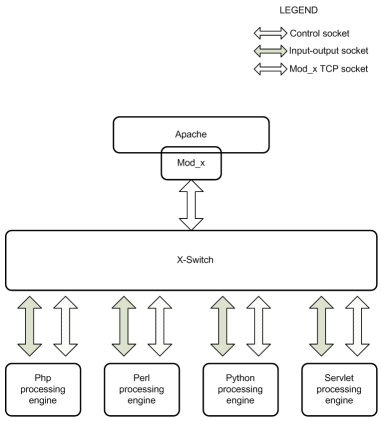

Figure 1. *Modular design of X-Switch-1.1*

3. Design and Implementation

The design of the X-Switch (pronounced 'cross switch') system uses a modular approach. It has three main levels of processing: the Web server module; the main processing module (X-Switch); and the processing engines (See Figure 1). It is a single threaded Web application server that uses a polling mechanism to listen on all its connections. The actual execution of the Web application scripts is done in the processing engines. Requests are received by Mod_x and routed to the X-Switch main module which then computes which processing engine to assign the request to. The processing engine then runs the script which writes its response to the standard output. X-Switch then reads the response from the engine's standard output and relays the response to Mod_x.

3.1. Mod_x

Mod_x is implemented as an Apache Web server module. The Apache Web server was chosen due to its popularity and widespread use. Mod_x is a dynamic shared object (dso) which is loaded at runtime. It uses a single handler to handle all request types. Upon receiving a request it establishes a socket connection with the X-Switch module. It uses this socket to send the headers and any other information needed for request processing to the X-Switch module. Mod_x also uses this same socket to read the response generated by X-Switch.

3.2. X-Switch

The main module is the X-Switch module which is used to manage user information and processing engines. It bridges Mod_x and the processing engines. It uses a single thread and a polling mechanism to read and write on all the sockets. Upon receiving a request it first determines the owner of the Web application that is required to process the request. If no engine required to process the request exists, X-Switch then spawns a new engine to process the request. X-Switch establishes two socket connections with the processing engine: the control socket and the input-output socket. The control socket is used for transmitting the request information to the processing engine and also for receiving the signal for the completion of request processing by the processing engine. The input-output socket is used for sending the request body data, in the case of a POST request, to the engine's standard input and also for reading the response from the engine's standard output. Upon establishing the connection, X-Switch sends the request processing information to the processing engine and relays the response from the processing engine to Mod_x. The X-Switch module also is responsible for managing user information and processing engine control. X-Switch controls and determines when processing engines are to be created and destroyed. It is also responsible for load control. When there are too few engines for the request load, X-Switch then spawns more processing engines. X-Switch uses timers to remove idle connections and engines.

3.3. Processing engines

Processing engines implement the back-end functionality of the X-Switch system. Any programming language that can write to the standard output stream and read from the standard input stream can be used to implement a processing engine. The processing engines are run with the privileges of the owners of the scripts they will execute using suexecme and run persistently. The current X-Switch system implements four types of processing engines: Perl, PHP, Java and Python processing engines. They have support for HTTP POST and GET methods, as well as APIs and local environment variables specific to each technology e.g., the Java processing engine supports a core subset of the Java Servlet API.

4. Evaluation

The evaluation of the X-Switch system focused on performance tests and analysis and compares the performance of the universal Web application server to known Web application servers. The performance evaluation also investigated the impact of the layers of processing on the total processing time. Two machines were used to create a simple network using a crossover cable. The client was run on a Pentium IV 3.2Ghz desktop with

Table 1. *The percentage of time that each processing layer contributes to the total processing time*

Processing layer	Percentage of time (%)	Measured time (s)
Web application	24.52	14.957005
Apache	40.13	24.479877
Mod_x	14.40	8.781802
Processing engine	11.06	6.033242
X-Switch	9.89	6.748551

512 Mb RAM while the server was installed on a Pentium M 1.73Ghz laptop with 512 Mb RAM. The software used to simulate user transactions and connections was Jakarta-jmeter.

4.1. Experiment 1

The first experiment measured the impact of the processing layers on the total response time. The experiment was conducted by issuing 100,000 requests to each layer of processing and the time required to service the requests was measured and recorded. The script used in this experiment had a simple 52 byte response.

4.1.1. Results

The results (See Table 1) show that most of the processing time is spent in the Apache and Web application processing layers. The response time for the Web application used in this experiment was small because a trivial response was used. Therefore as the response size increases the amount of time that the generation of the response takes would also increase and the Web application would contribute the bigger proportion of time. Therefore the response time is mostly affected by the time it takes to run the Web application.

4.2. Experiment 2

This experiment measured the response time of X-Switch using a synthetic work load and compared it to other Web application servers. Jakarta-Jmeter was used to issue requests simulating ten concurrent connections (threads). Each of the threads issued 1,000 requests. The experiment was conducted with Apache Tomcat [3], FastCGI, SpeedyCGI and the Perl and Java processing engines using simple "Hello World" applications.

4.2.1. Results

On average, the X-Switch system performs comparably with other Web application servers as demonstrated by the recorded average response times (See Figure 2 and 3). The high values for the initial response times are the result of the preparation and engine

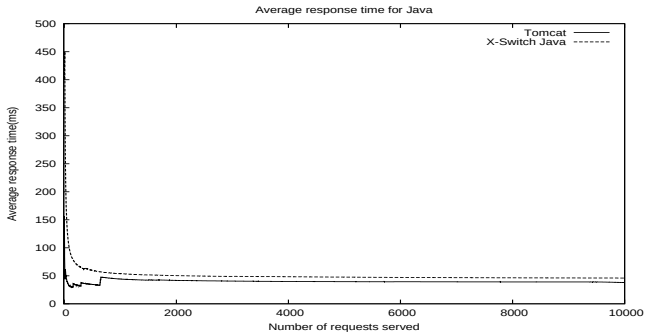

Figure 2. *Average response time for Apache Tomcat and the Java processing engine*

setup costs. The persistence of the processing engines however leverages this with time. It was anticipated that the extra layers of processing and the generality would reduce the performance but not to a great extent and thus the results confirm what was anticipated.

5. Conclusions and Future Work

The X-Switch system demonstrates the feasibility of a secure multi-user language independent Web application server. Performance tests show that the performance of the

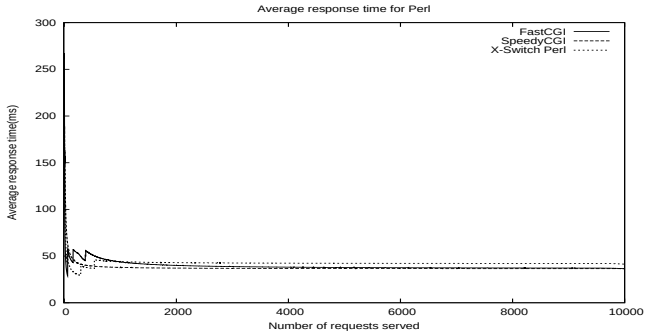

Figure 3. *Average response time for the Perl processing engine, SpeedyCGI and FastCGI*

X-Switch system is comparable to known Web application servers and using a modular design approach has little impact on the performance of a Web application server.
The X-Switch system can be used in educational institutions and training environments where multiple implementation technologies are required and used to make administration easier. In addition Web hosts can use a single server to provide services for multiple implementation technologies as opposed to dedicating machines to each single technology. Future work could include many performance optimizations. For example, Mod_x currently waits for X-Switch to close a connection as a signal for the end of the response. Developing a protocol that can signal the end of a response without closing the connection would improve the performance of the universal Web application server.

6. References

[1] D. NOURIE, "Books to Shorten Your Learning Curve" *Sun microsystems*, March, 2001.

[2] D. PARIAG, T. BRECHT, A. HARJI, P. BUHR, A. SHUKLA, R. DAVID, "Proceedings of EuroSys'07", *Comparing the performance of Web Server Architectures*, Lisboa, Portugal, March, 2007.

[3] R. LERNER "At the Forge: Server-Side Java with Jakarta-Tomcat", *Linux journal*, vol. 2001,num. 84es, 2001.

[4] A. MAUNDER, R. VAN ROOYEN, H. SULEMAN "Designing a 'universal' web application server ", *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries SAICSIT '05*, July, 2001.

[5] S. HORROCKS "Introduction to SpeedyCGI ", *Proceedings of Yet Another Perl Conference North America*, June, 2001.

[6] K. COAR "Writing modules for Apache 1.3", *Proceedings of the O'Reilly Open Source Conference*, August, 1999.

[7] S. GUNDAVARAM "CGI Programming on the World Wide Web ", *O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.*, First Edition, March, 2006.

[8] " CGIWrap - User CGI Access", *http://www.unixtools.org/cgiwrap*.

[9] "Apache suEXEC Support ", *http://httpd.apache.org/docs/1.3/suexec.html*.

[10] S. MARSCHING "suPHP", *http://www.suphp.org/*, 2006.

[11] B. BARRETT"The Webalizer - What is your Web server doing today?", *http://www.mrunix.net/webalizer/*, November, 2006.

[12] MICHAEL L. NELSON, H. VAN DE SOMPEL, X. LIU, TERRY L. HARRISON, N. MCFARLAND "Securing and Optimizing Linux RedHat Edition - A Hands on Guide", *Open NA*, 2000.

[13] P. HEINLEIN "FastCGI", *Linux journal*, vol. 1998,num. 55es, November, 1998.

[14] L. D. STEIN , "SBOX: Putting CGI Scripts in a Box" *Proceedings of the USENIX Annual Technical Conference*,Monterey, California, USA, June, 1999.