Universal Web Server

The X-Switch System

Technical Report #CS04-20-00

Andrew Maunder
University of Cape Town
Department of Computer Science
+27 21 650 2663
amaunder@cs.uct.ac.za

Reinhardt van Rooyen
University of Cape Town
Department of Computer Science
+27 21 650 2663
rvanrooy@cs.uct.ac.za

-

Supervisor:
Dr Hussein Suleman
University of Cape Town
Department of Computer Science
+27 21 650 2663

ABSTRACT

Web servers have become increasingly powerful since they were created. The services they offer have changed as computer hardware has improved, networks have sped up and people demand more interaction for their Web browsers. Web servers perform their function well. They are built purely for one purpose, namely speed. Web servers have sacrificed some functionality by prioritizing efficiency and security. Web servers take up a lot of system resources and are so efficient that they can provide their service to multiple users on the server. However, the only way users can currently use a Web server to its full potential is to own the process running the Web server. As users demand more functionality from Web servers, there is growing interest in providing additional capabilities to Web servers without affecting the efficiency that they operate at. The X-Switch system is a project dedicated to evaluating the feasibility of creating a Web server capable of providing users with all the features they need whilst maintaining the performance of current Web servers. The X-Switch system will also investigate the possibility of creating an extensible, modular system.

General Terms

Measurement, Performance, Design, Experimentation.

Keywords

Multi-user, Multi-language, Context switching, Persistence, Web Application server, Web server.

1. INTRODUCTION

The popularity of the Internet has increased the demand for relevant content to be delivered to users. Running a website has become a de facto business practice and many individuals also choose to have a presence online. Web servers have the capacity to process numerous requests on behalf of many users on the server in an efficient manner. For the sake of efficiency and security, comprehensive multi-user Web server systems have not been developed to their full potential. A Web server can only be run as one particular user and with that particular user's permissions and resources. This introduces artificial limitations on users of the Web server. For example, scripts executed by the Web server cannot create files on behalf of the user who owns the script. The Web server can only create files on behalf of the user who owns the Web server. Previous projects have tried to address this lack of functionality in Web servers. These projects, while successful, often suffered severe performance penalties and were often too limited in scope. The solutions forced the user to alter the way that scripts had to be written to make it compatible with the application. The combination of some or all of these negative side effects hampered the introduction of a Web server that could offer multiple users the desired functionality.

The X-Switch project is an investigation into the feasibility of creating a Web server that can provide the additional functionality mentioned above, whilst retaining the performance benefits of Web servers optimised for speed. It also investigates the possibility of creating a general solution independent of a specific implementation language.

2. BACKGROUND

The development of early Web server functionality was driven by requirements of its users. Initially these requirements amounted to displaying static content. As network speed and server performance increased, users required Web servers to provide additional services such as the generation of dynamic content by server side applications. The Common Gateway Interface (CGI) was one of the first standards that allowed server side applications to service Web requests.

CGI initialised a new application instance for each request received by the server. The overhead of repetitive process

initialisation severely hampered the performance of CGI. The World Wide Web Consortium (W3C) requested a more efficient alternative to the CGI standard. Open Market Inc. responded with the development of FastCGI [6], a persistent implementation of CGI that provided a mechanism for reusing existing application instances to service future requests while retaining all the benefits of CGI such as process isolation, language and architecture independence. CGI is still used as an interface between a Web server and an application or programming language. It thus serves as a reliable measure of performance. Any system that can beat CGI's performance is applicable to the Web server industry. Similarly, any system that cannot beat CGI's performance will not be accepted by anyone.

As multiple users used the same server to host their websites and their Web applications, the need to isolate the users' applications from each other grew. Web servers were not capable of letting a user's application create files belonging to them, nor allowing the applications the same permissions as their owners. Applications such as Apache's suEXEC [7] were developed to allow Web applications to run with the permissions of the owner of the script

3. FRAMEWORK

3.1 Overview

The aim of the X-Switch system is to provide a general, extensible solution to provide both persistence and context switching to users, independent of a programming language or application. The X-Switch system receives requests from the Web server, and creates an environment in which the requests can be processed. This environment incorporates both the persistence required for efficient processing and context switching to allow the processing engine to be executed with the same permissions as the user who owns the requested script.

3.2 Architecture

The X-Switch architecture consists of three functional modules.

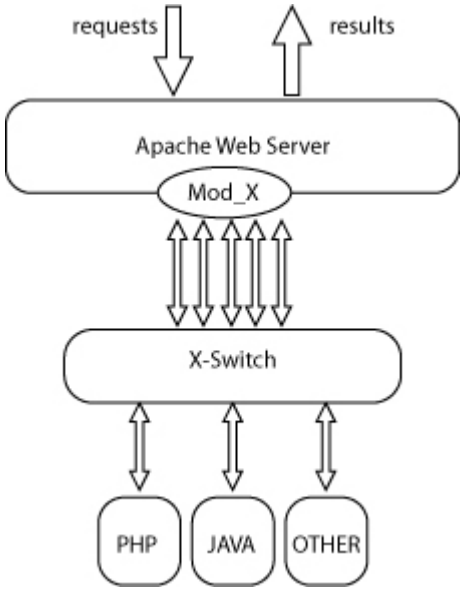

Figure 1. The X-Switch system's architecture diagram.

The diagram above illustrates the system architecture of the X-Switch system. Requests enter the Web server and are sent to the X-Switch module. The module controls the processing engines below it and once the process has been processed, sends the results back to the Web server. A detailed explanation of the various modules of the X-Switch system is given below.

The X-Switch system consists of:

- **Apache Web server module**: Mod_X is an Apache module that utilises the Apache API to define the X-Switch request handlers.
- **X-Switch**: The X-Switch system controls the creation and utilisation of processing engines for different users and different types of requests. It is responsible for allocating the correct requests to the correct processing engines and to control the system's resource usage.
- **Processing engines**: The processing engines are the modules that actually process a Web request. The engine keeps the application or language persistent and listens for requests sent by the X-Switch system.

4. IMPLEMENTATION

4.1 The Mod_X Apache Web server module

Mod-X is a functional module that is incorporated into the Apache core during the Web server startup phase. This is done via Apache's DSO [3] interface which loads modules without having to recompile Apache. The Mod-X module utilises two main features of the Apache API: the Apache custom handlers and configuration directives.

- Custom handlers

  The Mod-X module contains a handler for each language supported by the X-Switch system. Apache analyses the request message to determine if any handler has been registered to service a request of that particular type. If a suitable handler is found, Apache passes it all the information about the current transaction and server configuration. These take the form of a reference to a request object and a server object. The handler will then extract the necessary data required for request processing.

- Configuration directives

  The Apache API [3] allows a module to install its own configuration directive in the main Apache configuration. The configuration directive provides the filtering rules to ensure that each request is sent to the appropriate handler. A Mod-X handler registers *'a per URI'* configuration directive that matches string patterns that may occur in a given URI. Apache will then match a request to an appropriate handler based on the directive registered.

The Mod-X module is called if the Apache core encounters a configuration directive that points to one of its handler definitions. A separate handler is provided for each X-Switch engine installed, thereby providing a mechanism for request modification depending on engine type. The handler then applies

for a new TCP/IP connection with the X-Switch module. If successful, the modified request message is sent down the data channel to the X-Switch module for redirection to an appropriate engine. The handler then keeps the data channel open, sending response data back to the requesting HTTP client as it arrives. When the entire response message has been received, the data channel is closed and the Mod-X module returns the OK message to the Apache core, signalling that the request has been successfully handled.

4.2 X-Switch component engines

A component engine can be written for any programming language or processing application so that it can interact with the X-Switch system. Essentially, a component engine wraps around a processing application or language to make it compatible with the X-Switch system. The goal of the engine is to provide a layer of abstraction between the X-Switch system's requirements and scripts created by users. This abstraction allows scripts to interact with the X-Switch system without having to be modified to make the script compatible with the X-Switch system. This makes it easy for users and Web server administrators to implement the X-Switch system to benefit from its added functionality.

4.2.1 The X-Switch Servlet Engine

The X-Switch Servlet engine provides an environment for the execution of Java Servlets. The X-Switch Servlet engine had to meet the following set of requirements:

- Light-weight execution environment
- Efficient servicing of requests
- Minimal startup time

4.2.1.1 Light-weight execution environment

The X-Switch system requires multiple component engine instances to exist simultaneously, therefore a single instance of the X-Switch Servlet engine should consume a minimum amount of system resources. The X-Switch Servlet engine achieves this by providing a sub-set of the functionality provided by heavy-weight industrial Java Web application servers.

4.2.1.2 Efficient servicing of requests

The X-Switch Servlet engine utilises two performance-enhancing techniques, namely thread pooling [5] and class buffering.

Thread pooling optimises the engines' efficiency by initialising threads during its startup sequence and keeps them dormant until they need to process a request. This reduces the overhead associated with repetitive runtime thread initialisation.

The class buffering technique attempts to improve engine performance by minimising disk I/O associated with loading a Servlet class file. Once a Servlet is read, it is loaded into the class buffer. Any subsequent call to the same Servlet will use the Servlet stored in the class buffer, reducing the number of times the file system has to be accessed. Disk I/O is a major factor in the overall response time to process a request and class buffering avoids this overhead.

4.2.1.3 Minimal startup time

The X-Switch Java Servlet engine is designed to minimise the time it needs to be fully initialised. It needs to do so because the engine is started up in a runtime environment. Traditional containers preload all the deployed Servlets and create an environment in which the Servlets can be executed efficiently. The initialisation time of industrial containers is not critical because no requests are waiting to be executed before the container is started up. The X-Switch Java Servlet container is started when no existing engine is available to service a request. This means that there is a request waiting to be executed, which in turn means that the amount of time spent initialising will directly impact the delay before the request is processed.

The X-Switch Servlet engine decreases its initialisation time by not preloading all the user's scripts and by only pooling a small amount of threads.

4.3 X-Switch module

The X-Switch module receives requests from a Web server through a TCP/IP socket, analyses the requests and then sends it to the appropriate processing engine. The engine that the request is sent to is determined by the type of request and the owner of the script being requested. The X-Switch system is responsible for creating specific engines based on the scripts requested. If the load for a particular user's engine is high enough, the X-Switch system will create a second instance of the processing engine. Both will process requests from one request queue, thereby increasing the throughput of scripts being executed. Each engine has a thread in the X-Switch system to listen for output from the engine. This multi-threading allows multiple requests to be processed without having to serialize the output from the engines.

To perform the context switch, the X-Switch system relies on a small application that is run as the root process. The program has it "suid" bit set by the root user to allow any person to execute the application and still have the process run as the root user. This program is responsible for setting up a process belonging to the user whose script was executed and for starting up an engine of the correct type. The small "suid" application is run each time an engine needs to be created so that there is no persistent root process running that can be exploited. The application only accepts predefined commands in order to increase the security of the application.

The X-Switch system sets up an environment in which a processing engine can easily operate. The main feature that the system offers processing engines is that it changes the standard input and output descriptors to communication sockets that are created by the X-Switch system. This means that processing engines only have to read from standard input and written to standard output in order to communicate with the X-Switch system. Processing engines do not have to worry about reading from sockets or buffering their output. This allows any language or application that can read from the keyboard and write to the screen to be used to process requests. Legacy applications can be

hooked up to the Web server through the X-Switch system without having to alter its code.

4.4 The PHP processing engine

The PHP processing engine provides a simple wrapper to turn the command line interface version of PHP into a processing engine. It provides the persistence needed to increase the speed of processing requests and sets the environment in which scripts can be executed. The engine is responsible for setting up the variables required by a script and prints the output of the script to the standard output. Once a script has finished executing, it sends a control signal to the X-Switch system to indicate its status and listens for more requests that need to be executed.

5. RESULTS

The X-Switch system was evaluated against two different criteria. The first was to determine whether the system could provide context switching for heterogeneous applications or programming languages. This functionality was proven experimentally by writing files to the file system as the user whose script was used to write the file.

The second criteria was that the system had to be comparable to similar technologies in terms of efficiency. CGI was used as the lower bound in the evaluation. If the performance of the X-Switch system was worse than that of CGI, the X-Switch system would serve no purpose. The results of the tests revealed that the X-Switch system was substantially faster than CGI and the performance was in fact comparable to the most optimised solutions available.

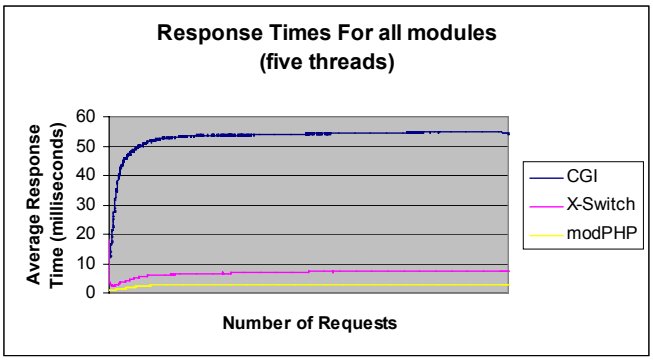

Figure 2. Graph comparing three processing engines

The graph in figure 2 shows a moving average of the response times in milliseconds per request over time. The graph clearly shows that the average response time of the X-Switch module is considerably less than that of CGI and comparable to the response time of modPHP. modPHP is the PHP engine the Apache Web server uses to process PHP scripts. It is designed purely for efficiency and therefore serves as a good indication of performance. The X-Switch system is marginally slower than modPHP. Given the additional functionality that the X-Switch system provides, the X-Switch system proves to be well within the bounds of accepted performance.

The test was performed by using a Web request load meter to generate 5000 requests and to measure the response times of each request. The load meter used five threads to simulate heavy Web traffic conditions.

The performance of the Java Servlet container was also tested. It was tested against an array of Java Servlet containers used in the Web server industry to process Java Servlets. These Java Containers are the fastest currently available and any comparable result would prove the X-Switch system and the Java Servlet container to be acceptable in terms of efficiency. The lower bound was chosen to be CGI.

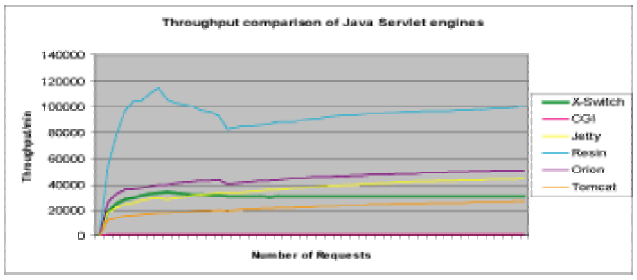

Figure 3. Comparing X-Switch to leading Java Servlet containers.

The graph in Figure 3 shows that the X-Switch system is comparable with most other Java Servlet containers and that it exceeds the performance of CGI by a significant margin. Given that the X-Switch system offers functionality that no other Java Servlet container can, the X-Switch system is in fact more desirable than many leading Java Servlet containers.

The test was performed by using a Web request load generator to send 5000 requests to each Java Servlet container. The response time for each request was captured. The graph in Figure 3 shows the throughput of each Java Servlet container per minute over time.

6. CONCLUSIONS

The X-Switch system investigated the feasibility of creating a multi-user Web application server independent of any particular implementation language.

A prototype was created after examining previous projects and analysing the requirements of the hypothesis. The prototype was then tested to see whether the prototype produced a feasible system in terms of the functionality that it added to a Web server and in terms of performance.

The results of the evaluation proved that the X-Switch system's efficiency and performance is well within today's standard norms of Web server response times. The evaluation also proved that the X-Switch system could add the functionality currently lacking Web servers.

7. FUTURE WORK

The X-Switch system produced a prototype that proved the project to be a feasible solution. However, a large number of improvements need to be introduced before it can be considered to be an industry-grade Web server solution.

The X-Switch system prototype was developed to be run in optimal or near optimal conditions. The system would need to be much more robust and secure in order to be used as part of a Web server.

The X-Switch system is currently only compatible with Apache 1.3. The mod_X Apache module is not compatible with Apache version 2. A compatible module would have to be created. Modules for different Web servers should also be created, as there is no reason why the X-Switch system would not work with any Web server available.

The X-Switch system is currently platform dependent. A future version of the system would have to be platform independent to be widely accepted and implemented in industry. Platform independent code would also make more use of standard programming practices, but may make the system slightly less efficient.

The prototype was developed without much consideration for its resource usage. The X-Switch system could be more memory efficient. Future development could improve the memory efficiency by sharing static code between identical processing engines instead of loading it into memory each time a processing engine is created. Numerous similar optimisations would improve the overall memory consumption of the X-Switch system.

8. REFERENCES

[1] Knambatti, M. “Named pipes, sockets and other IPC.” April 2001. [Online]. http://www.public.asu.edu/~mujtaba/Articles%20and%20Papers/cse532.pdf. Available.

[2] Sun Microsystems. 2003. The Java Servlet API: White Paper. Available at: http://java.sun.com/products/servlet/whitepaper.html. (Accessed 20 July 2004)

[3] Laurie B., and Laurie P., Apache: The definitive guide. O’Reilly, Sebastopol, CA, 1999.

[4] Heaton J., Creating a thread pool with Java, Sams. Available at: http://informit.com/articles (Accessed 18 September 2004)

[5] Open Market Inc. 1996. FastCGI: A high performance Web server interface. Available at: http://www.fastcgi.com (Accessed 21 July 2004)

[6] Apache Software Organisation., 2004, Apache suEXEC Support Technical Document. Available at: http://www.apache.org

(Accessed 2 July 2004)