Performance Analysis of Dynamic Web Page Generation Technologies *

Bhupesh Kothari and Mark Claypool

Department of Computer Science
Worcester Polytechnic Institute, Worcester, MA 01609, USA
{bhupesh|claypool}@cs.wpi.edu

Abstract

The World-Wide Web has experienced phenomenal growth over the past few years, placing heavy load on Web servers. Today's Web servers also process an increasing number of requests for dynamic pages, making server load even more critical. The performance of Web servers delivering static pages is well-studied and well-understood. However, there has been little analytic or empirical study of the performance of Web servers delivering dynamic pages. This paper focuses on experimentally measuring and analyzing the performance of the three dynamic Web page generation technologies: CGI, FastCGI and Servlets. In this paper, we present experimental results for Web server performance under CGI, Fast CGI and Servlets. Then, we develop a multivariate linear regression model and predict Web server performance under some typical dynamic requests. We find that CGI and FastCGI perform effectively the same under most low-level benchmarks, while Servlets perform noticeably worse. Our regression model shows the same deficiency in Servlets' performance under typical dynamic Web page requests.

Keywords
World Wide Web, Web server, dynamic Web pages, benchmark, performance evaluation

1 Introduction

Dynamic Web pages are an important tool in the exchange of information in today's business community. Electronic commerce servers rely heavily on dynamic applications to access and present content to users distributed across the Internet. The need for dynamic Web pages is fueled by the requirements for interactive business transactions as well as Web sites that are personalized on an individual basis. The need for Web servers to create dynamic Web pages has resulted in the emergence of new dynamic page generation technologies. As a consequence of the increasing wide spread use dynamic Web page generation with the ever increasing growth of Web, Web servers are becoming more stressed then ever [8]. Performance of Web servers has become a critical issue.

The important movement away from static Web content to dynamic content is pushing the limits of the Common Gateway Interface (CGI), the de facto standard for dynamic Web page creation. CGI has benefits like ease of understanding, language independence and server architecture independence but with some significant drawbacks, including creation of a new process for each request. Two promising approaches to move beyond CGI are:

- *FastCGI:* FastCGI removes the inherent CGI performance problem by making the CGI processes persistent; after finishing a request, the FastCGI processes waits for a new request instead of exiting [19].
- *Servlets:* Servlets are Java objects that are invoked by the Web server and run in a resident Java Virtual Machine as threads [22]. They are loaded only once instead of being spawned at every request.

*This paper appears in *Proceedings of the International Network Conference (INC)*, Plymouth, United Kingdom, July 3-6, 2000.

Web servers can easily be configured to deliver static pages to high numbers of concurrent users without substantial performance degradation [5] . Moreover, performance analysis of Web servers for static requests has been well studied [4] [23] [3] and modeled [23] [2]. However, to the best of our knowledge, Web server performance for dynamic documents has neither been experimentally measured nor thoroughly analyzed. Moreover, benchmark tools for measuring dynamic page generation technologies exist only for CGI.

This paper presents the performance of three dynamic Web page generation technologies: Servlets, CGI and FastCGI. We develop Web server benchmarks for measuring Servlets, FastCGI and CGI performance and present experimentally based results of the performance of each technology. Lastly, we develop a multivariate linear regression model based on our experimental results that predicts the performance of Web servers for different workloads. We show that, contrary to popular belief, performance of CGI is comparable or better when compared to other dynamic page generation technologies. Our analysis will help in comparing bottlenecks in the performance of dynamic page generation technologies to other bottlenecks in Web server performance.

The contributions of this work are:

- Experimentally based measurements of the fundamental parameters of three dynamic Web page generation technologies: CGI, FastCGI and Servlets.
- A flexible multivariate linear regression model for predicting dynamic Web page generation performance under varying request sizes.
- Performance predictions for typical requests as request size increases.
- A methodology for conducting future Web server performance measurements.

The remainder of this paper is organized as follows: Section 2 details our experiment design to measure the parameters of dynamic Web page requests and presents performance results obtained from running our experiments; Section 3 introduces our regression model for predicting dynamic Web page generation performance; and Section 4 summarizes our conclusions and lists possible future work.

2 Experiments

2.1 Design

In this section, we describe experiments designed to measure the parameters of dynamic Web page requests and present performance results obtained from running our experiments. In order to compare the performance of CGI, FastCGI and Servlets, we designed experiments to carefully measure the fundamental parameters of dynamic Web page generation technologies:

- *Input Size:* Amount of data the client sends to the server.
- *Output Size:* Amount of data that the server generates and sends back to the client.
- *Disk Write:* Amount of data the server writes to the disk on receiving a request from a client.
- *Disk Read:* Amount of data the server reads from the disk.
- *Computation:* The CPU load required by the server to service the client's request.

For the experiments, we measured the individual effects of each of these parameters on the performance of CGI, Servlets and FastCGI. The results presented in this paper represent the average of 10 experiment runs, where each run executes requests repeatedly for approximately 180 seconds.

We developed benchmarks to measure the above parameters and carried out a series of experiments to measure CGI, Servlets and FastCGI performance. The experiments consisted of running benchmarks on the client machine with specified request parameters and workload characteristics. Details on our benchmark are described in [13]. We performed the experiments on machines in single user mode on a dedicated network. In single user mode, the CPU runs a bare minimum of system processes and no other user processes.

Our server hardware platform was an Intel Pentium II 300 MHz system with 64 Megabytes of RAM. It had a standard 10 Megabits/second Ethernet card. The operating system was Linux version 2.0.35.

The server software was Apache, version 1.3.2, a public domain Web server. We ran the Apache Web server in stand-alone mode. There are pre-defined limits to the number of idle processes. The lower and upper bounds for our experiment were 5 and 10. The number of Keep Alive requests per connection was set to 0 (only one HTTP request was serviced per connection).

For adding Servlet support to Apache, we used JRun, version 2.2.1 [15]. For adding FastCGI support to Apache, we used the FastCGI Developer's Kit, version 2.0b2.

Our client hardware platform was an Intel Pentium 400 MHz system with 128 Megabytes of RAM. The operating system was Linux version 2.0.35. The workload was generated by our benchmark. The client was connected to the server by a dedicated 10 Mbps Ethernet network.

The CGI and FastCGI applications were written in C, compiled by gcc, version 2.7.2.1. All Servlets applications were written in Java and compiled with JDK version 1.1.6.

In the following sub-sections we present the results of each parameter on CGI, Servlet and FastCGI performance.

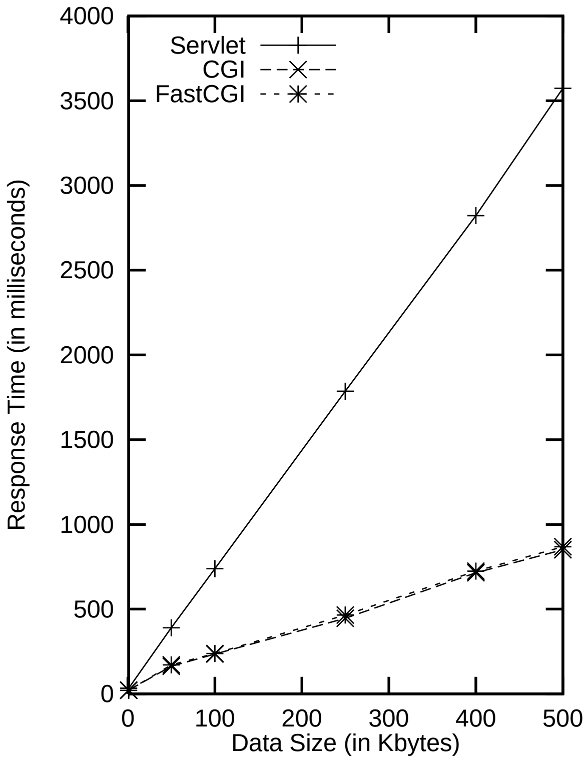

Figure 1: **Response Time vs Input Data Size. The horizontal axis is the size of the data sent from the client to the server. The vertical axis is the response time. The data points are the average time for a single request.**

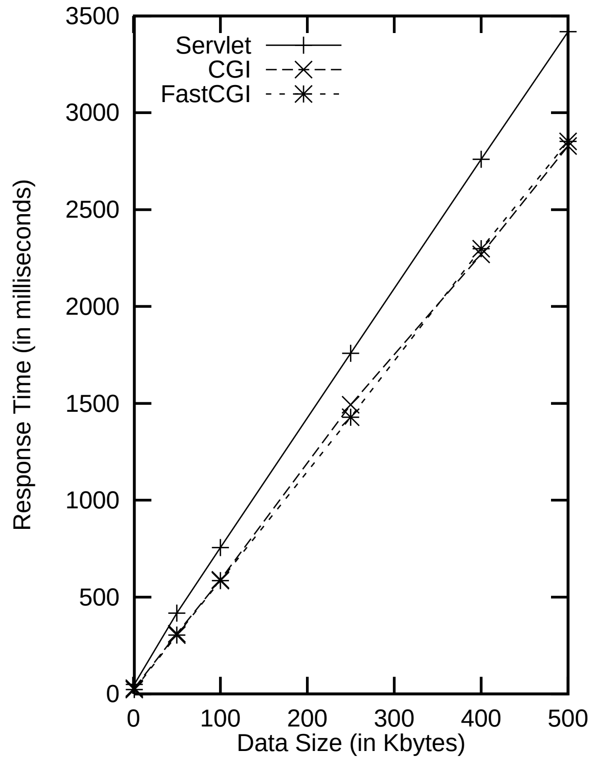

Figure 2: **Response Time vs Output Data Size. The horizontal axis is the size of the data sent back from the server to the client. The vertical axis is the response time. The data points are the average time for a single request**

2.2 Input Data Size

One of the main reasons for dynamic Web pages is to allow servers to process input data from a client. For example, a server application might be responsible for taking data filled in an HTML order entry form and applying business logic. We measured the affect of input data size on all the three technologies by having the client send POST requests to the server. Figure 1 depicts the performance of each of the server technologies obtained from the measurements for the input data size. The results show that CGI performs as well as FastCGI. For large data sizes, Servlets perform significantly worse than either CGI or FastCGI.

2.3 Output Data Size

No matter how much data the server application receives, it always has to send data back to the client. For example, typical search engines have server applications which dynamically generate Web pages and send these back to the client. We measured the affect of output data size by having the clients send GET requests to the Web server. The results in Figure 2 show that again FastCGI performance is the best, with CGI performing almost as good as FastCGI, even for small data sizes. The results clearly show that the cost of running Servlets is much more than the total cost of loading CGI programs into the memory of the server for each request and the cost to run the CGI programs.

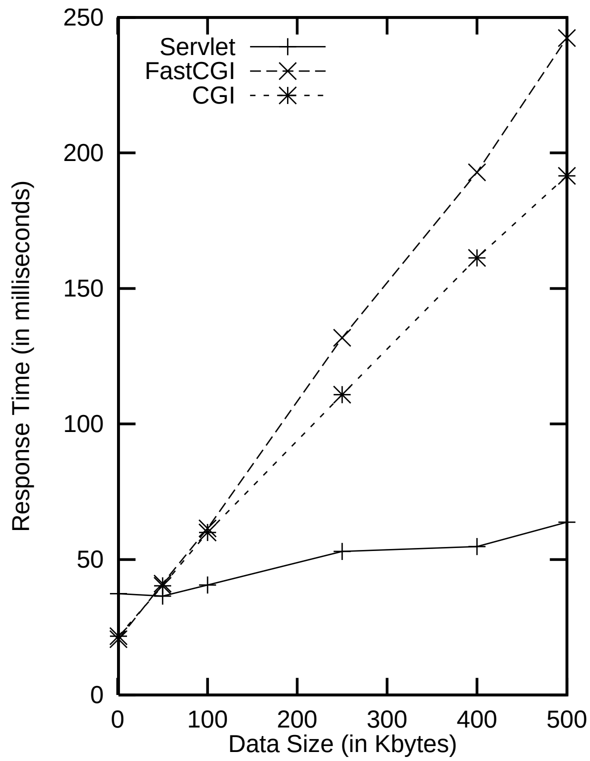

Figure 3: **Response Time vs Disk Read. The horizontal axis is the size of the data read by the server. The vertical axis is the response time. The data points are the average time for a single request.**

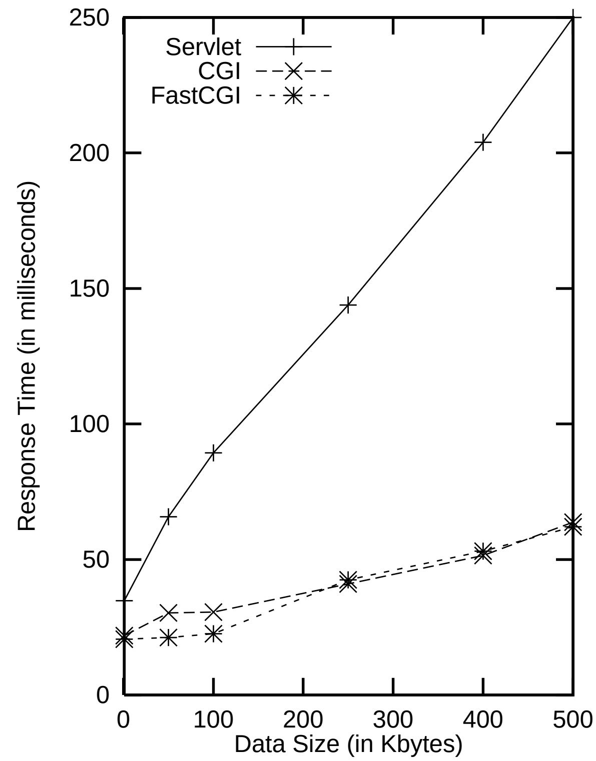

Figure 4: **Response Time vs Disk Write. The horizontal axis is the size of the data written on disk by the server. The vertical axis is the response time. The data points are the average time for a single request.**

2.4 Disk Read

The amount of data to be read from the disk depends on the role of the server application. For example, a server application might be responsible for retrieving chapters or sections of a book from the disk in response to a request on a typical virtual bookstore. Figure 3 shows the line equations for the performance of each of the server technologies obtained from the measurements for the data size read from the disk. Surprisingly, the results show that Servlets are able to do disk reads much more efficiently than CGI and FastCGI applications. For an increase of data size of 500 Kbytes, the increase in the response for Servlets is just around 40 milliseconds whereas for CGI, it is around 170 milliseconds. FastCGI in this test performs the worst out of the three.

2.5 Disk Write

It is quite common in business applications to record data to the disk. For example, a server application might be responsible for writing the data from a transaction entry form to the disk. In measuring the affect of disk-write, the clients sent GET requests to the Web server, with the amount of data to be written specified in the query parameters. Unlike the disk read results in subsection 2.4, Servlets take much longer time to do disk write than CGI and FastCGI application, as shown in Figure 4. Performance of FastCGI is better than CGI up to 250 Kbytes of data size and after that CGI does equally well as FastCGI.

2.6 Computation

The cost of computation is important to applications that do CPU intensive work. For example, on-line image processing servers might be required to run server applications to perform various graphic algorithms or different filtering techniques, which are CPU intensive. To measuring the affect of this parameter, the clients sent GET requests to the Web server, with the count for the computation specified in the query parameters. The server application

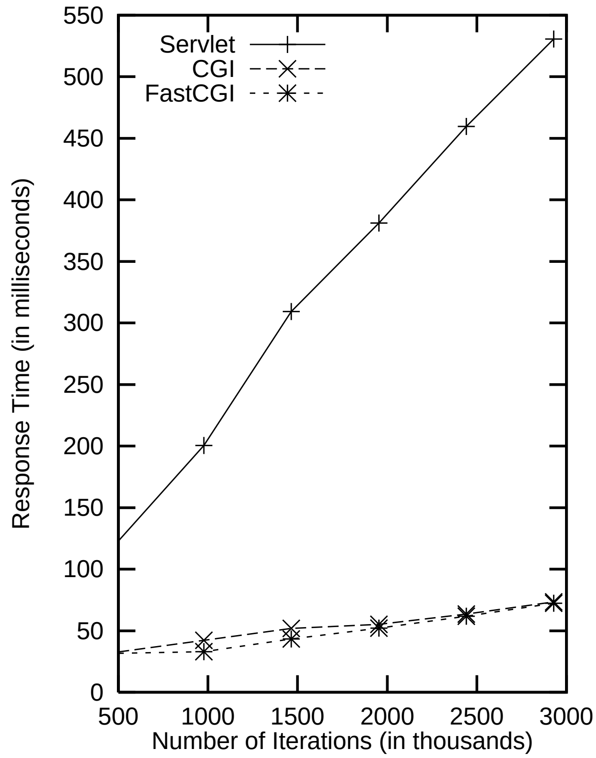

Figure 5: **Response Time vs Computation Time. The horizontal axis is the computation work by the server. The vertical axis is the response time. The data points are the average time for a single request.**

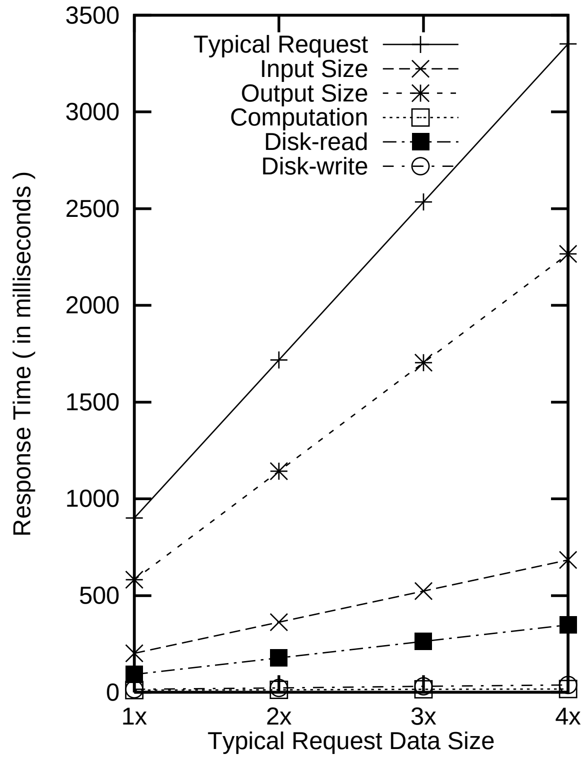

Figure 6: **Response Time vs Request Size. Weights, in terms of response time, as measured by our multivariate linear regression model for individual parameters for a typical large CGI request.**

incremented an int variable in a tight loop, counting up until the parameter specified. As soon as it completed the loop, the Web server is notified of the request completion and the Web server in turn, sends an acknowledgement back to the client. The Figure 5 shows the results obtained from the computation measurements. Servlets are much slower than CGI and Fast CGI.

3 Performance Model

This section describes how we use the results obtained and analyzed in the previous section in developing a multivariate linear regression model with categorical predictors for dynamic Web page generation technologies. Our model is extremely flexible for predicting the performance of the Web server for different applications. For example, an electronic commerce Web server would have a traffic pattern different than from a Search Engine site. By studying the pattern of the traffic and formalizing the data sizes of each of the variables used in our model, our model can predict the performance. Moreover, by varying the size of the individual parameters we are able to predict the bottlenecks in dynamic Web page generation performance.

Parameter	Size
Input Size	500 bytes
Output Size	2 Kbytes
Disk Read	4 Kbytes
Disk Write	1 Kbytes
Compute	1000 increments

Table 1: **A typical small request**

Parameter	Size
Input Size	50 Kbytes
Output Size	50 Kbytes
Disk Read	100 Kbytes
Disk Write	100 Kbytes
Compute	1 million increments

Table 2: **A typical large request**

Our analysis consisted of modeling request sizes of two types: typical large requests and typical small requests, based on the study done in [1]. Table 1 shows a typical small request.

Server applications like database transaction modules or image processing filters performs massive processing at the server and return more data than the smaller dynamic requests. Table 2 shows a typical large request.

A multivariate regression model [9] allows one to predict a response variable y as a function of k predictor variables $x_1, x_2, ..., x_k$ using a linear model of the following form: $y = b_0 + b_1x_1 + b_2x_2 + ... + b_kx_k + e$

Here $b_0, b_1x_1, b_2x_2, ..., b_kx_k$ are $k + 1$ fixed parameters and e is the error term. In vector notation, the model is represented as: $y = Xb + e$. Where: y is a column of n observed values of y; X is an n row by $k + 1$ column matrix whose $(i, j+1)^{th}$ element $X_{i,j+1} = 1$ if $j = 0$ else x_{ij}; b is a column vector with $k + 1$ elements; and e is a column vector with n error terms.

Our model is based on both quantitative and categorical predictors [9]. Our model has 8 predictor variables: input, output, disk read, disk write, computation time, operating system, Web server and dynamic page generation technology (ie- CGI, FastCGI and Servlets). The predictor variables: input, output, disk read, disk write and computation time are quantitative predictors, presented in Section 2.2 to 2.6. The three predictor variables: operating system, Web server and dynamic page generation technology are categorical variables. The categorical variable for dynamic page generation technology takes three values, CGI, Servlets and FastCGI. The categorical variable for operating system takes two values, Linux or Windows NT[1]. The categorical variable for Web server also takes two values, Apache or Netscape Enterprise.

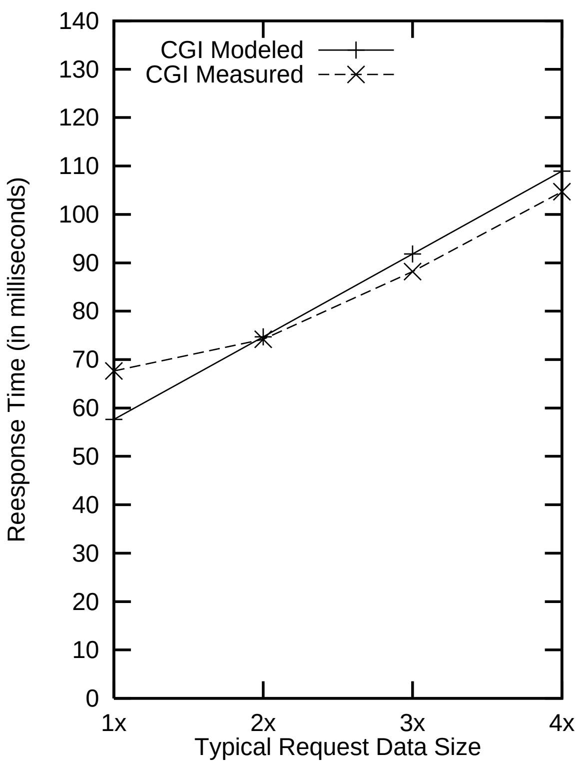

Figure 7: **Response Time vs Request Size. Performance, as predicted by our multivariate linear regression model model for a small CGI request and as measured by our benchmark.**

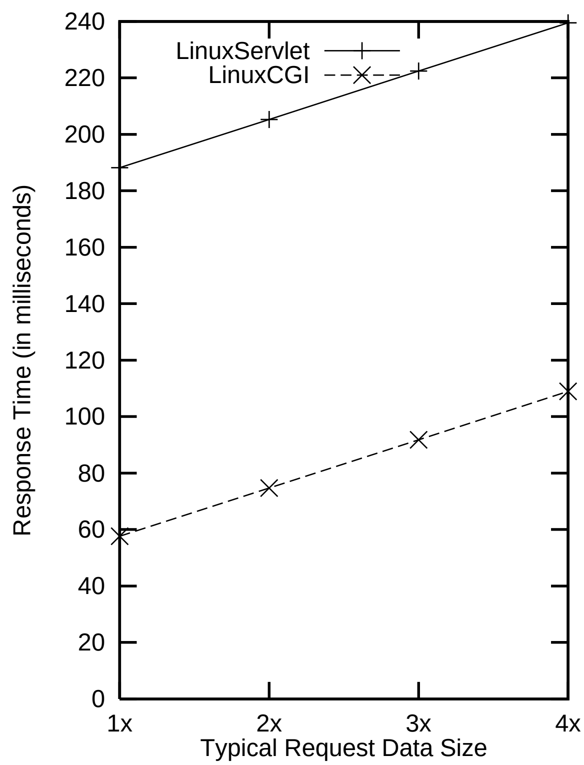

Figure 8: **Response Time vs Request Size. Performance, as predicted by our multivariate linear regression model, for dynamic page generation technologies for a typical request. The horizontal axis is the data size for a typical request which the client sends to the server.**

Based on the multivariate linear regression analysis in [9], we computed the multivariate linear regression coefficients for our model. The *coefficient of determination* for our model was 0.93. Thus the regression explains 93% of the variance of response time in our model.

In order to validate our performance model, we ran a typical small request. In doing so, we developed a CGI

[1]The experiments for the impact of operating system and Web server on the performance of dynamic page generation technology are described in [13].

application which handles and service the typical requests the benchmark generates. The benchmark sends a POST request to this CGI application. The data sizes ranged from `1x` to `4x`, where `1x` corresponds to the small request as described earlier in this section.

Figure 6 shows the individual weights of each of the parameters which amounts to the total weight of a typical large CGI request. A `1x` on the x-axis for individual components corresponds to their data size in a typical large request size as described earlier. A `2x` on the x-axis for individual components corresponds to twice the data sizes in the typical request size and so on. The result clearly shows that output parameter has a significant cost in terms of the response time for a CGI request.

Figure 7 shows the result of the evaluation of our model for typical request for CGI. The result shows that our model predicts the performance of CGI within 10% of measured values. We assume that differences of 10% in predictions made by our model are significant.

Figure 8 shows the total response time obtained on running our model for CGI and Servlets. CGI performs significantly better than Servlets. This clearly suggests that threads solely are not sufficient to achieve high performance. We attribute these results to the relatively small amount of time required for process creation in a CGI request. Our experiments in [12] show that the process creation time is of the order of 10 milliseconds whereas the response times here are of the order of 100 and 1000 milliseconds.

The results also show that Servlets are slower, having the drawback of being interpreted by a Java Virtual Machine. In the JDK interpreter, calling a synchronized method is typically 10 times slower than calling an unsynchronized method [16] [11]. With JIT compilers, this performance gap can decrease to as little as 5 times slower but the time to create an object does not improve at all. The time to create an object, 15 milliseconds, is high if the Java code creates many objects. Even with using Just In Time compilers and other optimization techniques, Java code is still anywhere from 10 to 30 times slower than compiled C or C++ code.

4 Conclusions

Dynamic Web page generation is becoming an increasingly critical component of Web server performance. Our goal was to empirically compare the performance of three dynamic Web page generation technologies: CGI, FastCGI and Servlets. We developed a benchmark tool for measuring the affect of the fundamental parameters of Web page generation technologies on the performance of the Web server. We also developed a model for predicting the performance of dynamic requests for a particular server technology.

Based on our analysis, CGI and FastCGI perform significantly better than Servlets. This results holds for all parameters but disk reads. Threads alone will not improve the performance of Servlets, since Servlets are inherently slower because of the dependency on a Java Virtual Machine. Even with the speed optimization techniques available, Servlets are still slower than compiled CGI (written in C) applications.

Future work includes studying other factors which may impact the performance of the dynamic Web page generation technologies. The concurrency strategy (single threaded, thread per request, thread pool, request pool, etc.) of the Web server [10], may influence the dispatch of the client request code, as well.

References

[1] Almeida, V.A.F, Almeida, J.M, Analyzing the impact of dynamic pages on the performance of Web servers, *Computer Measurement Group Conference, 1998.*

[2] Almeida, V.A.F, Almeida, J.M, Murta, C.D., Oliveira, A.A, and Mendes, M.A.S., Performance Analysis and Modeling of a WWW Internet Server, *Fourth Telecommunication Conference, March 1996.*

[3] Aoki, P., Woodruf, A., Brewer, E., and Gautheir, L., An Investigation of Documents from WWW, *Proceedings of the Fifth International Conference on WWW, May 1996.*

[4] Arlitt, M. and Williamson, C., Web Server Workload Characterization, *Proceedings of the 1996 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1996.*

[5] Arlitt, M. and Williamson, C., Internet Web Servers: Workload Characterization and Performance Implications, *IEEE/ACM Transactions on Networking, October 1997.*

[6] Banga, G., and Druschel, P., Measuring the Capacity of a Web Server, *USENIX Symposium on Internet Technologies and Systems ,USITS, 1997.*

[7] Douglas C. Schmidt, and James C. Hu, Developing Flexible and High-performance Web Servers with Frameworks and Patterns, *ACM Computing Surveys, May 1998.*

[8] Graphic, Visualization, & Usability Center's (GVU) 9th WWW User Survey, *http://www.gvu.gatech.edu/user_surveys/survey-1998-04/*

[9] Jain, R. (1991), *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc.

[10] James C. Hu, Sumedh M., and Douglas C. Schmidt, Techniques for Developing and Measuring High-Performance Web Servers over ATM Networks, *INFOCOM, 1998.*

[11] Java Microbenchmarks, *A report*, `http://www.cs.cmu.edu/~jch/java/benchmarks.html`

[12] Kothari, B., Claypool, M., PThreads Performance, *Worcester Polytechnic Technical Report, WPI-CS-TR-99-11, 1999.*

[13] Kothari, B., Claypool, M., Performance Analysis of Dynamic Web Page Generation Technologies.*Thesis Report, Worcester Polytechnic Institute, May 1998.*

[14] Krishnamurthy, D., Rolia, J., Predicting the QoS of an Electronic Commerce Server:Those Mean Percentiles, *Performance Evaluation Review, December 1998.*

[15] Live Software, Inc., JRun Servlet Engine, `URL:http://www.livesoftware.com/products/jrun/jrun-manual/index.htm`

[16] Make Java fast: Optimize, *An article, April 1997*, `http://www.javaworld.com/javaworld/jw-04-1997/jw-04-optimize.html`

[17] Manley, S., Seltzer, M., Courage, M., A Self-Scaling and Self-Configuring Benchmark for Web Servers, *Proceedings of the ACM SIGMETRICS '98 Conference, June 1998.*

[18] Mindcraft, Inc, WebStone 2.5, `URL:http://www.mindcraft.com/benchmarks/webstone/`

[19] Open Market, Inc., Fast CGI: A High-Performance Web Server Interface, *A Technical White Paper, April 1996.*

[20] Somin, Y., Agarwal, S. and Forsyth, M., Measurement and Analysis of Process and Workload CPU times in UNIX environments, *Proceedings of the CMG, 1996.*

[21] Standard Performance Evaluation Corporation, *A Technical White Paper, 1996*, SPECweb96, `URL:http://www.specbench.org/osg/web96/webpaper.html`

[22] Sun Microsystem, The JAVA Servlet API, *A Technical White Paper, 1998*, `URL:http://www.javasoft.com/marketing/coll-ateral/servlets.html`

[23] Wallace, R. and McCoy, T., Performance Monitoring and Capacity Planning for Web Servers, *Proceedings of CMG, 1996.*