WPI-CS-TR-99-12 January1999

Performance Analysis of Dynamic Web Page Generation Technologies

by

Bhupesh Kothari and Mark Claypool

Computer Science Technical Report Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Performance Analysis of Dynamic Web Page Generation Technologies

Bhupesh Kothari
Mark Claypool
{bhupesh,claypool}@cs.wpi.edu
Computer Science Department
Worcester Polytechnic Institute

Abstract

The 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 no 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 an analytical 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 analytic model shows the same deficiency in Servlets performance under typical dynamic Web page requests.

1 Introduction

Dynamic Web pages are an important medium for the exchange of information in today's business community. The surge in the use of the World Wide Web by business has created tremendous need for Web servers to create dynamic content. Electronic commerce servers use dynamic page generation technologies to access and present content to users distributed across Internet, thus helping further in the emergence of new dynamic page generation technologies, which are becoming more wide-spread. [16]. As a consequence of the increasing wide spread use of these technologies with the ever increasing growth of Web, Web servers are becoming more stressed then ever [15]. Performance of Web servers has become a critical issue.

The important movement away from static Web content to dynamic content 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 CGI has some significant drawbacks, one of which is that a new process is created and destroyed for each request. New approaches to move beyond CGI include:

- *Server API:* Server APIs such as Perl module [18] and Python [17] allow dynamic page generation to be incorporated directly into the Web server process. Server APIs do not have the CGI performance cost of process startup/initialization, since the applications runs in the server process and are always persistent across requests. However, Server APIs are Web server specific. More importantly, since Server API modules run in the server process space, a buggy module can corrupt other modules or even bring down the whole Web server.
- *Fast CGI:* Fast CGI removes the inherent CGI performance problem by making the CGI processes persistent; after finishing a request, the Fast CGI processes waits for a new request instead of exiting [21]. But the Fast CGI interface may involve heavy-weight process context switching on each request, which is costly in terms of CPU usage and thus can degrade Web server performance.
- *Servlets:* Servlets are protocol and platform independent server side components which run in a resident Java Virtual Machine as threads [22]. Servlets are basically Java classes which are loaded and invoked by the Web server. They are loaded only once instead of being spawned at every request. Servlets inherit the advantage of Java language, such as cross-platform portability, garbage collection and independence from proprietary servers. Servlets still have the drawback of being interpreted by a Java Virtual Machine [19] [20] .

Web servers can easily be configured to deliver static pages to high number of concurrent users without substantial performance degradation. Moreover, performance analysis of Web servers for static requests has been studied [1] [6] [7] [13] and modeled [6] [12]. However, Web server performance for dynamic documents has neither been experimentally measured nor thoroughly analyzed. In order to determine the technology that gives best performance, the costs and benefits of each need to be weighed. Benchmark tools for measuring the server side technologies exists only for CGI.

This paper presents the performance of the above server side technologies: Servlets, CGI and Fast CGI. We show that, contrary to the common belief, performance of CGI is comparable or better when compared to other dynamic page generation technologies. We developed our Web server benchmarks for measuring Servlets, Fast CGI and CGI performance and present experimentally based results of the performance of each technology. We also identify factors which impact their performance. Lastly, we developed an analytical model based on our experimental results that predicts the performance of the Web server for different workloads. Our analysis will help in comparing bottlenecks in the performance of dynamic page generation technologies to potential bottlenecks from Web server concurrency alternatives and operating system selection.

2 Related Work

Much related work has developed and measured high-performance Web servers [4] [5], through Performance Analysis, Modeling or Benchmarking.

2.1 Performance Analysis

Studies such as [6] look at the evolving standards for performance measuring, monitoring, and benchmarking Web servers. A lot of work had been done on looking for the characteristics of the Web that makes it different from traditional distributed systems. Arlitt and Williamson in [1] [2] did a workload

characterization for Web servers. Basically, their study showed the types of objects available at Web servers and analyzed relevant aspects such as file size distribution. Woodruf et al [7] examined characteristics of a large number of HTML documents. Their results showed that for most Web servers, the request load consists of normal HTTP requests, however there were some sites that had primarily CGI requests. For sites like these, the CGI technologies impact the Web server performance a lot. Our work proposes to build upon these results by providing CGI performance measurements.

Schmidt et al [4] presents empirical analysis illustrating how dynamic and static adaptivity can enhance Web server performance. They examined different concurrency strategies (thread pool, process pool, etc.) and their effect to different workload characteristics. Their study concentrated on the design techniques and optimizations for developing high-performance Web servers.

2.2 Modeling

Almeida et al modeled Web server performance by building a simple queueing network model representing the server architecture [12]. Modeling Web server performance has been done by very few because of the complexity to develop a realistic model which captures the true nature of the Web server. A brief on some performance models for sizing Web servers can be found in [6]. Krishnamurthy and Rolia studied Quality of Service (QoS) measures and Service Level Agreements (SLA) for an electronic commerce server [14]. They developed an analytical model based on Mean Values Analysis for predicting QoS in terms of response time. The performance results from our research may be useful in building performance models for dynamic page generation technologies.

2.3 Benchmarking

A considerable amount of work had been done in Web server benchmarking. Banga and Druschel [10] looked into the problems with existing benchmarks, such as simulating a large number of potential clients with a given number of test machines. A very similar study was done in [9] in which the authors developed *httperf*, a tool for measuring Web server performance. Another project developed a tool, *hBench:Web*, which is a self-scaling and self-configuring benchmark for the Web [11]. WebStone [23] and SPECweb96 [24] are industry standard benchmarks for measuring Web server performance. We will develop benchmarks that will stress the server with different CGI workloads.

3 Experiments

3.1 Design

Performance of a Web server is dependent upon a number of variables: the server hardware and operating system, server software, network speed, and workload. Performance also depends on the client environment: client hardware platform and operating system. Our approach was to identify the parameters that affect CGI, Servlet and FastCGI performance by keeping all the other variables constant. Thus for all our performance measurement experiments we had the same client environment and the same server environment. The parameters that influences these technologies are:

- *Output Size:* Amount of data that the server generates and sends back to the client.
- *Input Size:* Amount of data the client sends to the server.
- *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 clients request.

For the experiments, we measured the individual affects of each of these parameters on the performance of CGI, Servlets and FastCGI.

We developed benchmarks to measure the above metrics. A series of experiments were carried out to measure the CGI, Servlet and FastCGI performance. The experiments consisted of running the benchmark tool on the client machine with specified request parameters and workload characteristics. The benchmark tool makes a master process which reads the configuration file that contains the dynamic application technology to be tested and the parameter for which it is to be tested. Depending on the request, it would then read the data size which the server application would process.We used a predefined set of data to be processed by the server application and measured the results returned by the system under examination. The dataset was a mix of request type, size, number of files, and the arrival rate of the requests. Our benchmark makes a number of HTTP 1.0 GET requests to the Web server. The only test in which it makes a POST request is one for the measurement of input-size parameter. For all the experiments presented in this paper, we ran just a single client as the aim here was to isolate the individual performance impact of CGI, Servlet and FastCGI on the Web server performance. For the same reason, we performed the experiments on machines in a single user mode. In single user mode, the CPU runs a bare minimum of system processes and no other user processes.

The standard form of the URL in all our tests for a GET Request was:
http://host-name/technology/server-script?qp
where:
host-name : saagar.wpi.edu:8080
technology : CGI, Servlet or FastCGI
server-script: name of server application
qp : query parameters

For example, a typical CGI request is of the following form:
http://saagar.wpi.edu:8080/cgi-bin/diskRead?size=1024

The URL for a POST request is exactly of the same form, except without the query parameters.

3.2 Test Environment

This section describes the server and client environment, hardware and software, that we used to perform the Web server performance tests.

Our server hardware platform was an Intel Pentium II 300 MHz system with 64 Megabytes of main memory. 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 HTTP server. The October 1998 WWW server site survey by Netcraft found that over 52% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined [25]. Apache can run in two different modes: from the inetd system process or, in stand-alone mode. For our experiments, we ran the Apache Web server in stand alone mode. Apache uses a process pool concurrency strategy. It maintains a pool of child processes to handle incoming requests. On startup, a master server process spawns a pre-defined number of child processes and as the load in the server increases, new processes are spawned and included in this pool. The master process periodically checks the number of idle child processes and dynamically adapts this number to the load it sees. 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 , a Servlet engine [26].

For adding FastCGI support to Apache, we used FastCGI Developer's Kit, version 2.0b2. The Web server and the FastCGI process communicates over Unix domain or TCP sockets. These processes use the FastCGI protocol to write and read the data from the socket.

Our client hardware platform was an Intel Pentium II 300 MHz system with 64 Megabytes of main memory. The operating system was Linux version 2.0.35. The workload was generated by our Benchmark tool. Client was connected to the server by a dedicated network.

The CGI and FastCGI applications were written in C. The gcc compiler version used in measurements, on both server and client, was 2.7.2.1. Servlets applications were written in Java and version 1.1.6 was used for all the measurements, on both server and client.

4 Results

In the following sub-sections we describe the affects of each parameter on the CGI, Servlet and FastCGI performance. The results presented here represent the average of 10 experiments, in similar conditions of load and network traffic. The last sub-section describes the analytical model that predicts the performance of the Web server.

4.1 Input Data Size

One of the main purpose the server applications are used is to process the data it receives from the 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 request to the server. Each client made a new request just after the acknowledgement of the current request was received from the server. The server application read and parsed the input data it received. When its done reading, the Web server sent an acknowledgement back to the client. The client then only makes another request. A typical Servlet request for this test was of the following form:
http://host-name/servlet/inputSize

Figure 1 shows the line equations for the performance of each of the server technologies obtained from the measurements for the input data size. The results show that CGI perform as well as FastCGI. Servlets perform a lot poor as compared to other two.

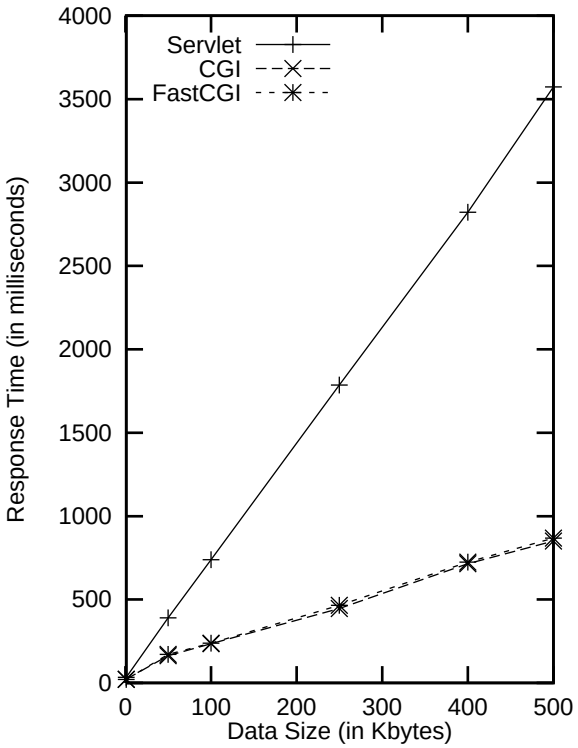

Figure 1: **Response Time vs Input Data Size. The horizontal axis is the size of the data sent from the client to the server. The vertical axis is the response time. The data points are the average time for a single request**

4.2 Output Data Size

No matter how much the data the server application receives, it almost always has to send data back to the client. For example, search engines have server application which are responsible to locate and collect the data and send it back to the client. The pages which the client receives in response to a search query are all dynamically generated by these server applications.

We measured the affect of this parameter on all the technologies, by having the clients send GET request to the Web server. A typical CGI request for this was of the following form:
http://host-name/cgi-bin/outSize?size=1024

Each client sent the data size in its query parameters. After parsing the request, the server application generated data equal to the size it received and sent the data back to the client. The client waited in a loop till it received all the data and as soon as it finished reading all the data, the benchmark tool recorded the statistics and made the client sent another request. Figure 2 shows the line equations for the performance of each of the server technologies obtained from the measurements for the output data size.

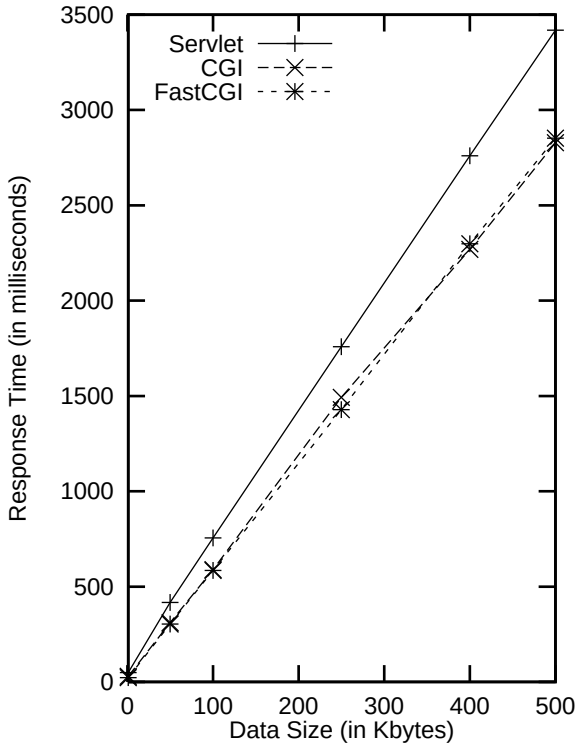

Figure 2: **Response Time vs Output Data Size. The horizontal axis is the size of the data sent back from the server to the client. The vertical axis is the response time. The data points are the average time for a single request**

The results show that again the FastCGI performance is better than the other two, with CGI performing almost as good as FastCGI. Even for small data sizes, the amount of data to be sent back, impacts the performance of Servlets. 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.

4.2.1 Disk Read

The amount of data to be read from the disk will depend on the role of the server application. But if disk read is the majority of the work the server application is doing, then this parameter certainly will dominate the performance of the server application.

We measured the affect of this parameter, by having the clients send GET requests to the Web server, with the data size to be read as the query parameter. A typical FastCGI request for this test has the following form:

http://host-name/fastcgi-bin/diskRead?size=1024

The results show that Servlets are able to do disk read in a much more efficient way than CGI and FastCGI applications. This has to be attributed to the way Java does disk read. 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 apparently performs the worst out of the three.

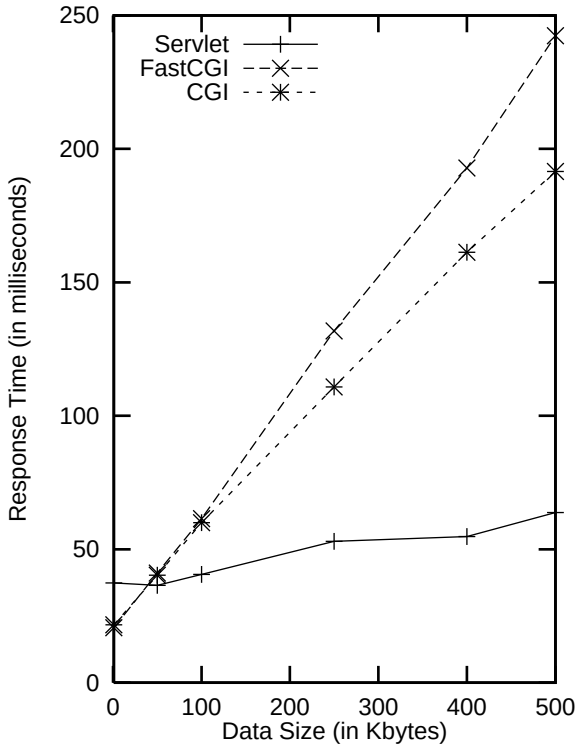

Figure 3: **Response Time vs Disk Read. The horizontal axis is the size of the data read by the server. The vertical axis is the response time. The data points are the average time for a single request**

4.3 Disk Write

It is quite common in business applications to have some mechanism of data recording. The amount of data written is important to applications which do data keeping. For example, a server application might be responsible for writing the data, filled in a transaction entry form, to the disk.

In measuring the affect of this parameter, the clients were sending GET requests to the Web server, with the amount of data to be written specified in the query parameters. A typical FastCGI request for this test has the following form:
http://host-name/fastcgi-bin/diskRead?size=1024

Surprisingly, results show that Servlets take much longer time to do disk write than CGI and

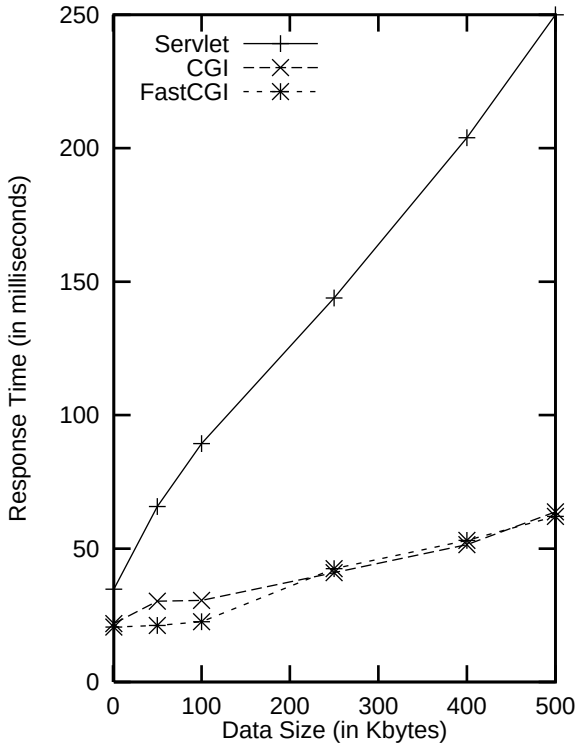

Figure 4: **Response Time vs Disk Write. The horizontal axis is the size of the data written on disk by the server. The vertical axis is the response time. The data points are the average time for a single request**

FastCGI application. The factor of time taken to write to data size is opposite to the one obtained in disk read test. Performance of FastCGI is better than CGI until 250 Kbytes of data size and after that CGI does equally well as FastCGI.

4.4 Computation

The cost of computation is important to applications that do CPU intensive work. For example, a chess game applet where the server application might be required to execute search algorithms.

In measuring the affect of this parameter, the clients were sending GET requests to the Web server, with the count for the computation specified in the query parameters. The server application increments a `int` variable in a tight loop. As soon as it completes the loop, the Web server is notified of the request complete and the Web server in turn, sends an acknowledgement back to the client. A typical CGI request for this test has the following form:
`http://host-name/cgi-bin/compute?count=1000000`

The Figure 5 shows that results obtained from the computation measurements. As expected, Servlet were much slower than CGI and FastCGI.

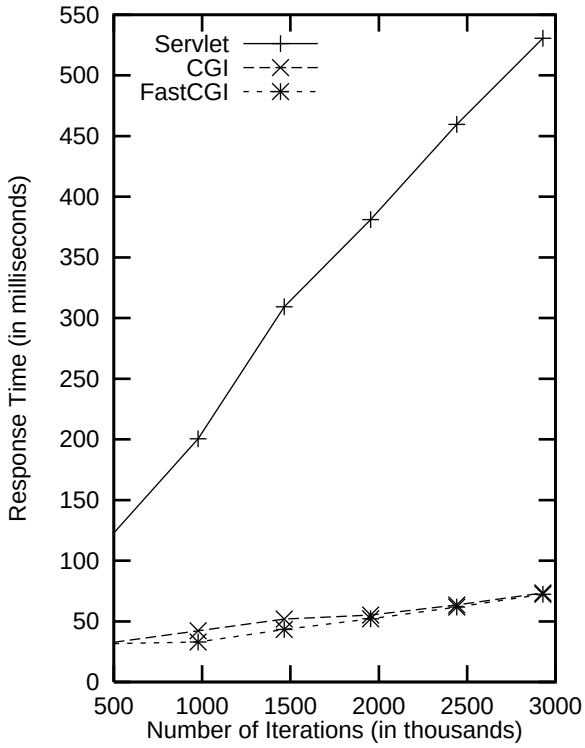

Figure 5: **Response Time vs Computation Time. The horizontal axis is the computation work by the server. The vertical axis is the response time. The data points are the average time for a single request.**

5 Analysis

Based on the above results, we developed a model based on the parameters I used in our experiments. For each of the experiment, we plotted a best linear-fit. The Linear function for each of the experiments is of the form:
$\mathbf{y = ax + b}$.

Table 1 shows the values for the constants 'a' and 'b', for all the parameters measured.

Our aim here was to predict the performance of each of the server technologies for a typical request that is seen in a real-world.

This model could be quite powerful in predicting the performance of the Web server, used for specific purposes. 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, this model would be able to predict the performance.

Figure 6 shows the results obtained on running this model.

Parameter	CGI	FastCGI	Servlet
Input Size	1.48	1.55	7.12
	65.33	60.05	28.38
Output Size	5.61	5.67	7.08
	36.00	16.98	307.77
Disk Read	0.34	0.44	0.05
	23.78	18.99	35.91
Disk write	0.07	0.08	0.41
	23.24	17.68	41.51
Compute	0.01	0.02	0.16
	25.92	19.01	43.94

Table 1: **Constant values of line equations for CGI, FastCGI and Servlet. The upper value in each cell is the value for constant 'a' and the lower value is for constant 'b' .**

With the results on performance of individual technologies, the above results were more than expected. The Servlets certainly will not perform better just because they are invoked as threads. The experimental analysis show that CGI performance is not only a lot better than what is assumed, it also performs almost equally good as FastCGI. Servlets has the poorest performance of the three. This clearly suggests that threads solely are not sufficient to achieve high performance. Moreover, our experiments in [8] show that the process creation time is of the order of 10 milliseconds whereas the response time measured 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. With JIT compilers, this performance gap has increased to 5-20 times but the time to create an object hasn't improved at all. The time to create an object, 15 milliseconds, seems high if the Java code creates a lot of them. 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. With these statistics, the poor performance of Servlets attributes to the slow interpretation speed by Java Virtual Machine. Another drawback associated with the Servlets is of being a proprietary technology and hence not every Web server understands it. Until now Java Web Server was the only server to fully support the servlet API. The current Servlet Engines too have some inherent drawbacks. A web server simply acts as a client that talks indirectly to the services, in order for a web server to talk to JRun Servlet Engine (JSE), it needs to establish some sort of connection. JRun provides and installs a native connector in the web server that is responsible for making the connection from the web server to JSE. When a request comes in for a servlet, the native connector opens a socket connection to a Network Proxy Service residing within JSM. The JRun Proxy Service serves as a transparent communicator and translates the request coming from the native connector on to the JSE Service. The JSE Service handles the request and sends its reply back to the Network Proxy service. The JSE Service does not know how to talk through sockets this is why it relies on the Network Proxy service to talk through the socket back to the web server. How well the Network Proxy Service is managing the sockets connection is not well know. If it takes a long time to establish a socket connection or if there is a limitation on the number of sockets connection this service can handle, the benefit that Servlet is invoked as threads is lost.

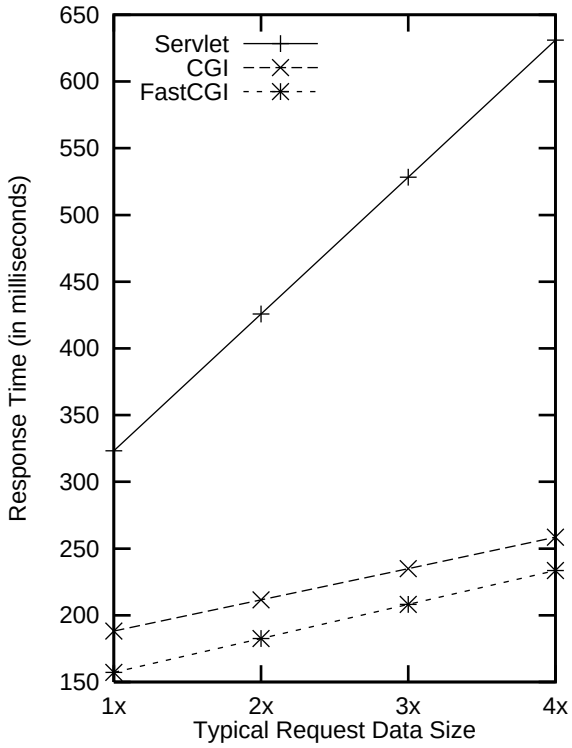

Figure 6: **Performance, as measured by our analytical model, for dynamic page generation technologies for a typical request. The horizontal axis is the data size for a typical request which the client sends to the server. A typical request is a total of all the data sizes for different parameters. A one on this axis means a typical request of the following construct:**
input data size : 500 bytes.
output data size : 2 Kbytes
data read from disk : 10 Kbytes
data written to disk : 1 Kbytes
amount of computation: 500 incrementations

6 Conclusions

Our goal was to identify the variable that affect server side technologies. We developed a benchmark tool for measuring the affect of each of these variables on the performance of server side technologies. We also developed a model for predicting the performance of a Web server for a particular server technology, given its nature and size.

Based on our analysis of individual technologies, CGI and FastCGI still have a lot better performance as compared to Servlets. This results holds for all the parameters but for the disk read. Given this, CGI and FastCGI still perform better as the results obtained by our model predicts. Threads solely will not improve the performance of Servlets. Servlets are inherently slower because of the dependency on Java Virtual Machine. Even with the speed optimization techniques available, Servlets are still slower than compiled CGI(written in C) applications.

There are a lot more factors on which the performance of the dynamic Web page generation

technologies depend. Performance impact on these technologies because of Web Server design and operating system selection still needs to be analyzed and measured.

7 Future Work

The results obtained in this research identifies the parameters that affect the Server side technologies. The benchmark we developed measures the affect of each of these technologies on the Web server performance. The next direction would be to measure and analyze the Web server performance results obtained by running more than one client. This would mimic the real-world traffic patterns and would help in analyzing the results in more directions, like after how many clients would the CGI performance would become exponential. This could still be taken further by running the experiments on realistic intranet applications, one that involves transactions with database. CGI application would open and close a new database connection for each request. This could definitely degrade the performance the a Web server.

Having done the measurements from client standpoint, the next future work is to monitor the system behavior from the server standpoint. Processor, disk and memory utilization could be measured for different workloads. These measurements will give us a measure of resource usage at the server side.

When a HTTP request arrives, it is dispatched to code in the server which processes the request, depending on whether the request was a CGI one or a normal one. This dispatching is controlled by the concurrency strategy (single threaded, thread per request, thread pool, request pool, etc.) of the Web server [4]. Another future direction could be to analyze the effect of these concurrency strategies with respect to server side technologies. To do this, experiments could be done on different Web servers, like on an Apache Web server which has a pre-forked process pool and on a Netscape Web server which has a thread pool concurrency strategy.

Performance of CGI request processing may be very dependent upon an operating system's thread and process management. Another rich research could be the study of effects of the operating system on the performance of Web server CGI's by running our Web server benchmarks on a couple of different operating systems.

References

[1] Arlitt, M. and Williamson, C., Web Server workload characterization, *Proceedings of the 1996 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1996.*

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

[3] Somin, Y., Agarwal, S. and Forsyth, M., Measurement and analysis of process and workload CPU times in UNIX environments, *Proceedings of the CMG, 1996.*

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

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

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

[7] 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.*

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

[9] Mosberger, D, and Jin, T., httperf - A Tool for Measuring Web Server Performance, *Workshop on Internet Server Performance, Madison, Wisconsin, June 1998.*

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

[11] 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.*

[12] 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.*

[13] Almeida, J.M., Almeida, V.A.F., Yates, D.J., Measuring the Behavior of a World-Wide Web Server, *Proceedings of the Seventh Conference on High Performance Networking, April 1997.*

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

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

[16] IBM e-business Site,
URL: http://www.ibm.com/e-business/

[17] PyApache, Python module for Apache,
URL: http://www.msg.com.mx/pyapache/

[18] The Apache/Perl Integration Project,
URL: http://perl.apache.org/

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

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

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

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

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

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

[25] Netcraft, WWW Server Usage, *URL:http://www.netcraft.com/Survey/*

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