Web Server Performance Under Heavy Loads

Arun Iyengar, Ed MacNair and Thao Nguyen
IBM Research Division
T. J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598

Abstract

Web server performance is a critical issue for sites which service a high volume of requests. This paper examines the performance of Web servers under high CPU loads. Simulations are used in conjunction with workloads which were obtained by analyzing Web logs and performance data from several real sites. Performance is significantly affected by the percentage of requests for dynamic HTML pages; dynamic HTML pages adversely affect server performance. In order to optimize performance, the number of dynamic pages should be kept as low as possible. When dynamic pages are required, techniques such as fast API's for invoking server programs and caching can be employed to keep the overhead of server programs generating the dynamic pages as low as possible.

When the server is operating at or near peak capacity, there is a trade-off between average latencies and the percentage of requests rejected; performance is improved by rejecting a higher percentage of requests. For the real request distributions which we encountered, we found that Web servers should reject enough requests so that the average load on the system is 95% or less of the maximum capacity in order to prevent latencies from becoming too large.

1 Introduction

The explosive growth of traffic on the World Wide Web makes performance a critical issue for many Web servers. During peak periods, a Web server might have to service several requests per second. If the server cannot adequately handle the request traffic, the server will fail to satisfy some requests and result in unacceptably slow responses for other requests.

This paper analyzes the performance of Web servers using simulations incorporating input parameters from actual systems. It focusses on situations where CPU processing power is the limiting resource. There

are two types of requests which can be satisfied by Web servers: requests for static files and requests for dynamic pages created by programs executing on the server. The probability of the CPU being a bottleneck increases with the percentage of requests for dynamic pages.

Web servers can typically satisfy several hundred requests per second for static files when the files are not large enough to overwhelm the network. By contrast, the number of requests for dynamic pages which can be satisfied per unit time is often less than the corresponding number for static files by a factor of around ten, even if the dynamic pages are created by programs which consume few CPU cycles. In these situations, the interface for invoking server programs is a major source of overhead. Programs are usually invoked by Web servers via the Common Gateway Interface (CGI) [7]. Programs invoked in this manner are known as *CGI programs.* Web servers usually implement CGI by forking off a new process for each invocation of a CGI program. These processes terminate after the CGI program completes. The overhead for forking new processes is substantial.

Additional overhead is incurred by the work performed by CGI programs. For example, many Web sites are hooked up to databases. Hypertext links allow clients to invoke CGI programs which query and/or update the database. Many databases require processes which access a database to connect to the database. A process connecting to the database might have to provide a valid user ID and password. The process of connecting to databases is expensive. Servers can typically make about one to three database connections per second before the CPU reaches 100% utilization. We have observed that the number of requests per unit time for dynamic pages which a Web server can handle is often less than the corresponding number for static files by a factor of 100 or more when the dynamic pages are created by programs which open new connections to databases.

In order to reduce the overhead of the Common Gateway Interface, a number of techniques have been developed which could become commonplace in the near future. The basic approach is to allow clients to execute server programs without spawning separate processes each time. This can be accomplished by linking server programs directly with the Web server or preforking multiple processes or threads which the Web server communicates with to invoke server programs. IBM's Internet Connection Application Programming Interface (ICAPI), the Netscape Server API (NSAPI) [8] and the Microsoft Internet Application Programming Interface (ISAPI) [5] use the first approach while FastCGI [9] by Open Market uses the second approach.

The network can also act as a performance bottleneck. Usually, it is the links between a client or server to the Internet backbone which is the limiting factor. The bandwidth of the Internet backbone is generally not a limiting factor. A Web server is typically connected to the Internet backbone via ISDN (Integrated Services Digital Network, 128 Kb/s), a T1 link (1.5 Mb/s), or a T3 link (45 Mb/s). For a Web server hooked up to a local area network (LAN) which is often the situation for Intranets, common LAN speeds are 10

Mb/s for Ethernet and 100 Mb/s for fast Ethernet (FDDI).

Figure 1 shows the maximum throughputs which servers can sustain based on network bandwidth. It is only necessary for the network to provide enough bandwidth to support the maximum connection rate which the CPU can handle. For efficient Web servers running on fast uniprocessors, this number is typically several hundred but less than 1000 per second.

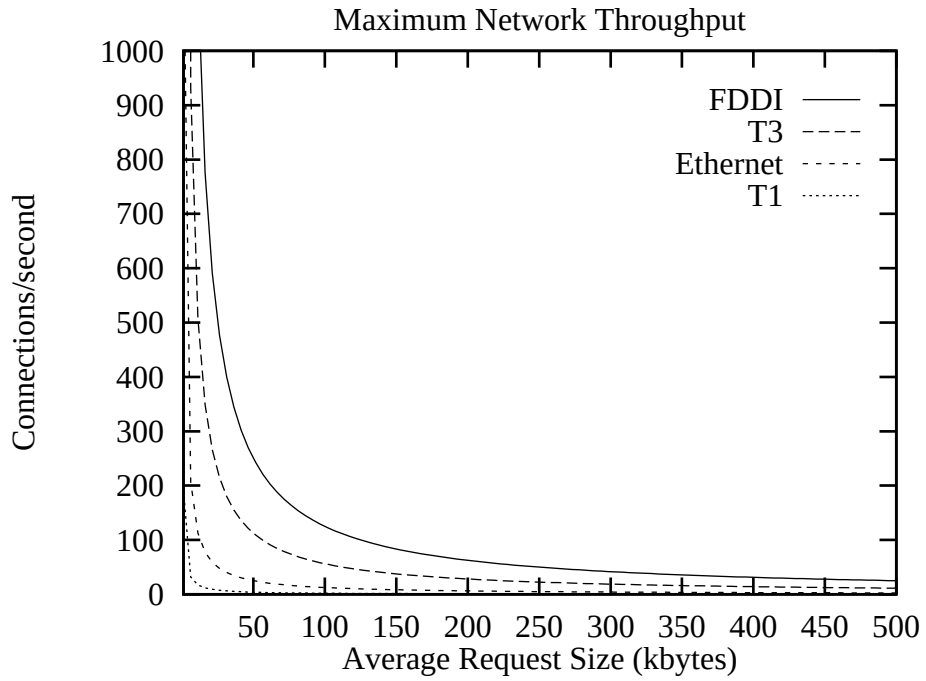

Figure 1: The maximum throughput which can be sustained by different types of networks.

Web clients are often connected to the Internet with slow links (e. g. 28.8 Kb/s or less). For such clients, the real bottleneck for accessing Web sites is the client connection to the Internet. There is little that the Web site can do to improve performance for such clients other than to keep request sizes small. The problem is compounded by casual or inattentive users who fail to configure their machines to utilize the maximum modem bandwidth.

If many slow clients are making requests to the Web server, the performance of the Web server for satisfying all requests can be adversely affected. This is because the Web server has to maintain open connections with each client while a request is being satisfied. For slow clients, connections will remain open for a long time. As a result, the Web server must maintain more open connections. As the number of open connections increases, the Web server responds more slowly and may even begin refusing new connections.

If I/O bandwidth at either the server or the client is a problem, one of the best ways to improve

performance is to reduce the average request size. In many cases, it is easy to do this while preserving useful information at a Web site. A high percentage of the bytes transferred from Web sites results from image files purely for aesthetic purposes. Eliminating or reducing the sizes of these images could reduce bottlenecks at both the server and client end. By contrast, ASCII text consumes little I/O bandwidth and often conveys the bulk of the useful information at a site.

1.1 Previous Work

Our study differs from others in that it makes more extensive correlations of latencies, rejected requests, and throughputs as a function of workloads when these quantities are limited by the CPU than any previous one. Mosedale [6] presents techniques learned from managing Netscape's Web site which receives over 120 million hits per day. Yeager and McGrath [12] provides a good overview of Web server design and what factors affect performance. Kwan [2] analyzes the performance of NCSA's Web server. Dias [1] examines the performance of scalable Web servers running on multiprocessors.

A number of benchmarks have been developed for measuring the performance of Web servers such as WebStone [10] and SPECweb96 [11]. These benchmarks test the maximum connection rate which can be sustained by issuing enough requests to saturate Web servers. The WebStone and SPECweb96 Web sites contain links to performance data obtained from using these benchmarks.

2 Methodology

We simulated a heavily loaded Web server and determined the distribution of request latencies for different sets of parameters. Our simulations are applicable to situations where a significant percentage of pages are created dynamically and Web server performance is limited by the processing power of the server and not by the network. We varied a number of request distribution parameters including the following:

- The proportion of requests for dynamic pages, f_{cgi}. Values of f_{cgi} around .2 and above are considered high. This is because dynamic pages often include image and other multimedia files. Sites which generate close to 100% of all Web pages dynamically often receive well over 50% of all requests for static files (usually images) included within the dynamic pages.

- The coefficient of variation for the distribution of request interarrival times, c_v. This quantity is a measure of the burstiness of distributions and is equal to the standard deviation divided by the average. Higher values indicate higher degrees of burstiness. An exponential distribution has a value of $c_v = 1$. We studied several request distributions from actual Web sites and found that c_v generally varies from 1 to 4 when measured over one hour intervals where the average request rate was roughly constant.

- The ratio of the average CPU time for satisfying dynamic requests to the average CPU time for satisfying static requests, t_{cgi}. The overhead of CGI is high enough so that t_{cgi} is around 10 on many platforms for very simple CGI programs. Server programs which open up a new connection to a database often have t_{cgi} values of 100 or greater. All of the results presented in this paper use $t_{cgi} = 100$ except for the results summarized in Figures 14 and 15.

- The normalized request rate, r, which is the average request rate divided by the maximum request rate the server can sustain. All of the results presented in this paper use $r = 1$ except for the results summarized in Figures 12 and 13.

The Web server is multithreaded with 25 threads. We ran experiments with varying numbers of threads and didn't find significant differences in our results. In order to prevent requests from being rejected during peak periods, a queue of requests is maintained which begins to fill up after all threads become busy. The Web server begins rejecting requests as soon as the queue becomes full. In our simulations, the normalized request rate, r, never exceeded 1. Therefore, it is possible to make the rejection probability as low as one wants by increasing the queue length. However, increasing the queue length also increases the average request latency. There is thus a trade-off between average request latencies and rejection probabilities.

Threads are scheduled using processor-sharing which is an idealized version of round-robin scheduling. We also tried round-robin scheduling using different time slices and got similar results. The average CPU time for satisfying requests for static files was 5 milliseconds. The server could thus handle 200 requests per second for static files. However, the results we present should also be valid for servers with other throughputs. For a server which can handle c static files per second, the latencies we present should be multiplied by $\frac{200}{c}$.

Processing rates for both static and dynamic pages were distributed with coefficients of variation of 0.6.

3 Results and Discussion

Figures 2-9 illustrate how performance degrades with increasing proportions of dynamic pages. The normalized request rate r was 1 for all of these cases. Request distributions were bursty and at peak periods, the server was receiving more requests than it could handle. In order to prevent latencies from becoming too large, the Web server limited the length of the queue of backlogged requests and rejected requests after the queue became full.

In order to improve performance, the percentage of dynamic pages should be kept as small as possible. If dynamic pages must be used, there are a number of techniques for reducing the overhead:

- More efficient interfaces such as ICAPI, NSAPI, ISAPI, and FastCGI can be used instead of the Common Gateway Interface for invoking server programs.

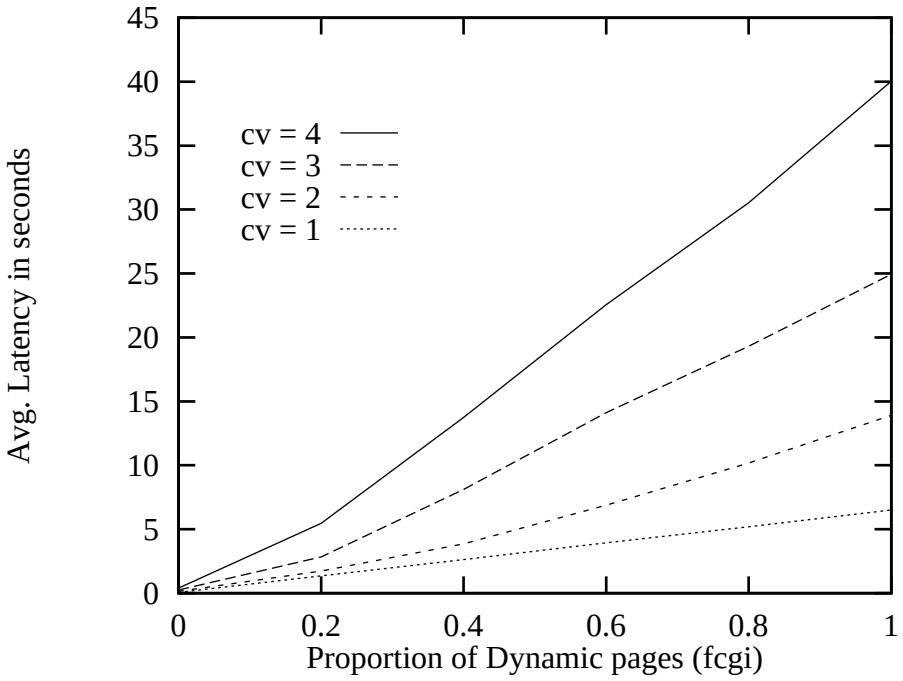

Figure 2: Average latency as a function of the proportion of dynamic pages (f_{cgi}). Each curve represents a request distribution with a different coefficient of variation (c_v) for the distribution of interarrival times. Request rejection rates were around 5%. The queue lengths for achieving these rejection rates are shown in Table 1.

c_v	*Queue Length*
1	0
2	24
3	66
4	120

Table 1: The queue lengths needed to achieve rejection rates of around 5% for the experiments summarized in Figures 2, 3, 6 and 7.

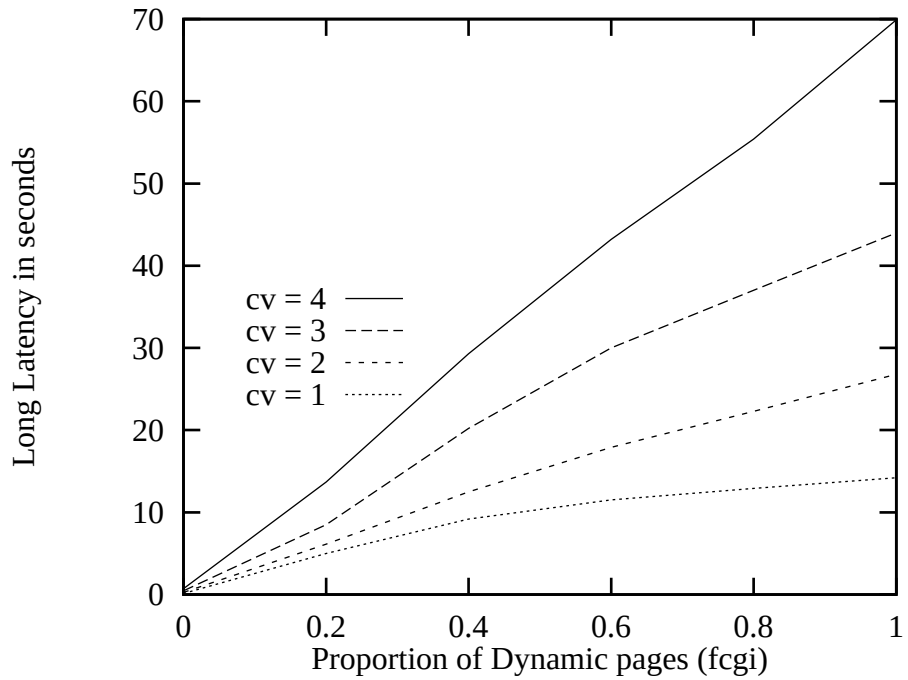

Figure 3: This graph corresponds to the same experiments which resulted in Figure 2. Ninety percent of all requests were satisfied in the times plotted on this graph.

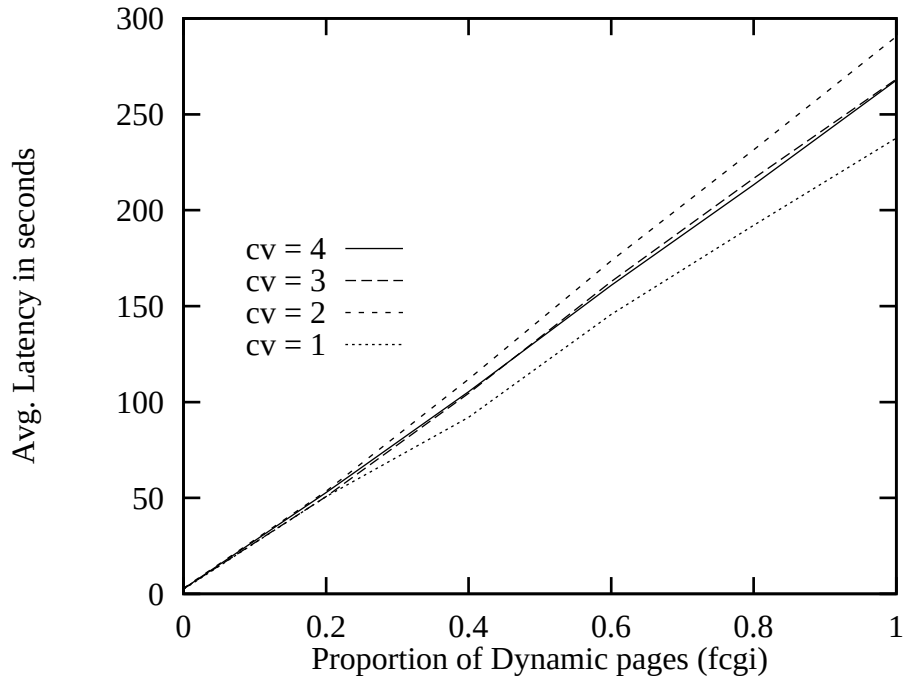

Figure 4: Average latency as a function of the proportion of dynamic pages (f_{cgi}). Each curve represents a request distribution with a different coefficient of variation (c_v) for the distribution of interarrival times. Request rejection rates were kept below 1% by using long queues (1000) for pending requests.

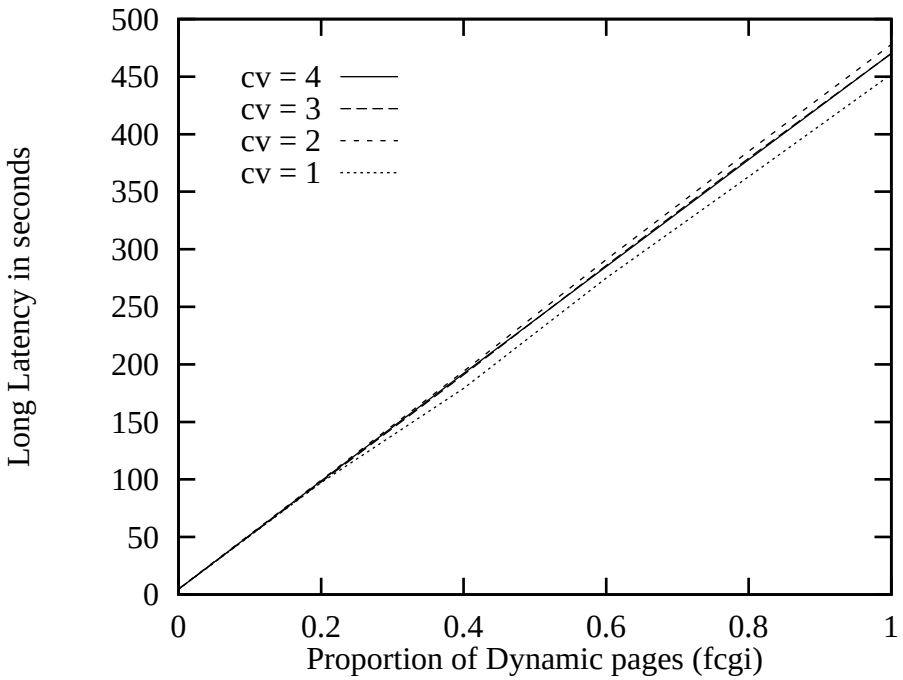

Figure 5: This graph corresponds to the same experiments which resulted in Figure 4. Ninety percent of all requests were satisfied in the times plotted on this graph.

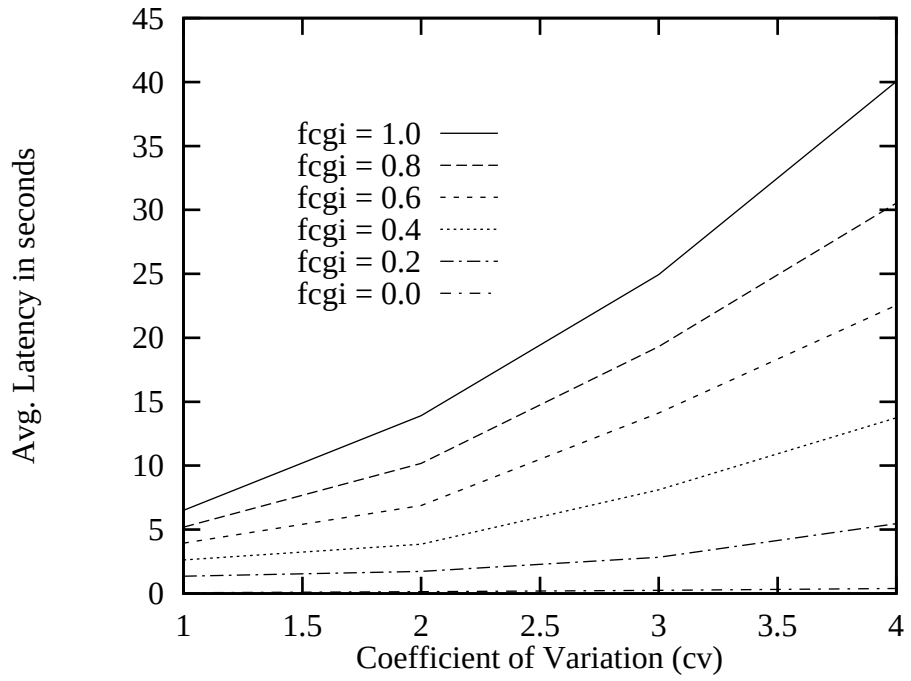

Figure 6: Average latency as a function of the coefficient of variation (c_v) of the request distribution interarrival times. Each curve represents a distribution with a different proportion of dynamic pages (f_{cgi}). Request rejection rates were around 5%. The queue lengths for achieving these rejection rates are shown in Table 1.

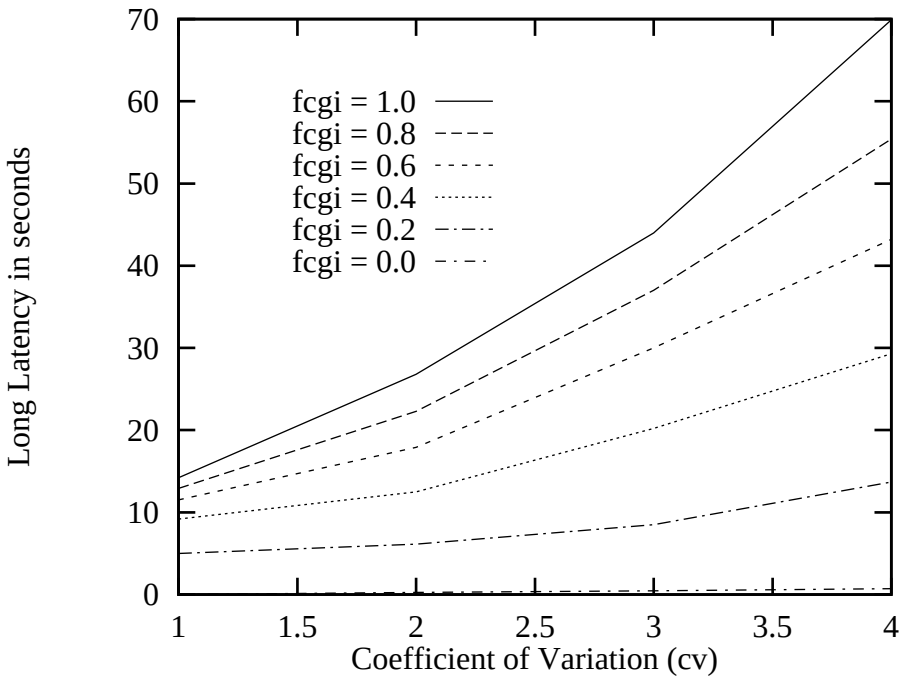

Figure 7: This graph corresponds to the same experiments which resulted in Figure 6. Ninety percent of all requests were satisfied in the times plotted on this graph.

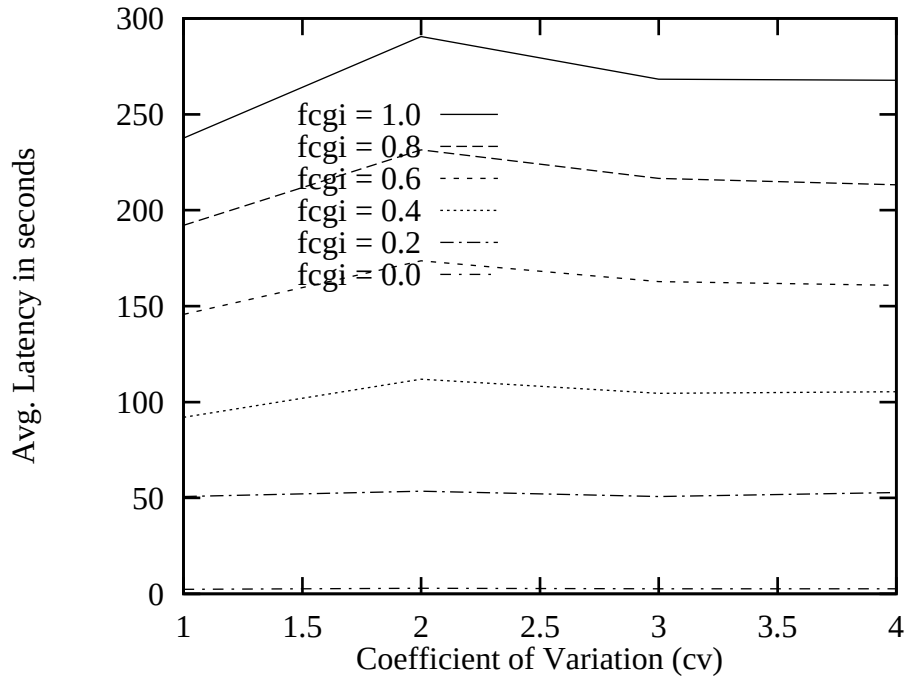

Figure 8: Average latency as a function of the coefficient of variation (c_v) of the request distribution interarrival times. Each curve represents a distribution with a different proportion of dynamic pages (f_{cgi}). Request rejection rates were kept below 1% by using long queues (1000) for pending requests.

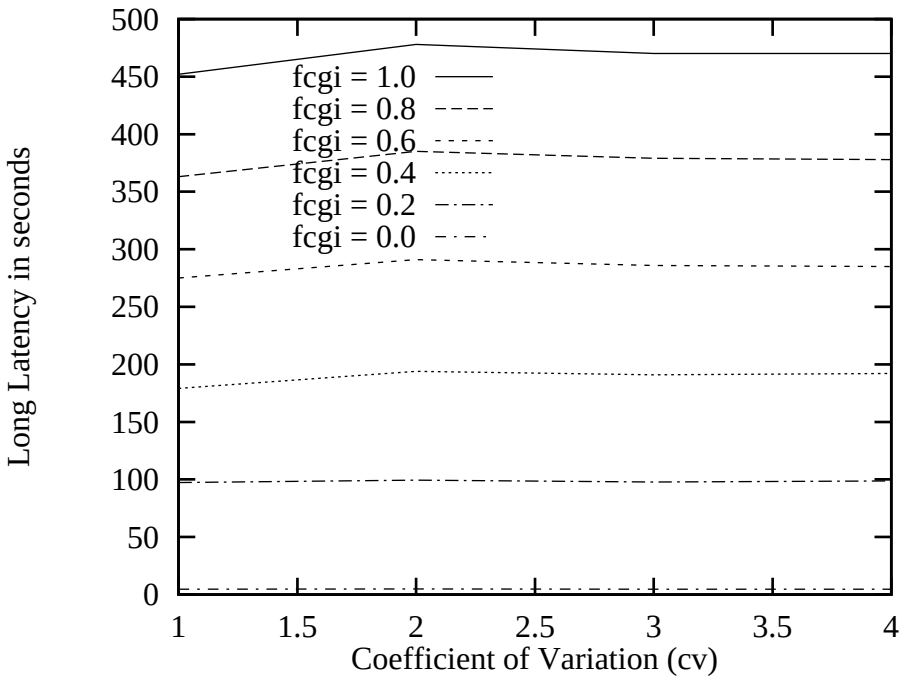

Figure 9: This graph corresponds to the same experiments which resulted in Figure 8. Ninety percent of all requests were satisfied in the times plotted on this graph.

- It is often possible to cache frequently accessed dynamic pages so that they don't have to be regenerated by a server program every time they are requested [3]. This can substantially reduce the number of programs which are invoked by the Web server.
- Server programs which are compute-intensive should be compiled instead of interpreted.
- Many server applications access databases. A straightforward implementation of such an application using a database such as IBM's DB2 would require each invocation of the application to make a new database connection in order to access a database. Connecting to databases is expensive. A more efficient approach is to have long-running processes which keep open connections to databases. Programs which need to access a database do so by communicating with a process holding an open connection to the database. This way, new connections don't have to be established with the database for each access [4].

Figures 2-9 also show how performance is affected by the burstiness in request distributions. When variable length queues are used to maintain rejection rates around 5% as in Figures 2, 3, 6 and 7, burstier distributions result in higher average latencies. This is because bursty distributions require longer queues to sustain the same rejection rates as less bursty distributions. When queue lengths are fixed at 1000 as in Figure 4, 5, 8, and 9, the burstiness of the distribution affects the average latency much less. This queue length was sufficient to achieve rejection rates of less than 1% for all of our test cases. The downside to using this queue length is that average latencies tend to be high.

Figures 4, 5, 8, and 9 also illustrate that performance is very bad when a Web server tries to sustain performance within 1% of the peak number of connections per second. Average latencies are very high. A better strategy is to reject requests so that on average only 95% of the peak number of connections per unit time are being satisfied as in Figures 2, 3, 6 and 7.

The trade-off between rejection rates and average latencies when the normalized request rate r is 1 is shown in Figures 10 and 11. Rejection rates rates are lowered by using longer queues which adversely affect the average latency. The same trade-off for different normalized request rates is shown in Figures 12 and 13. Performance is generally not a problem when r is below .9.

4 Summary and Conclusion

This paper has examined Web server performance in situations where server processing power is the limiting resource. We found that performance is severely degraded by dynamic pages. In order to optimize performance, the percentage of dynamic pages should be kept as low as possible. In situations where dynamic pages are essential, performance can be improved by using server API's such as ICAPI, NSAPI, ISAPI, or FastCGI

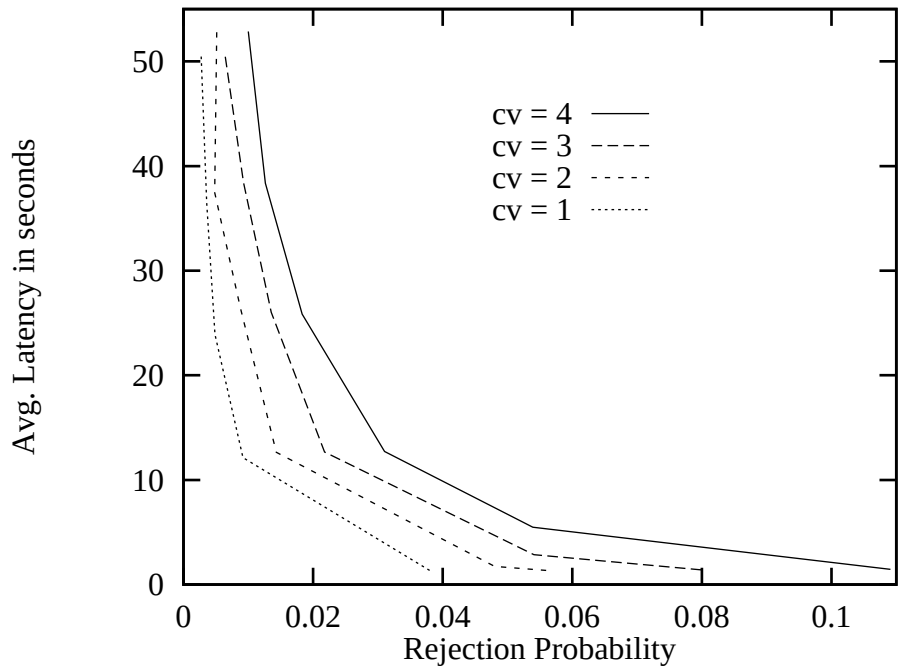

Figure 10: Average latency as a function of the probability of a request being rejected. Request rejection probabilities were varied by varying the maximum length of the queue of pending requests from 0 to 1000. The proportion of requests for dynamic pages (f_{cgi}) was .2. Each curve represents a request distribution with a different coefficient of variation (c_v) for the distribution of interarrival times.

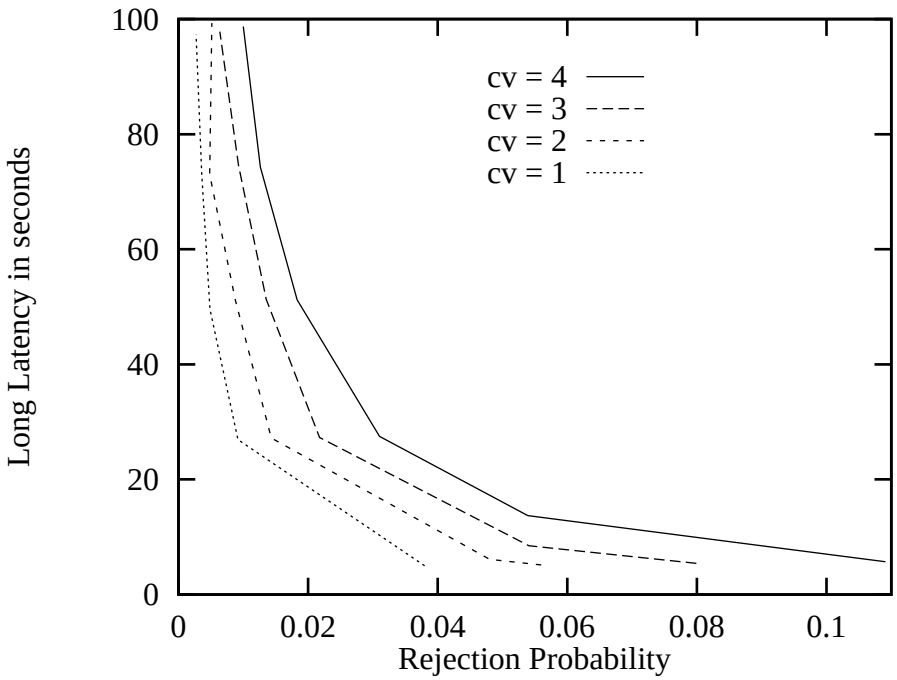

Figure 11: This graph corresponds to the same experiments which resulted in Figure 10. Ninety percent of all requests were satisfied in the times plotted on this graph.

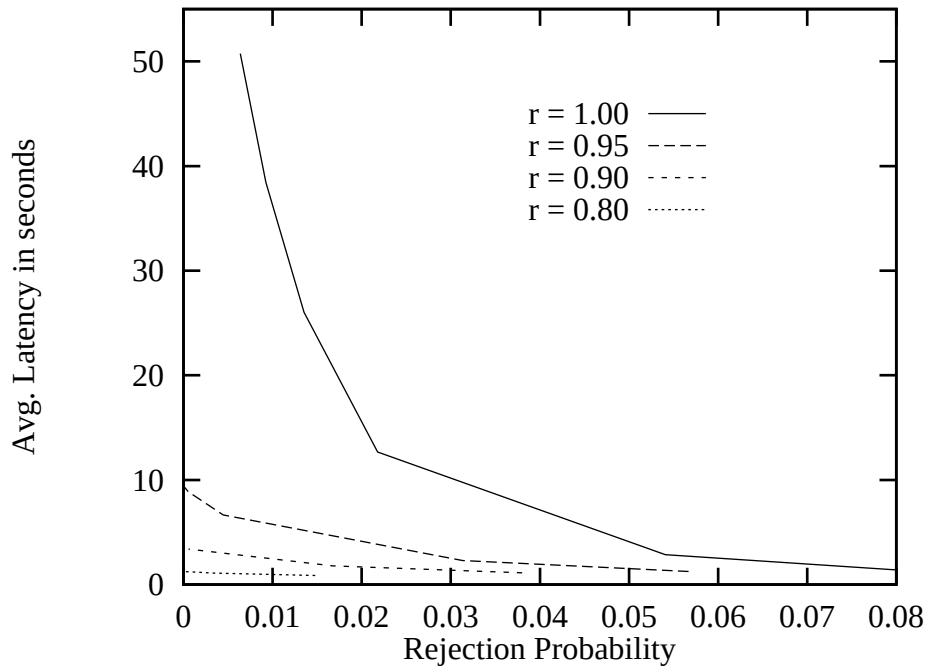

Figure 12: Average latency as a function of the probability of a request being rejected. Different curves represent different normalized request rates (r.) Request rejection probabilities were varied by varying the maximum length of the queue of pending requests from 0 to 1000. The proportion of requests for dynamic pages (f_{cgi}) was .2. The coefficient of variation (c_v) for the request distribution interarrival times was 3.

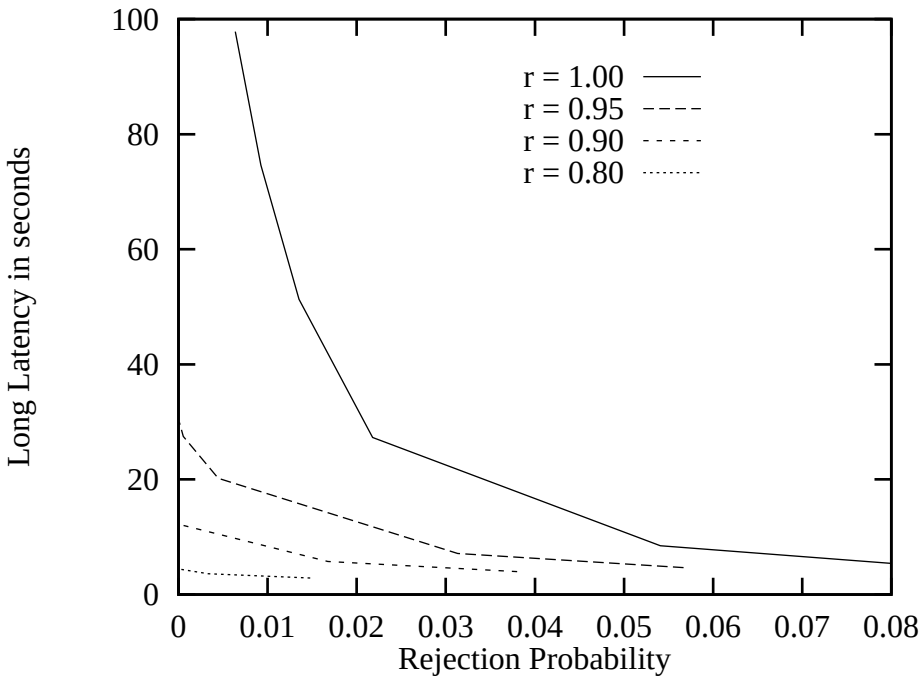

Figure 13: This graph corresponds to the same experiments which resulted in Figure 12. Ninety percent of all requests were satisfied in the times plotted on this graph.

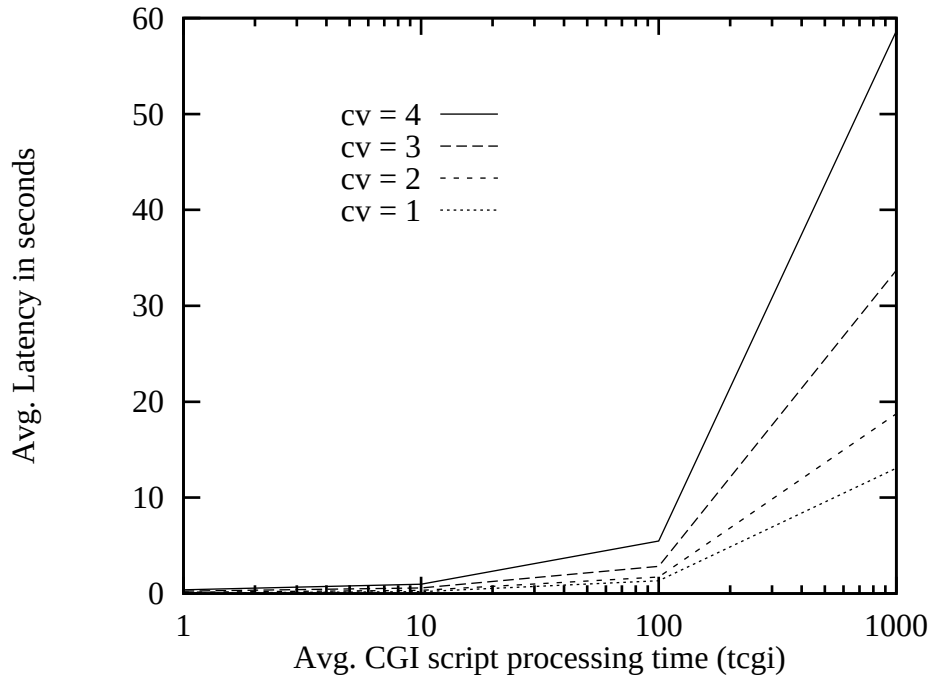

Figure 14: Average latency as a function of the relative CPU time for building each dynamic page (t_{cgi}). The proportion of requests for dynamic pages (f_{cgi}) was .2. Each curve represents a request distribution with a different coefficient of variation (c_v) for the distribution of interarrival times. Request rejection rates were around 5%.

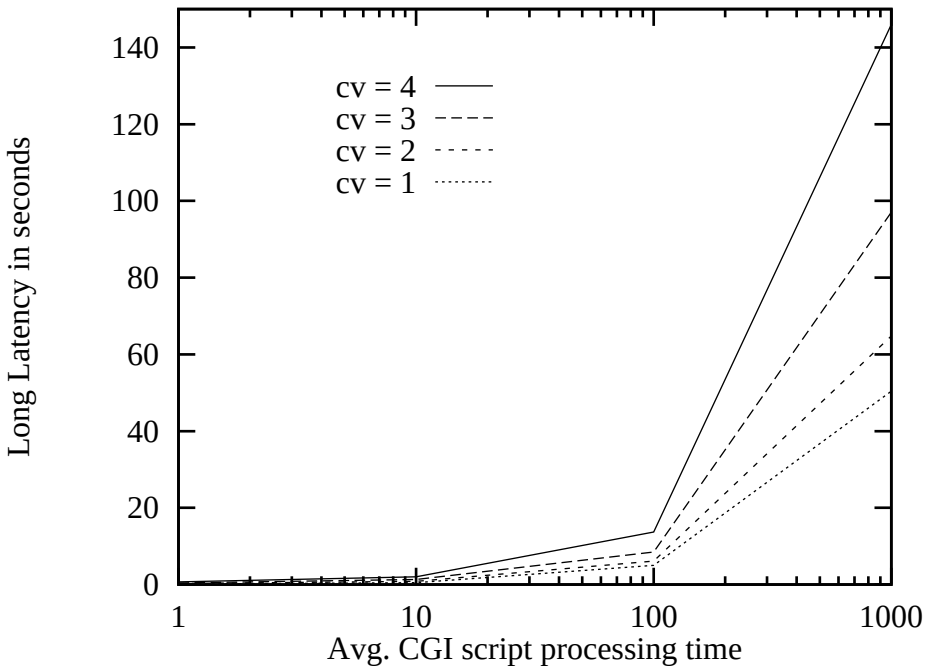

Figure 15: This graph corresponds to the same experiments which resulted in Figure 14. Ninety percent of all requests were satisfied in the times plotted on this graph.

instead of CGI. Caching frequently accessed dynamic pages can also substantially improve performance.

When server load approaches 100%, there is a trade-off between average latencies and the percentage of requests rejected by the server. Average latencies can be reduced by rejecting higher percentages of requests. Due to burstiness in request distributions, servers should not attempt to sustain the maximum connection rate as measured by benchmarks such as WebStone or SPECweb96. Attempts to drive servers to near 100% utilization result in high latencies for satisfying requests. A good strategy for keeping latencies low is for the server to attempt to satisfy 95% or less of the maximum number of requests which can be satisfied per unit time. This can be achieved by maintaining appropriate queue lengths for pending Web requests and rejecting requests as soon as the queue is full.

We studied the characteristics of several request distributions from real Web sites. We characterized the burstiness of the distributions by the coefficient of variation c_v of the request interarrival times during periods when the number of requests per minute was roughly constant. Values for c_v generally range from 1 to 4. Burstier distributions adversely affect performance. Requests for static files usually constitute well over 50% of all request distributions even for Web sites which generate all or most pages dynamically. This is because dynamic pages often include static files containing image or other multimedia data.

References

[1] D. Dias, W. Kish, R. Mukherjee, and R. Tewari. A Scalable and Highly Available Web Server. In *Proceedings of the 1996 IEEE Computer Conference (COMPCON)*, February 1996.

[2] T. T. Kwan, R. E. McGrath, and D. A. Reed. NCSA's World Wide Web Server: Design and Performance. *IEEE Computer*, 28(11):68–74, November 1995.

[3] Y. H. Liu, P. Dantzig, C. E. Wu, J. Challenger, and L. M. Ni. A Distributed Web Server and its Performance Analysis on Multiple Platforms. In *Proceedings of the International Conference for Distributed Computing Systems*, May 1996.

[4] Y. H. Liu, P. Dantzig, C. E. Wu, and L. M. Ni. A Distributed Connection Manager Interface for Web Services on SP Systems. In *Proceedings of the International Conference for Parallel and Distributed Systems*, June 1996.

[5] Microsoft Corporation. (ISAPI) Overview. http://www.microsoft.com/msdn/sdk/platforms/doc/sdk/internet/src/isapimrg.htm.

[6] D. Mosedale, W. Foss, and R. McCool. Lessons Learned Administering Netscape's Internet Site. *IEEE Internet Computing*, 1(2):28–35, March/April 1997.

[7] NCSA. The Common Gateway Interface. http://hoohoo.ncsa.uiuc.edu/cgi/.

[8] Netscape Communications Corporation. The Server-Application Function and Netscape Server API. http://www.netscape.com/newsref/std/server_api.html.

[9] Open Market. FastCGI. http://www.fastcgi.com/.

[10] Silicon Graphics, Inc. World Wide Web Server Benchmarking. http://www.sgi.com/Products/WebFORCE/WebStone/.

[11] System Performance Evaluation Cooperative (SPEC). SPECweb96 Benchmark. http://www.specbench.org/osg/web96/.

[12] N. J. Yeager and R. E. McGrath. *Web Server Technology*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.