Hewlett-Packard NetServer LT 6000r Web Server Performance Analysis and Details

By Bruce Weiner
(PDF version, 101 KB)

February 11, 2000

White Paper Contents

Analysis

The following analysis will group tests of comparable features and SPI alternatives together to make it easier to compare them.

Static Tests

The WebBench static test makes requests for HTML pages stored in files without any additional processing. Its results represent an upper limit on the performance of a Web server.

Figures 1 and 2 give the request rate and latency, respectively, for 100% static requests. We will use these static test results to evaluate the SPICE request efficiency of each SPI. Figure 2 shows the latency values both at peak performance (indicated by the marker on each line) and when 60 clients are making requests.

Figure 1: Static Request Rate Performance
(larger numbers are better)

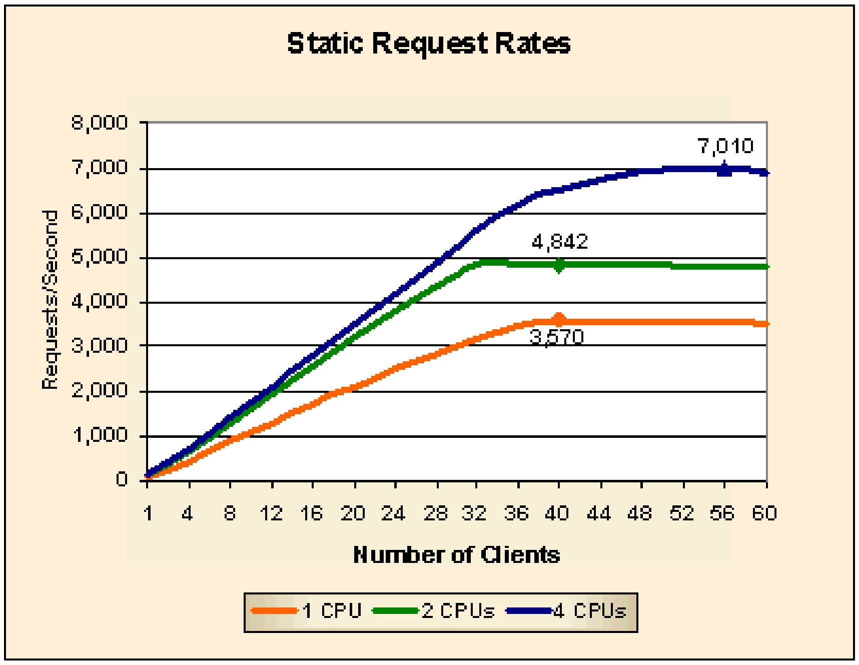

Figure 2: Static Latency Performance
(smaller numbers are better)

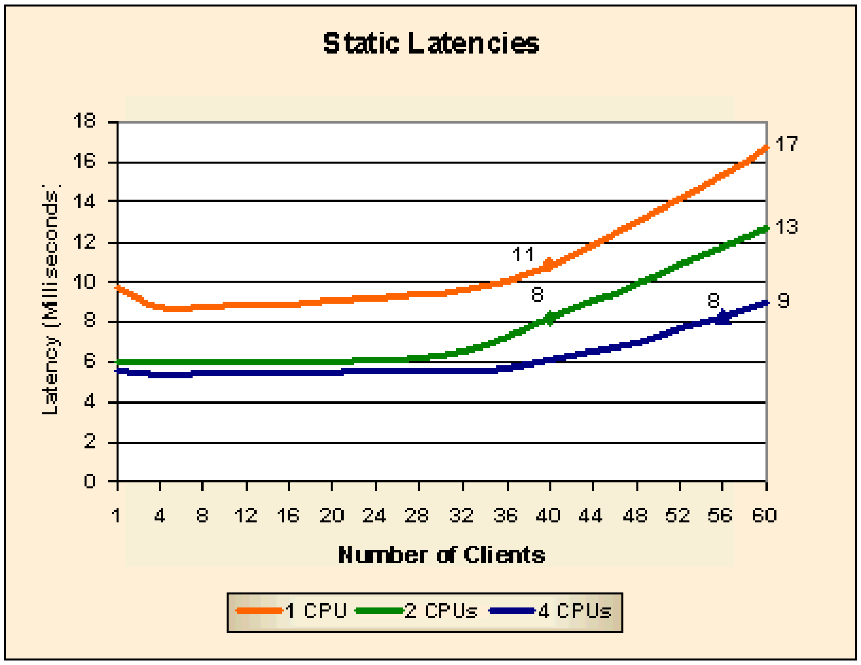

Table 1 shows the scalability of the NetServer LT 6000r for static requests. We were unable to reach peak performance for the 6-CPU configuration because of the performance limitations of the client systems in our test lab at the time of this test. That's why Table 1 shows the 6-CPU configuration performance as Not Available (NA).

Table 1: Static Peak Performance Scaling

Number of CPUs	Peak Request Rate	% Than 1 CPU
1	3,570	-
2	4,842	36%
4	7,010	96%
6	NA	NA

Conclusion

- Web server performance scales well as more processors are added to the NetServer LT 6000r.

100% ASP and 100% ISAPI Tests

We wrote our SPICE ASP test program in VB Script. Our ISAPI module was written in C to make it as fast as possible.

Figures 3 and 4 give the request rate and latency, respectively, for 100% ASP requests. Figure 4 shows the latency values both at peak performance and when 60 clients are making requests.

Figure 3: 100% ASP Request Rate Performance
(larger numbers are better)

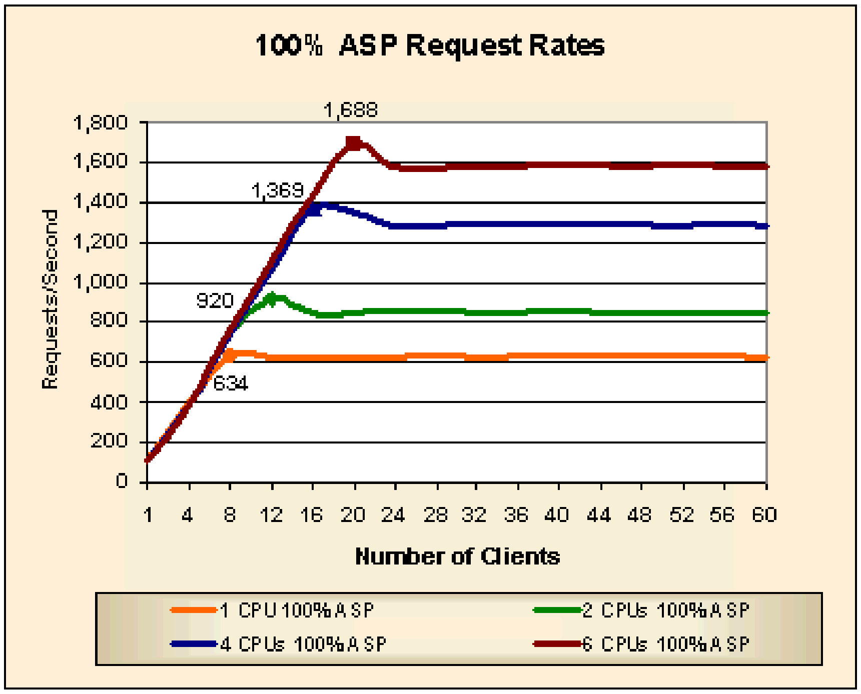

Figure 4: 100% ASP Latency Performance
(smaller numbers are better)

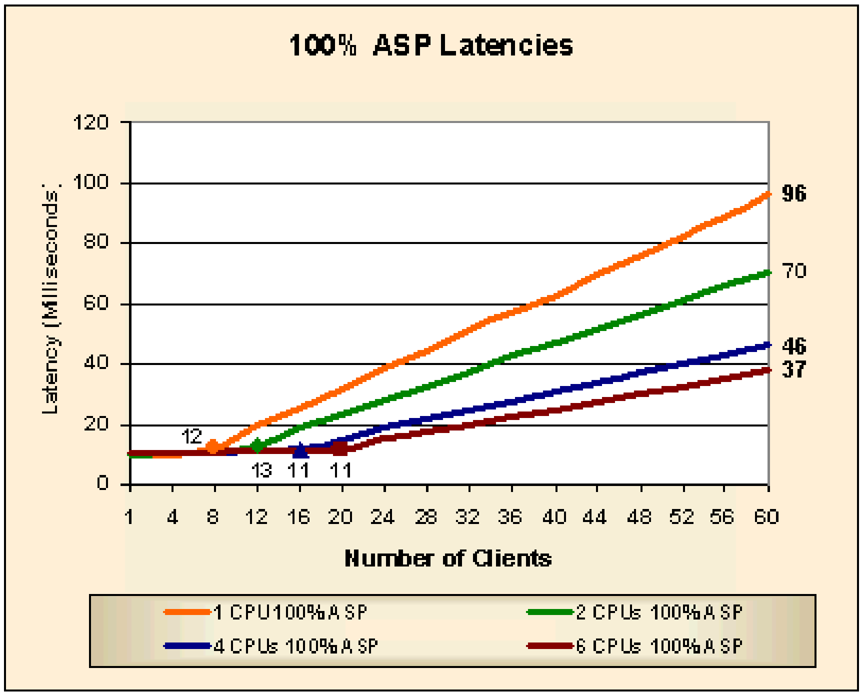

Table 2 shows the scalability of the NetServer LT 6000r for when all of the HTTP requests are satisfied by an ASP program.

Table 2: 100% ASP Peak Performance Scaling

Number of CPUs	Peak Request Rate	% Than 1 CPU
1	634	-
2	920	45%
4	1,369	116%
6	1,688	166%

Figures 5 and 6 give the request rate and latency, respectively, for 100% ISAPI requests.

Figure 6 shows the latency values both at peak performance and when 60 clients are making requests.

Figure 5: 100% ISAPI Request Rate Performance
(larger numbers are better)

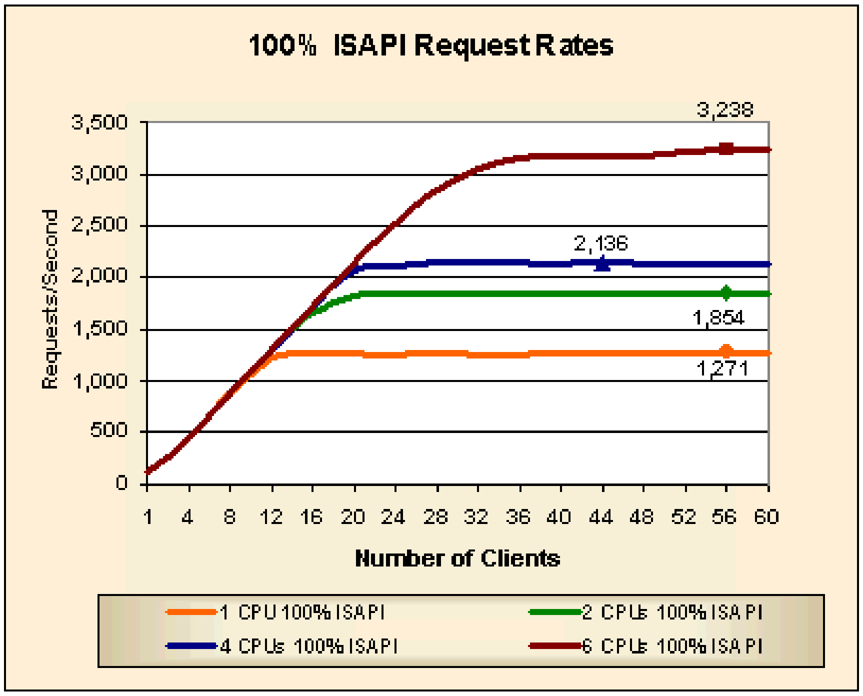

Figure 6: 100% ISAPI Latency Performance
(smaller numbers are better)

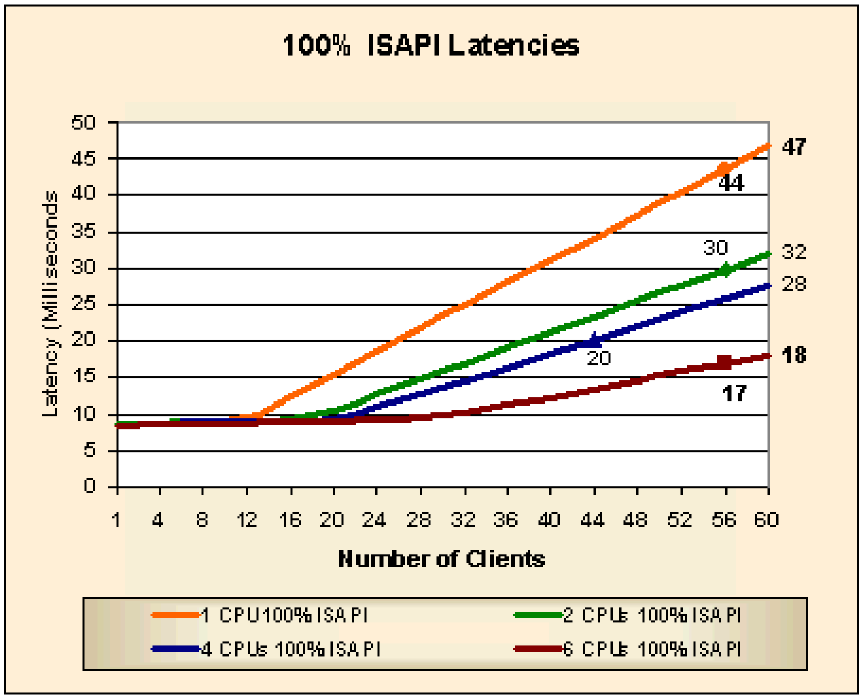

Table 3 shows the scalability of the NetServer LT 6000r for when all of the HTTP requests are satisfied by an ISAPI module.

Table 3: 100% ISAPI Peak Performance Scaling

Number of CPUs	Peak Request Rate	% Than 1 CPU
1	1,271	-

2	1,854	46%
4	2,136	68%
6	3,238	155%

Conclusions

- ASP program performance scales slightly better than ISAPI module performance on the NetServer LT 6000r.
- ISAPI modules run the SPICE benchmark programs almost twice as fast as ASP programs.

E-Commerce Tests

The WebBench e-commerce tests are a mix of static and dynamic requests using both normal and SSL connections. We substituted our ASP program and C ISAPI module for the standard WebBench dynamic programs in the WebBench e-commerce workload so that we could test and compare performance for these programming interfaces.

Figures 7 and 8 give the request rate and latency, respectively, for the e-commerce mix of static and dynamic ASP requests over both SSL and non-SSL connections. Figure 8 shows the latency values both at peak performance and when 60 clients are making requests.

Figure 7: E-Commerce ASP Request Rate Performance
(larger numbers are better)

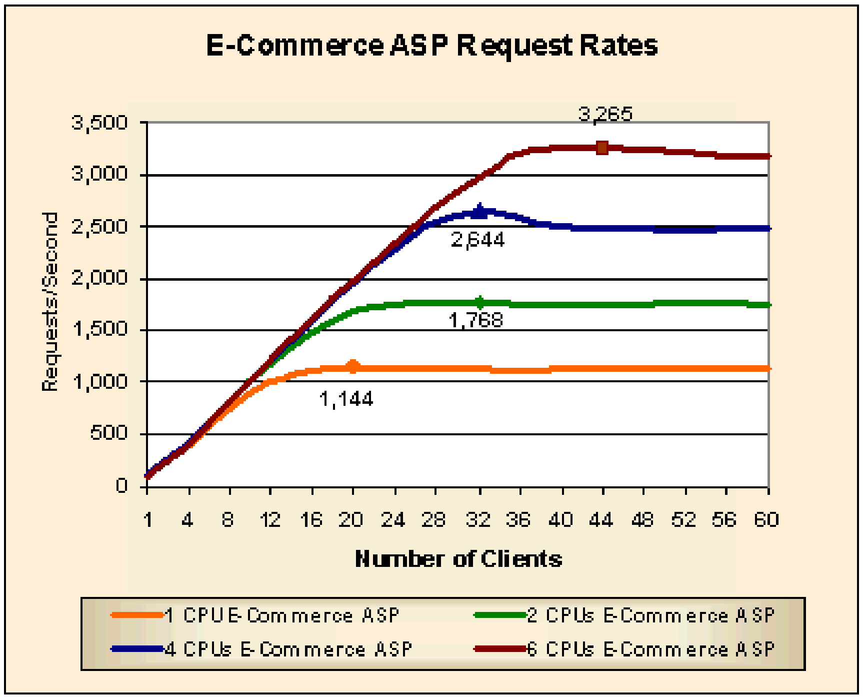

Figure 8: E-Commerce ASP Latency Performance
(smaller numbers are better)

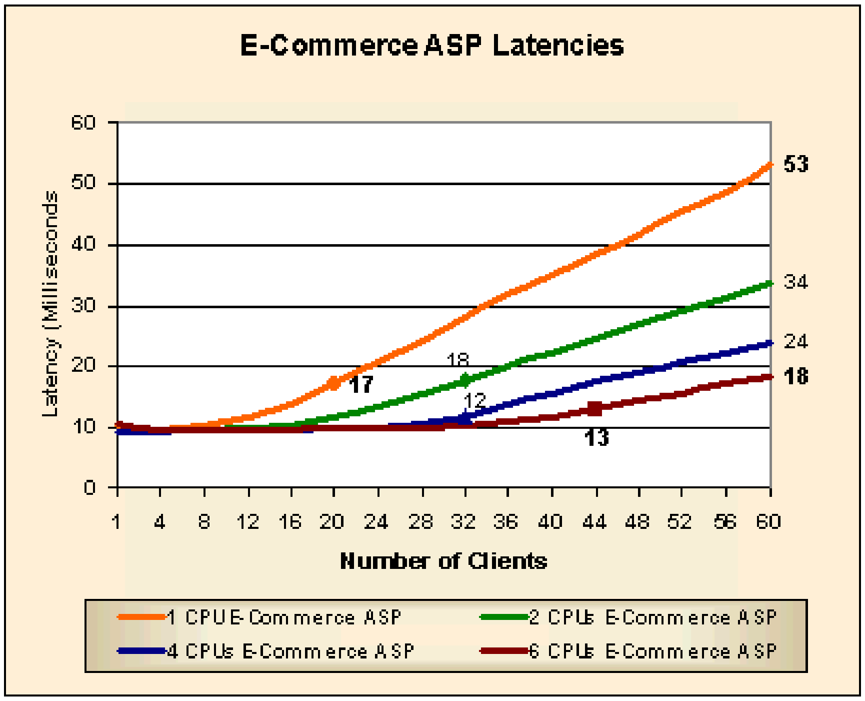

Table 4 shows the scalability of the NetServer LT 6000r for the e-commerce test mix with the dynamic requests satisfied by an ASP program.

Table 4: E-Commerce ASP Peak Performance Scaling

Number of CPUs	Peak Request Rate	% Than 1 CPU
1	1,144	-
2	1,768	55%
4	2,644	131%
6	3,265	185%

You may notice that the e-commerce ASP request rates are almost twice as high as the 100% ASP request rates. This should be expected because the e-commerce test mix makes static, non-SSL requests 75% of the time.

Figures 9 and 10 give the request rate and latency, respectively, for 100% ISAPI requests. Figure 10 shows the latency values both at peak performance and when 60 clients are making requests.

Figure 9: E-Commerce ISAPI Request Rate Performance
(larger numbers are better)

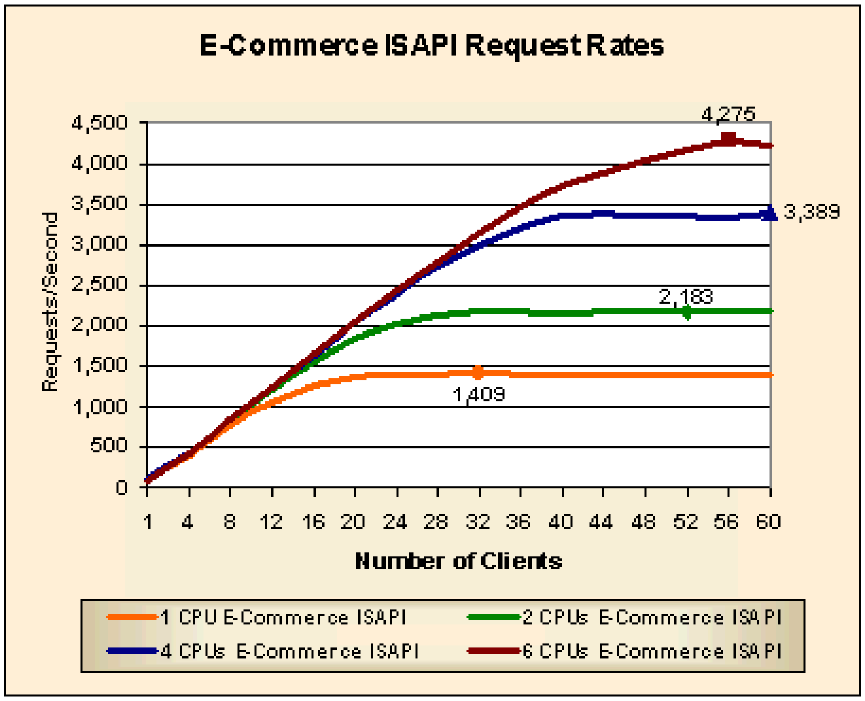

Figure 10: E-Commerce ISAPI Latency Performance
(smaller numbers are better)

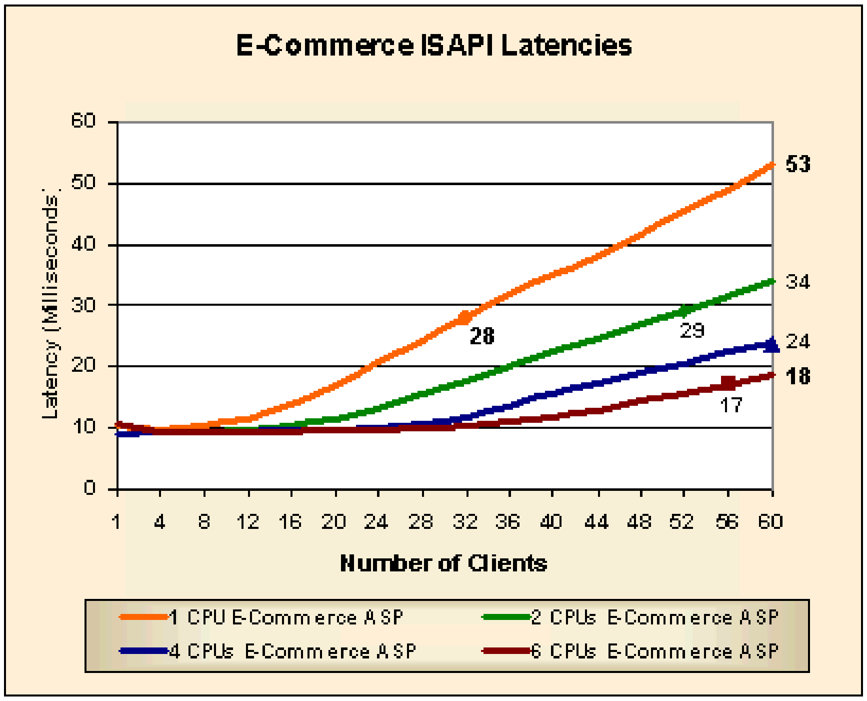

Table 5 shows the scalability of the NetServer LT 6000r for the e-commerce test mix with the dynamic requests satisfied by an ISAPI module.

Table 5: E-Commerce ISAPI Peak Performance Scaling

Number of CPUs	Peak Request Rate	% Than 1 CPU
1	1,409	-
2	2,183	55%
4	3,389	141%
6	4,275	203%

You may notice that the e-commerce ISAPI request rates are higher than the 100% ISAPI request rates. This should be expected because the e-commerce test mix makes static, non-SSL requests 75% of the time.

Conclusion

- SSL processing consumes much of the available CPU power which is why performance scales better for the e-commerce tests than any of the others. This scaling means that the NetServer LT 6000r has the processing power to make real-world, secure Web applications perform well.

Test Methodology

This section of the white paper describes the testing tool we used, WebBench 3.0, and the special test programs we developed to measure the calling efficiency of a server-side programming interface (SPI) that dynamically generates HTML pages (for example, ASPs or ISAPI modules). We use the term server-side programming interface rather than application programming interface (API) to distinguish that the programs we refer to run on the server and not the client.

WebBench 3.0

WebBench tests the performance of a Web server by making HTTP GET requests. WebBench increases the stress on a Web server by increasing the number of client test systems (simply called clients) that request URLs.

The number of clients at which peak Web server performance is measured is a function of the performance of each WebBench client, the Web server's performance, and the requests made. Thus, it will take more slow clients than fast clients to make a Web server reach its peak performance. This means that the shape of a curve that plots Web server performance against the number of clients participating in a test mix is not significant before the peak performance point. However, the curve is significant after the peak performance point because it shows how well a Web server handles an overload.

WebBench can generate a heavy load on a Web server. To do this in a way that makes benchmarking economical, each WebBench client sends an HTTP request to the Web server being tested and waits for the reply. When it comes, the client immediately makes a new HTTP request. This way of generating requests means that a few test systems can simulate the load of hundreds of users. You need to be careful, however, not to correlate the number of WebBench client test systems with the number of simultaneous users that a Web server can support since WebBench does not behave the way users do.

WebBench Metrics

WebBench produces a standard set of reports as a spreadsheet workbook. The Summary Report provides two metrics as a function of the number of clients participating in a test mix: the number of HTTP GET requests/second a Web server can satisfy and the corresponding throughput measured in bytes sent/second.

The peak request rate is a useful measurement for comparing Web servers and SPIs. A larger number of requests/second means that one product can handle a larger load than the alternative. However, the peak request rate is also a function of the average size of a response. For example, a Web server responding only to static requests for files averaging 14 KB might perform at 1,000 requests/second. However, if the average file size requested drops to 500 bytes, that same Web server would respond at a significantly higher rate, perhaps 2,500 requests/second. So when we compare request rates we must be careful to look at the average size of the responses. This was a motivating factor for Mindcraft developing the SPICE Benchmark specification.

The throughput measurements in the WebBench Summary Report are useful for determining the average response size at each request rate and for seeing what the peak number of bytes sent across the network is. Because throughput is equal to the number of requests/second times the average response size in bytes, you can easily find the average response size given the request rate.

WebBench also provides a detailed Client Data Report showing a great deal of information for each client in each test mix. The average latency across all clients in a test mix is included in this report. *Latency* is defined as the time from starting a connection to a Web server to send a request until the last byte of the response is received. Latency is especially interesting to look at because it will indicate how long a user will have to wait to have a request satisfied.

WebBench Test Scenarios and Workloads

WebBench comes with several standard tests. Each test is divided into two parts: a workload and a test scenario. Except for the standard static test, Mindcraft either modified a WebBench test or created a new one in order to do the tests for this white paper. The following is a description of each test we ran:

- **100% Static Test**
  We used the standard WebBench static test. Its workload (the set of URLs to request and their access frequencies) uses 6,000 files totaling 60 MB. The average response size is approximately 6,250 bytes. This is a dynamic average based on the access frequencies for the workload's files, which range in size from 223 bytes to 529 KB.
- **100% ASP**
  We created this test to measure the performance of ASPs. All of the requests are made to the same ASP, which did four writes to return a total of 6,250 bytes of HTML.
- **100% ISAPI**
  We created this test to measure the performance of ISAPI modules written in C. All of the requests are made to the same ISAPI module, which sent a total of 6,250 bytes of HTML in response to each request.
- **E-Commerce Mix using ASPs**
  We started with the standard WebBench e-commerce test and modified the workload to use our ASP instead of the standard WebBench dynamic program. Otherwise, we used the standard WebBench test. This e-commerce workload includes 2% dynamic requests over SSL, 6% static SSL requests, and 17% dynamic requests over a normal connection.
- **E-Commerce Mix using an ISAPI module**
  This e-commerce test is identical to the one using an ASP except that we replaced it with our C ISAPI module.

We used HTTP 1.0 without keepalives for all of our tests, just like the standard WebBench 3.0 tests.

SPICE Tests

The programs we developed test Server-side Programming Interface Call Efficiency, or simply SPICE. We developed the concept of SPICE tests:

- To provide an application-independent way to compare both the dynamic-only and the mixed-static-dynamic performance of Web servers using various SPIs and
- To determine the overhead associated with using a particular SPI.

SPICE tests use applications that do the minimum processing needed to return the same average number of bytes as a static test will return. This means that SPICE tests can be used to make fair comparisons between static-only, dynamic-only, and mixed static-dynamic test scenarios. Because SPICE test programs return the same amount of data as the average static request, the response rate for the SPICE programs is not artificially inflated over the static data response rate.

A SPICE test essentially measures the minimum overhead for using a particular SPI. A common alternative to SPICE tests is to simulate a real-world application. One obvious problem with this alternative is that the simulated application will almost certainly behave differently than the one you want to deploy. While that is also true with SPICE programs, they have an advantage over simulated applications because they are simpler to implement, smaller and use fewer resources. This simplicity and size advantage lets you use SPICE programs to evaluate the true overhead of using a SPI as well as making it much easier to develop and run a test. Of course, if you have your own real application available, you are much better off using it to make your comparisons than a SPICE test.

SPICE Metrics

You can use almost any Web server performance measurement tool to do SPICE tests as long as it supports executing a program that does the minimum processing necessary to return the same number of bytes as a static test. Of course, if your tool returns a different average number of bytes than WebBench, you will have to modify your SPICE programs to return the correct number of bytes.

For SPICE tests, your tool needs to make two types measurements: the number of requests/second the Web server delivers and the average latency for requests. We call these the *SPICE request rate* and the *SPICE latency*, respectively.

The SPICE request rate is useful for comparing Web server performance under load. It incorporates the performance of the server hardware, the server operating system, the Web server software, and the SPI. It is based on aggregating the performance of all of the requests from all of the client systems used in a test. Comparing Web server performance based on SPICE request rate results in a server-centric comparison.

Evaluating Web server and SPI performance solely based on the SPICE request rate can be misleading. The SPICE request rate will always be the upper bound of your server's peak performance, unless your application does almost nothing. You can improve the validity of your evaluation by incorporating SPICE latency.

Using SPICE latency to evaluate Web server and SPI performance helps you make decisions from a user perspective. You can think of SPICE latency as answering the question, "How long will I have to wait to get a response if I try to use a server with the current load on it?" The answer to this question will give you an idea of how responsive a user will find your Web application.

By looking at the SPICE latency on each client system, you can see if a Web server is handling client requests unequally and if the clients (your future users) will experience an unacceptably long wait for a response. If either of these undesirable conditions is true, the *useful peak performance* of the Web server is lower than the peak performance you are measuring. You can determine the useful peak performance by finding the maximum performance point at which the Web server treats client requests equally and at which it has an acceptable SPICE latency.

SPICE latency also is useful to estimate the response time a user will experience from a lightly loaded server. Simply add the SPICE latency to the time it takes your application to do its job and your estimate is done. Unfortunately, this simple addition will not be accurate for a heavily loaded server because your real application will take up CPU time, memory and other resources thereby increasing the actual latency.

Finally, the SPICE latency curve will show you how responsive you server will be as the load increases.

We define SPICE request efficiency as the ratio of the SPICE request rate divided by the static request rate. In other words:

SPICE request efficiency =
 SPICE request rate/static request rate

SPICE request efficiency is a measure of how much dynamic request performance using a particular SPI will degrade from that of static-only requests.

Configuration Details

HP NetServer LT 6000r

We used the same NetServer LT 6000r for all of the test reported here. We used the directive *numproc=* in the *boot.ini* file to specify at boot time how many CPUs to use . Table 6 shows the system configuration.

Table 6: NetServer LT 6000r Configuration

Feature	Configuration
CPU	6 x 550 MHz Pentium III Xeon Cache: 16KBI+16KBD on chip per processor and 2 MB L2 cache per processor
RAM	4 GB
Disks	**Four disks**: 18.2 GB HP B80-DC47, 10,000 RPM Embedded NetRAID controller with 64 MB cache **2 logical drives**: OS and paging on a 17,365 MB logical drive; Web Data and Web Logs on a 52,095 MB logical drive
Networks	2 x Aleton ACEnic Gigabit Ethernet Network Interface Cards (SX fiber)

Windows 2000 and IIS 5

We made the following Windows 2000 configuration and tuning changes:

1. We set the following registry parameters in HKLM\System\CurrentControlSet\Services\:
   - NDIS\Parameters\ProcessorAffinityMask = Dword 0
   - InetInfo\Parameters\ObjectCacheTTL = Dword 10,000
   - InetInfo\Parameters\MaxCachedFileSize = Dword 0x100000
   - InetInfo\Parameters\PoolThreadLimit = Dword [the number of active CPUs in the system]
   - InetInfo\Parameters\MaxPoolThreads = Dword 1
2. We set the following registry parameters in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ Control\Class\{4D36E972-E325-11CE-BFC1-08002BE10318} which holds the Alteon ACEnic parameters under the keys 0000 and 0001:
   - SendCoalMax = Dword 80
3. We set IIS to handle over 100,000 hits per day.

We used the Microsoft Visual C/C++ compiler, version 6 for compiling the C ISAPI modules.

Test Lab

Mindcraft's test lab consists of three types of systems for a total of 60 clients:

. 12 Type A clients configured as specified in Table 7.
B. 12 Type B clients configured as specified in Table 8.
C. 36 Type C clients configured as specified in Table 9.

All of the clients were connected to an HP ProCurve 4000M switch via a full-duplex 100Base-TX link. The ProCurve 4000M has two full-duplex Gigabit Ethernet links, which were used to connect the server. We used an independent system to be the WebBench controller. This controller was also connected to the ProCurve 4000M switch.

Table 7: Type A Client Configuration

Type A Clients		
System	**CPU**	200 MHz Pentium® Pro; Intel VS440FX motherboard
	Cache	L1: 16 KB (8KB I + 8KB D) L2: 256 KB
	RAM	64 MB EDO
	Disk	2GB EIDE
Operating System	Windows NT Server 4.0, Service Pack 5 installed	
Network	1 x 100Base-TX 3Com 3C905-TX Network Interface Card	

Table 8: Type B Client Configuration

Type B Clients		
System	**CPU**	266 MHz Pentium® II; Intel AL440LX motherboard
	Cache	L1: 16 KB (8KB I + 8KB D) L2: 256 KB
	RAM	64 MB EDO
	Disk	2GB EIDE
Operating System	Windows NT Server 4.0, Service Pack 5 installed	
Network	1 x 100Base-TX Intel EtherExpress Pro/100+ LAN Adapter	

Table 9: Type C Client Configuration

Type C Clients		
System	**CPU**	466 MHz Celeron®; Abit BM6 motherboard
	Cache	L2: 128 KB
	RAM	128 MB Kingston ECC SDRAM, 100 MHz
	Disk	4GB ATA/66
Operating System	Windows NT Server 4.0, Service Pack 5 installed	
Network	1 x 100Base-TX Intel EtherExpress Pro/100+ Management Adapter	

of their respective companies.