Performance Comparison of PHP and JSP as Server-Side Scripting Languages

Scott Trent, Michiaki Tatsubori, Toyotaro Suzumura, Akihiko Tozawa, and Tamiya Onodera

IBM Tokyo Research Laboratory
16-23-14 Shimotsuruma Yamato-shi, Japan 242-8502
{trent,mich,toyo,atozawa,tonodera}@jp.ibm.com

Abstract. The dynamic scripting language PHP has become enormously popular for implementing lightweight web applications, and is widely used as a server-side scripting language for web servers. To contrast the performance of PHP and JSP for this purpose, we used the SPECweb2005 benchmark, which provides three application scenarios implemented in both PHP and JSP. This paper describes and contrasts the results of SPECweb2005 performance benchmark testing performed on different configurations of PHP and JSP using the popular web servers Apache and Lighttpd. Despite the execution overhead of interpretation in PHP engines observed in micro benchmarks, the experimental result of SPECweb2005 benchmark yields valuable performance data for web server implementers. The efficiency of scripting language runtimes still matters for the end-to-end performance. However, once carefully architected and tuned, the language runtime is less of a bottleneck than the web server performance itself.

Keywords: PHP, JSP, SPECweb, Benchmarking, Web Server.

1 Introduction

The dynamic scripting language PHP (PHP Hypertext Preprocessor) has become enormously popular for implementing lightweight web applications, and is widely used to access databases and other middleware. Apache module popularity surveys performed by Security Space in October 2007 indicate that 37% of Apache servers have PHP support enabled [11], making it the most popular Apache module by 10 percentage points. Businesses are quickly realizing the powerful combination of a service oriented architecture environment with dynamic scripting languages like PHP [5]. However, we believe that there are still critical performance issues involving PHP which remain to be investigated.

This paper focuses on the use of dynamic scripting languages to implement web server front-end interfaces. This corresponds with the way that the industry standard web server performance benchmark SPECweb2005 utilizes PHP and JSP (JavaServer Pages). In this case, scripts are used for the implementation of dynamic page generation, rather than the realization of complex business logic. This contrasts with

V. Issarny and R. Schantz (Eds.): Middleware 2008, LNCS 5346, pp. 164–182, 2008.

the traditional uses of complex JSP-based business logic implementation. While there are numerous studies on dynamic web content, this paper complements these studies with detailed analysis focusing on PHP. For example, following the performance study on CGI (Common Gateway Interface) based web servers for dynamic content by Yeager & McGrath back in 1995, researchers and practitioners have been examining the performance of more recent dynamic Web content generation technologies [3, 13, 15, 17]. These works, however, handle application scenarios where servlet front-ends implement relatively complex business logic.

Although Warner and Worley discuss the importance of also using PHP with SPECweb2005 [18], to the best of the author's knowledge, this paper is the first to publish a detailed analysis of SPECweb2005 experimental results using both PHP and JSP. The detailed analysis of PHP and JSP performance based on SPECweb2005 offered by this paper enables designers and implementers of web servers to understand the relative performance and throughput of different versions and configurations of PHP and JSP.

The rest of this paper is organized as follows. Section 2 discusses multi-tier web server architecture and the lightweight front-end approach using PHP and JSP. Section 3 reports on our findings regarding PHP and JSP language runtime micro benchmark performance. Section 4 details our SPECweb2005 benchmark methodology, environment, and test configurations. Section 5 analyzes SPECweb2005 benchmark throughput results, CPU usage profiling, and related performance metrics. Section 6 discusses the importance of these results. Section 7 covers related work, followed by our conclusions in Section 8.

2 Multi-tier Web Server Architecture: Lightweight Front-End Using PHP/JSP

Developers typically use PHP to implement a front-end interface to dynamic Web content generators, which are combined with web server software and back-end servers to provide dynamic content. The web server directly handles requests for static content and forwards requests for dynamic content to the dynamic content generator. The dynamic content generator, supported by back-end servers, executes code which realizes the business logic of a web site and stores dynamic state. Back-end servers may be implemented as a straight-forward database, or may be more complex servers handling the business logic of the web site. The front-end implementation may vary from heavy-weight business logic handlers to lightweight clients composing content received from back-end servers.

This paper focuses on multi-tier web site development scenarios utilizing such lightweight front-ends, supported by one or more layers of heavy-weight back-ends. This assumption is reasonable when considering Service-Oriented environments where PHP scripts are used to implement a "mash-up" of services provided elsewhere, in addition to the case of simple web sites such as bulletin boards where PHP scripts are just a wrapper to a database. Within the scenarios described in this paper, the dynamic content generator provides client implementation in addition to page composition. It connects to the back-end server through a network using either standard protocols such as HTTP or application/middleware-specific protocols.

JSP technology can be considered an alternative to PHP in implementing such front-ends. While it is part of the Java Servlet framework, developers typically use JSP to implement lightweight front-ends. Both PHP and JSP allow developers to write HTML embedded code. In fact, although there are language inherent differences between PHP and Java, the use of PHP scripts and JSP files can be very similar.

The objective of the experiments detailed in this paper is to measure the performance of lightweight front-end dynamic content generation written in PHP and JSP with popular web servers such as Apache and Lighttpd. This web server architecture scenario involves users who access a web server with pages written in plain static HTML, as well as JSP and PHP scripts which mix scripting language with HTML code. The configuration assumed within the paper is a typical one, where web server software, such as Apache, distinguishes between pure HTML, JSP, and PHP respectively with suffixes such as .html, .jsp, and .php. HTML code is directly returned to the requesting end-user's web browser, where JSP and PHP pages are respectively parsed by the Tomcat script engine and the PHP runtime engine which both provide pure HTML which is forwarded to the end-user on a remote system. (A sample comparison of similar trivial JSP and PHP scripts, along with resulting HTML code can be seen in Table 1) A common point between JSP and PHP is that implementations which perform well have a dynamically compiled and cached byte code. For example, the Java runtime used by the Tomcat script engine which we used performs much better when the Just-in-Time (JIT) compiler is enabled to create efficient cached native runtime code. Similarly, the Zend PHP runtime we used also performs significantly better when the Alternative PHP Cache (APC) is enabled, in which APC stores PHP byte codes compiled from the script source code in shared memory for future reuse.

Table 1. Sample PHP and JSP scripts with resulting HTML code

PHP Script	JSP Script	Resulting HTML Code

`<html> <body>`
`The date is`
`<?php`
`echo`
`date(DATE_RFC822);`
`?>`
`</body> </html>`
	`<html> <body>`
	`The date is`
	`<%=`
	`new`
	`java.util.Date();`
	`%>`
	`<body> </html>`
		`<html> <body>`
		`The date is`
		`Tue, 1 Jan 08`
		`12:00:00`
		`</body> </html>`

3 Language Runtime Performance Micro Benchmarking

To understand the difference in performance characteristics between PHP and Java at the language runtime level, we compared the following engines using a series of micro benchmark tests:

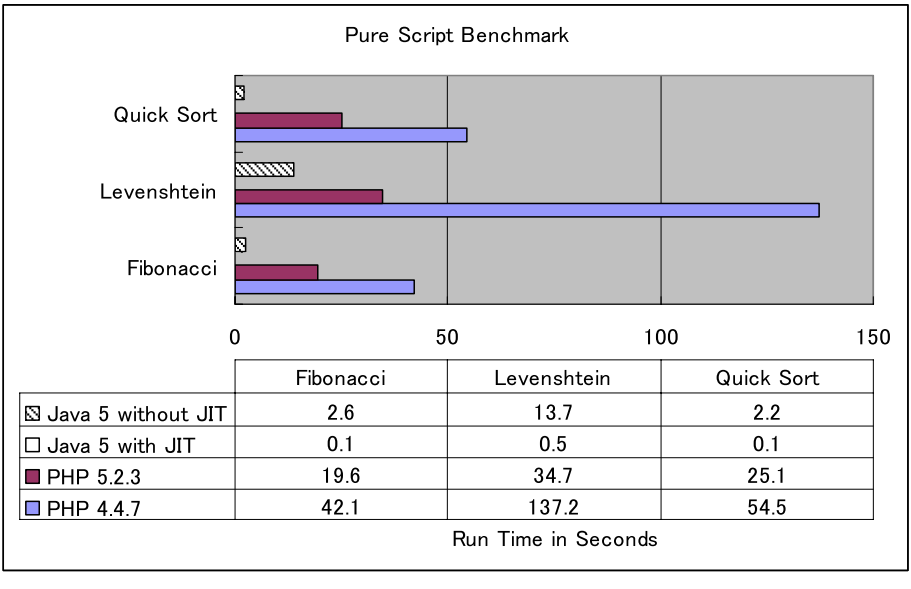
- PHP 4.4.7
- PHP 5.2.3
- Java 5 with Just-In-Time (JIT) compilation (IBM J9 VM 1.5.0 Build 2.3)
- Java 5 without Just-In-Time (JIT) compilation (same as above)

The PHP language framework allows developers to extend the language with library functions written in C. These functions, which are known as "extensions", are then available to be used within PHP scripts. The PHP runtime provides a variety of extensions for string manipulation, file handling, networking, and so forth. Since our first goal was to understand the performance of the PHP runtime itself, we conducted our experiments without the use of extensions. We developed the following micro benchmarks:

- A quick sort benchmark which sorts 100 integers,
- A Levenshtein benchmark which measures the similarity between two strings of 56 characters,
- A Fibonacci benchmark which calculates the 15th value in a Fibonacci series with two arbitrary starting values.

These PHP benchmarks were implemented entirely with PHP language primitives and avoided the use of PHP extensions. The Java versions also focused on using language primitives rather than standard classes. We compared the total run time of executing each test 10,000 times with each engine. We also executed each benchmark an additional 10,000 times as a warm-up, before the measured test. This prevents Java just-in-time compilation overhead from impacting the score in the Java tests. We ran the experiment on an Intel Pentium 4 CPU at 3.40 GHz with 3GB RAM Memory, with the Linux 2.6.17 kernel.

This test demonstrates large performance differences between each of the measured scripting languages and implementations. The experimental results in Figure 1 indicate that "Java 5 with JIT compilation" performs the best, followed by

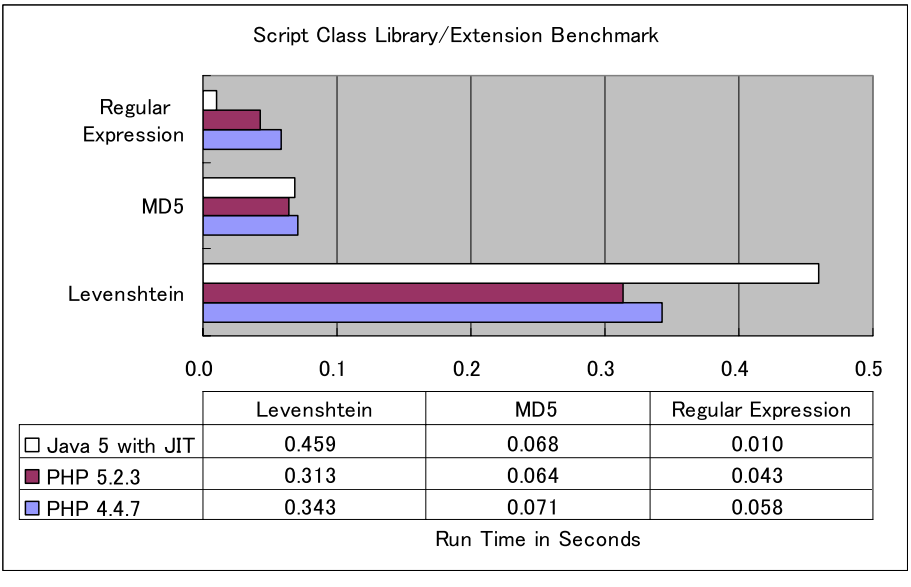


	Fibonacci	Levenshtein	Quick Sort
Java 5 without JIT	2.6	13.7	2.2
Java 5 with JIT	0.1	0.5	0.1
PHP 5.2.3	19.6	34.7	25.1
PHP 4.4.7	42.1	137.2	54.5

Fig. 1. Pure Script Benchmark Performance

"Java 5 without JIT compilation", "PHP 5.2.3", and "PHP 4.4.7" in all measured cases. Java 5 with JIT demonstrated nearly three orders of magnitude better performance due to the use of efficiently generated native code. It is also obvious that PHP 5.2.3 has a two to three times performance improvement over PHP 4.4.7 with the measured computations.

Secondly to determine the performance effect of PHP extensions compared with Java class methods, we developed and tested three additional micro benchmarks: regular expression matching, MD5 encoding, and Levenshtein comparison. For regular expression matching, the Perl Compatible Regular Expression extension (through the `preg_match()` function) was used in PHP, and the java.util.regex package was used in Java. For MD5 encoding, the MD5 extension was used in PHP and java.security.MessageDigest was used in Java. This experiment does not compare exactly the same logic, but rather demonstrates that the use of PHP extensions is competitive with Java using just-in-time compilation, as seen in Figure 2.



	Levenshtein	MD5	Regular Expression
Java 5 with JIT	0.459	0.068	0.010
PHP 5.2.3	0.313	0.064	0.043
PHP 4.4.7	0.343	0.071	0.058

Fig. 2. Script Class Library/Extension Benchmark Performance

Although the pure script experiment showed three orders of magnitude difference between the performance of various implementations of Java and PHP, the use of PHP extensions (written in C) and compiled Java class libraries show much less variation. In the extreme, the regular expression test showed a maximum performance difference of about five times between Java and PHP, on the other end, the MD5 test results were nearly equivalent between Java and PHP. Thus a inherent performance risk of interpreted scripted languages such as PHP can be overcome with the use of efficient library functions such as PHP extensions written in C.

4 PHP/JSP SPECweb2005 Benchmark Methodology

Although micro benchmarks are simple to implement and analyze, and are thus often used in performance analysis, we next used the industry standard SPECweb2005 benchmark to understand the impact of different versions and configurations of PHP and JSP in more realistic situations. The SPECweb2005 benchmark, developed by the Standard Performance Evaluation Corporation (SPEC), is comprised of three test scenarios based on common website usage: a banking site scenario, an e-commerce site scenario, and a support site scenario. The banking site scenario allows for typical encrypted account transactions with Secure Sockets Layer (SSL) libraries where 60% of the data is generated through dynamic web pages. The e-commerce shopping site allows a user to browse catalogs and "purchase" products using both encrypted and unencrypted data. As shown in Table 2, experimentally about 5% of the data in the e-commerce scenario is transmitted using SSL encryption and 70% of the data transmitted is generated through dynamic web pages. Finally, the vendor support site provides downloading of large unencrypted support files such as manuals and software. As this scenario primarily allows for accessing large non-confidential static files, there is no encryption, and only 12% of the data transmitted is generated through dynamic web pages. Since SPECweb2005 is implemented in both PHP and JSP, it is particularly well suited for comparing performance between the two languages. Yet because every single officially published SPECweb2005 benchmark result as of Summer 2008 was performed using JSP rather than PHP [12], this paper provides a unique comparison of both implementations, which is valuable considering the popularity of real world web servers based on PHP.

Table 2. Experimentally measured percentage of encrypted and dynamic data transfered for each SPECweb2005 scenario

	Banking	**Ecommerce**	**Support**
Percentage of encrypted data	100%	4.4%	0%
Percentage of dynamic data e.g., script output	59.5%	71.6%	11.7%

A typical SPECweb2005 test bed has multiple client machines controlled by a Prime Client to provide a load on the System Under Test (SUT) to simulate hundreds to tens of thousands of users accessing the scenario web sites. Although multiple software components can run on the same physical system, a high level of distribution is desirable to provide a realistic environment. For example, an average of 22 physical clients were used in the officially published SPECweb2005 scores [12]. To reflect modern multi-tier web server architecture, SPECweb2005 uses one or more machines to serve as a Back End SIMulator (BESIM), emulating the function of a "Back End" database server.

4.1 SPECweb2005 Benchmark Environment

We used a single System Under Test machine running the web server, a BESIM server running the Back End SIMulation engine, a prime client machine, and three

additional dedicated client machines. The computers were connected via a gigabit Ethernet network. The System Under Test was an IBM IntelliStation M Pro with a 3.4 GHz Xeon uniprocessor running Fedora Core 7 (kernel 2.6.23), Apache 2.2.6, and Lighttpd 1.4.18. Apache Tomcat was used as the JSP servlet container [1]. PHP 5.2.4, and Tomcat 5.5.25 were used in their respective tests. Tomcat was configured to use an IBM implemented Java Virtual Machine: J9 VM 1.5.0 Build 2.3. The standard distribution of SPECweb2005 was installed and configured as described in SPEC documentation [12].

4.2 Testing Methodology

In addition to following the guidelines laid down in the SPECweb2005 documentation [12] we developed a testing tool which could be configured to automatically run multiple tests, iterating such variables as the script engine language (PHP, JSP), the web server (Apache, Lighttpd), the number of simultaneous sessions, and the SPECweb2005 scenario (banking, ecommerce, and support), and other tuning factors. We varied the number of simultaneous sessions from 250 to 3000 by increments of 250. To ensure valid results, the SPECweb2005 test harness will abort individual tests when the web server response threshold is exceeded. We used 3000 simultaneous connections as our maximum because beyond this, with our configuration, it is rare for a test to run successfully to completion. To avoid genetic skewing of data, this paper only displays data for tests that ran successfully without repeated retries. Load levels that may not run to completion are extremely unlikely to result in a suitable Quality of Service (QoS) level to qualify as a valid SPECweb2005 test run.

To assure a fair comparison, before each individual test is initiated, our testing tool restarted the SPECweb2005 client components, all middleware such as Tomcat, and web server, and otherwise ensured that the environment on each system in this distributed environment was in a consistent and receptive state. An officially published SPECweb2005 benchmark score is a single value which based on three 30-minute test runs from each of the three scenarios shows the performance improvement over SPEC's reference machine. This can be used to compare the relative performance of web serving hardware platforms from different vendors. Since our goal was to analyze in detail how the use of different scripting languages and web servers affects performance, we used internal metrics such as the number of good/tolerable/failed requests served as reported from the SPECweb2005 test harness for each test. To improve test coverage in the time available, we used 10-minute test runs rather than the official 30-minute run, and only ran each test once rather than three times. Although our test runs are not suitable for reporting as an official score, they are very useful for identifying trends seen as over tens of tests, and variation seen with identical test runs was small as demonstrated in Figure 3. The `vmstat` command was also used to monitor such performance statistics as memory usage, swapping activity, and CPU utilization [6]. No swapping activity was observed during our reported tests. In separate test runs, we used the oprofile tool to identify process, module, and function CPU utilization.

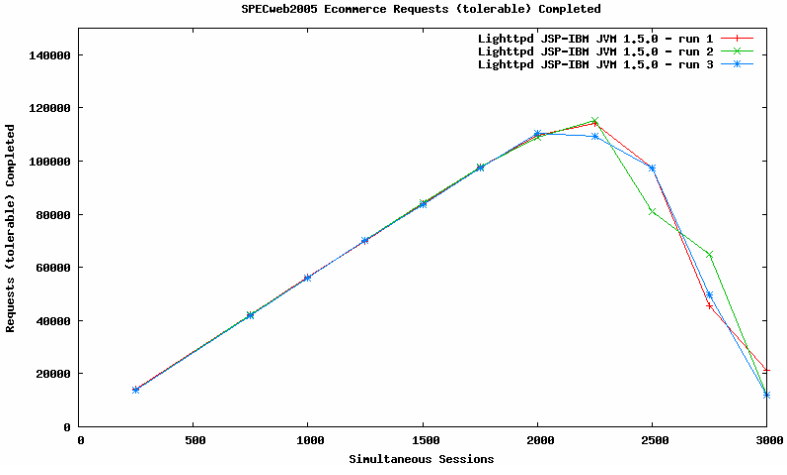


Fig. 3. Repeated test runs demonstrate similar results

We measured each of the SPECweb2005 scenarios with the following five configurations of scripting language and web server with the goal of contrasting JSP with PHP, and Apache with Lighttpd:

- JSP with Apache via mod_jk connector
- JSP with Lighttpd via mod_proxy module
- PHP with Apache via FCGI protocol
- PHP with Lighttpd via FCGI protocol
- PHP with Apache via in-process mod_php

While as the four potential combinations of two scripting languages and two web servers are obvious, the methods for connecting scripting languages and web servers are rather arcane. We chose connectors and connection methods based on availability and general practice. mod_jk is a commonly used connector between Apache and Tomcat using the Apache JServ Protocol (AJP). FCGI (Fast Common Gate Way Interface) is a protocol developed by Open Market to improve the performance and usability of the CGI model for web server to back-end (e.g., scripting language engine) communication which is commonly used with the Lighttpd web server. In our test, the Lighttpd mod_proxy module serves as a general purpose connector between Tomcat and the Lighttpd web server. mod_php is a dynamically loadable module for Apache which enables PHP script processing within the web server process via direct function calls rather than interprocess communication as used by the other methods. With Apache, mod_php is more common than FCGI for PHP script processing.

4.3 Tuning Considerations

Significant tuning effort was expended to ensure that performance was not limited by obvious configuration limitations or trivial system resource limitations. We removed unused daemons, services, and web server modules to reduce computational noise [8].

When initial tests suffered from thrashing under high loads, we added more physical memory, and paid attention to memory related tuning [6]. We considered guidelines used by published SPECweb2005 results [12], and techniques described in Linux, Apache, PHP, and Tomcat reference books and primary websites [2, 4, 6, 7, 8, 9, 14]. Although the Lighttpd web server is designed as a minimally threaded asynchronous event-handling program, with Apache we used the single-threaded/multi-process "prefork" model, since it considered more reliable and is more commonly used than the multi-threaded "worker" model. The significant tuning parameters that we found beneficial in our environment include the following.

Table 3. Significant Tuning Parameters

Tuning Modification	Benefit
/etc/security/limits.conf nofile 65536	Allow more files/sockets to be simultaneously opened by specific user.
sysctl fs.file-max=1000000	Allow more files/sockets to be open simultaneously.
Apache KeepAliveTimeout 2 on SUT	Reduce time an httpd process spends waiting for client response.
Apache KeepAliveTimeout 28800 on BESIM	Enable BESIM to use persistent http connections to reduce connection restart overhead.
Apache ServerLimit 1200	Specify enough httpd processes so that connection availability is not a bottleneck, yet not so many that httpd process memory usage causes thrashing.
Apache MaxRequestsPerChild 0	Avoid overhead of having httpd processes restarted after receiving a certain number of requests.
sysctl net.core.so.maxconn=10000	Increase the connection queue size to prevent denied connections.
vm.swappiness = 50	Improve caching throughput.
max*threads in tomcat5/server.xml = 15000	Improve the response time provided by JSP.
APC extension compiled into PHP	Improve PHP processing time. (Comparable to using JIT in Java.)
tmpfs filesystem used for /tmp	Improved performance for access to temporary files in /tmp.
Lighttpd max-procs=16, max-connections=8192, max-fds=16484, max-worker = 2	Ensure that lighttpd has sufficient sockets and FCGI processes to avoid bottlenecks.
Non-error logging minimalized	Avoid unnecessary overhead.
Debug modes disabled	Avoid unnecessary overhead.

5 PHP/JSP Performance Benchmark Results

5.1 Overall Performance

Figure 4 shows the maximum performance for each configuration and scenario, as determined by the maximum number of simultaneous sessions (e.g., users) which can be supported with acceptable Quality Of Service as defined by SPEC. The results were largely consistent between test scenarios, showing that JSP tended to perform better than PHP (yet PHP with Lighttpd performs nearly as well as the JSP test cases), and Lighttpd tends to perform better than Apache (yet, JSP with Apache performs nearly as well as Lighttpd). Although the Ecommerce test scenario stands as it handles as much as 50% more simultaneous sessions than the other scenarios, since the load per session is not normalized between test scenarios, one must conclude that a single user SPECweb2005 Ecommerce scenario session load is less than that of either a Banking or Ecommerce scenario user session load. However, the fact that the high performing JSP/Apache, JSP/Lighttpd, and FCGI PHP/Lighttpd configurations had a higher percentage performance increase in the Ecommerce scenario than Apache using either mod_php or FCGI PHP does emphasize the superiority of these configurations.

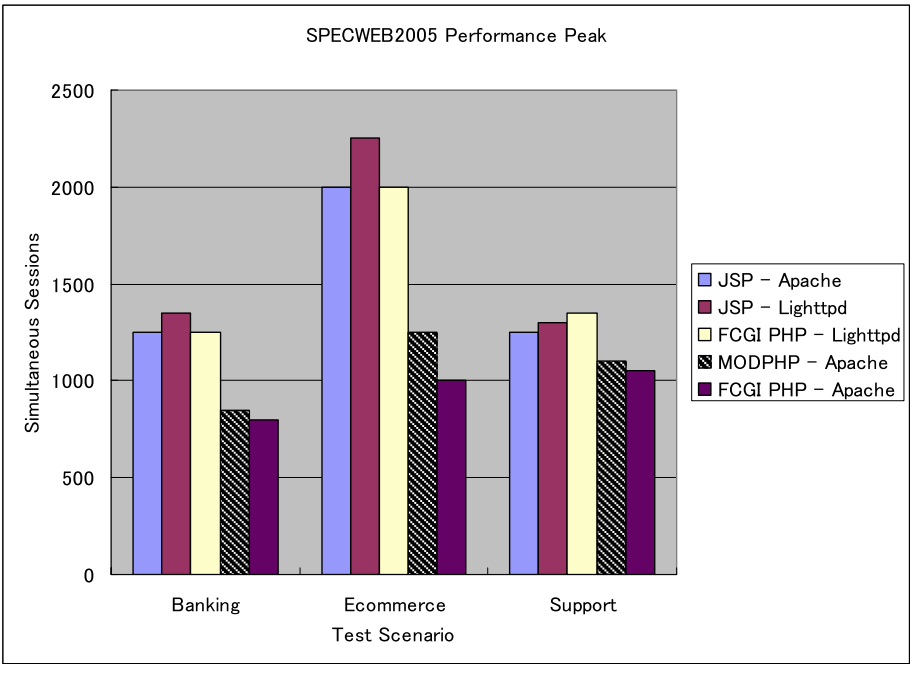


Fig. 4. SPECweb2005 Performance Peak

5.2 Throughput Results

Figures 5-7 show the number of tolerable (or better) requests fulfilled for each of the configurations. At low loads, throughput performance is not gated by SUT resources,

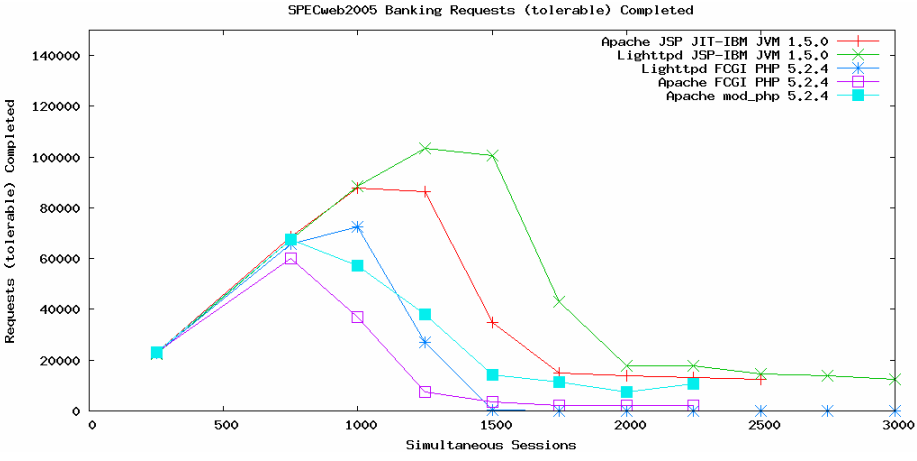


Fig. 5. SPECweb2005 Banking Scenario (Tolerable or better) Requests Completed

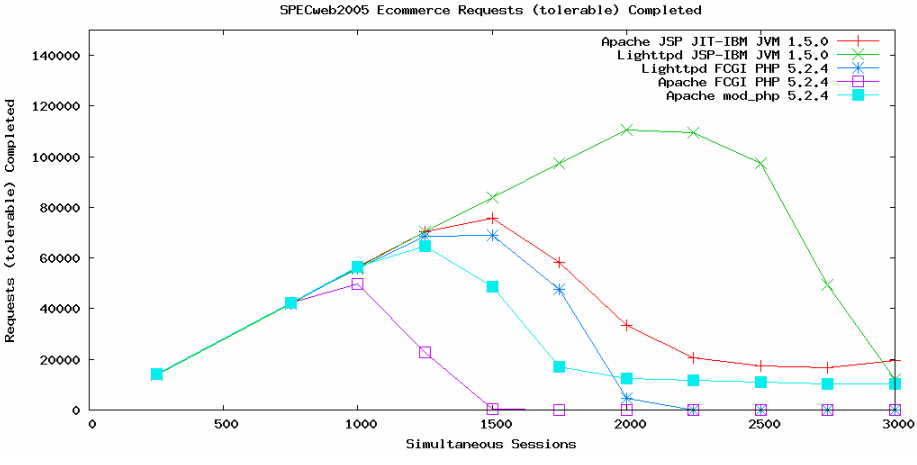


Fig. 6. SPECweb2005 Ecommerce Scenario (Tolerable or Better) Requests Completed

but rather simply by the amount of load placed by the SPECweb2005 test harness, hence at low loads all configurations demonstrate nearly the same throughput. JSP with both servers demonstrated the highest peak throughput in all tests, and generally performed better than PHP under high loads.

Although the performance of PHP in performing fine grain tasks such as executing trivial function calls and simple instructions has been shown to be hundreds of times slower than C, PHP does relatively better at coarse grain activities such as calling complex external libraries to perform actions such as DB access [10]. Ramana and Prabhakar [10] use micro benchmarks to demonstrate that file I/O on PHP is more efficient than, for instance, calculating Fibonacci numbers in PHP. (These results are also consistent with the micro benchmarks we used in Section 3 of this paper.) Thus

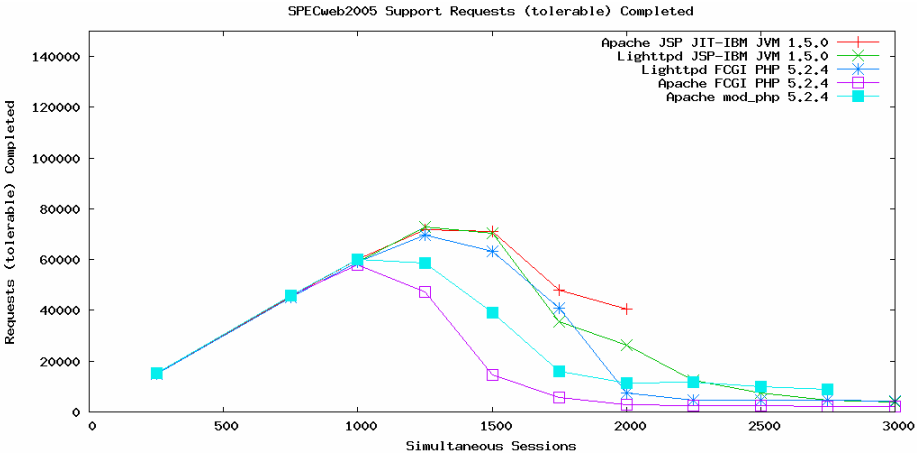


Fig. 7. SPECweb2005 Support Scenario (Tolerable or Better) Requests Completed

we theorize that although all scenarios in SPECweb2005 contain a significant number of fine grain tasks, the high level of file I/O performed in the SPECweb2005 Support scenario allowed PHP to narrow the performance gap with JSP under high loads in this case, as seen in Figure 7. This result implies that micro benchmarks of read performance for large static files would be comparable between PHP and JSP.

Figures 8-12 show detailed results of the Ecommerce scenario for each of our five configurations with test loads from 250 to 3000 simultaneous sessions. Similar results are observed with the Banking and Support scenarios, which are omitted to save space. Data on the number and quality of requests serviced at each point is gathered and shown in these graphs. A "Good Response" is one that is returned to the user within 2-3 seconds (depending on the scenario), a "Tolerable Response" is one that is returned within 4-5 seconds (depending on the scenario), a "Failed Response" is one that returns after that, and a "Validation Error" is a response which is incorrect irregardless of how fast or slow it is. As observed earlier, performance under low loads is the same with each configuration, since the limiting factor is simply the load provided by the SPECweb test suite. As load increases, the expected shifting of request categorization from Good to Tolerable to Failed is observable with all configurations. This shifting can be directly predicted by the increase in average response time reported by the SPECweb2005 test harness. The JSP Lighttpd configuration demonstrated the best performance, but the JSP/Apache and PHP (mod_php) Apache configurations continued to service 10-15% of their requests with good Quality of Service even under extremely high loads, where the other configurations did not. This indicates a wider standard deviation among request response time, implying a potentially "unfair" (e.g., not FIFO) scheduling algorithm with configurations that continue to return a percentage of "Good Responses" under very high load.

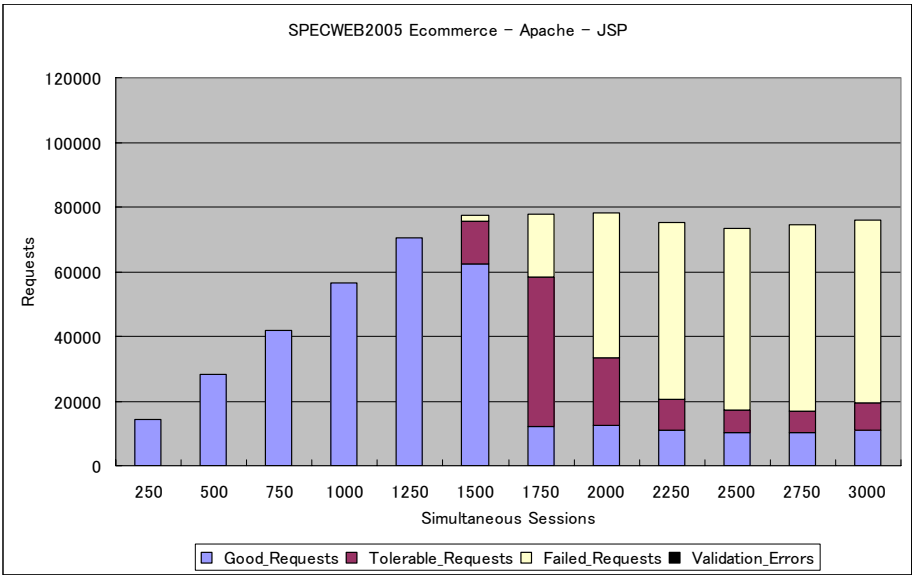


Fig. 8. SPECweb2005 Ecommerce Performance with JSP and Apache

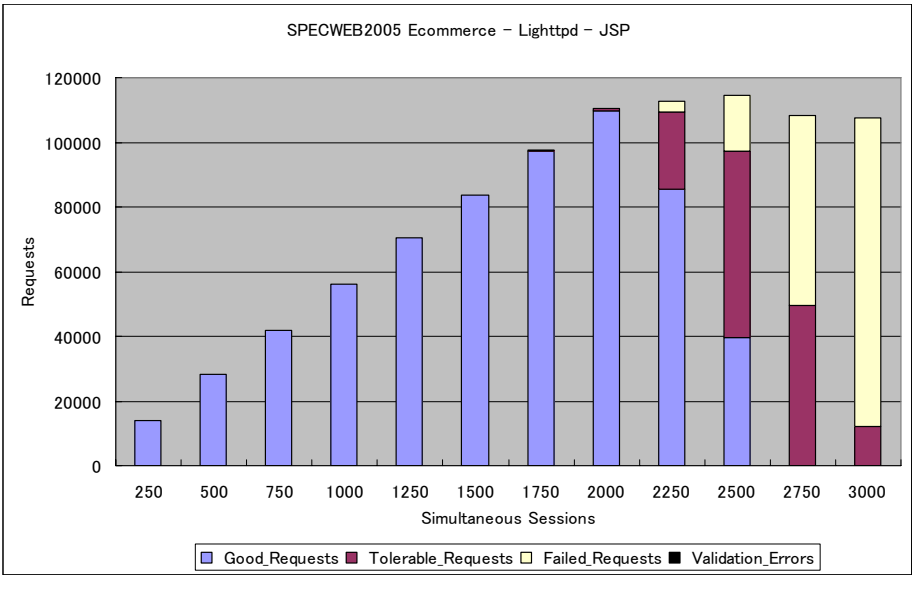


Fig. 9. SPECweb2005 Ecommerce Performance with JSP and Lighttpd

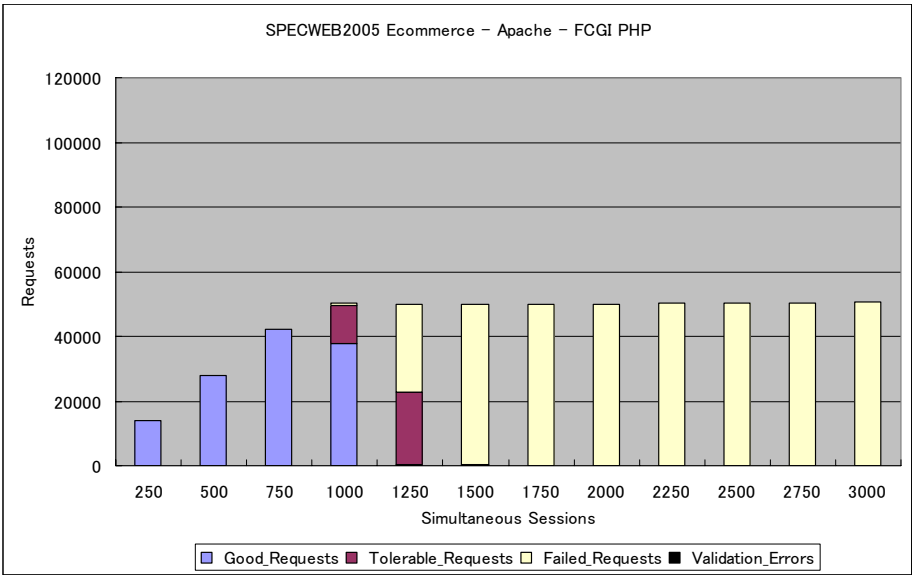


Fig. 10. SPECweb2005 Ecommerce Performance with PHP and Apache (via FCGI)

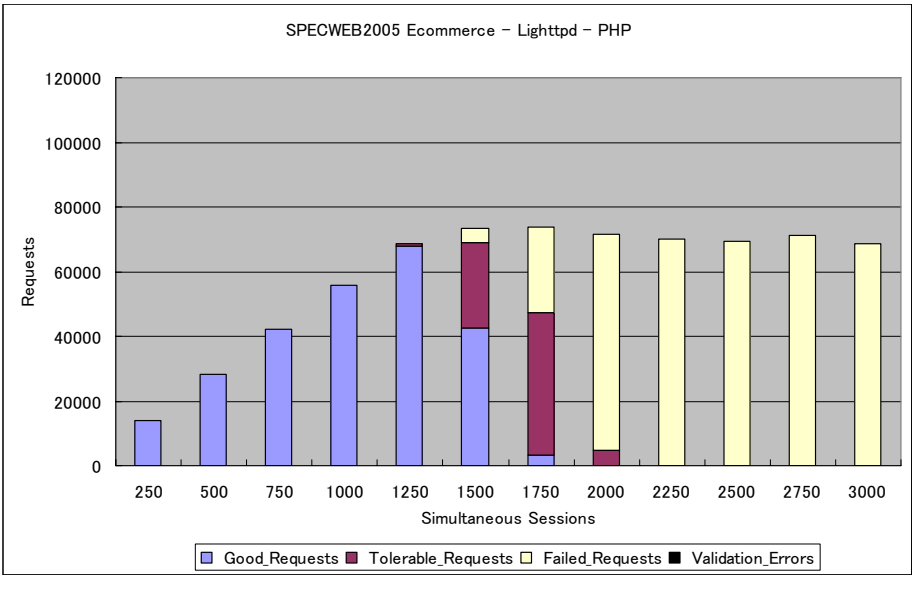


Fig. 11. SPECweb2005 Ecommerce Performance with PHP and Lighttpd (via FCGI)

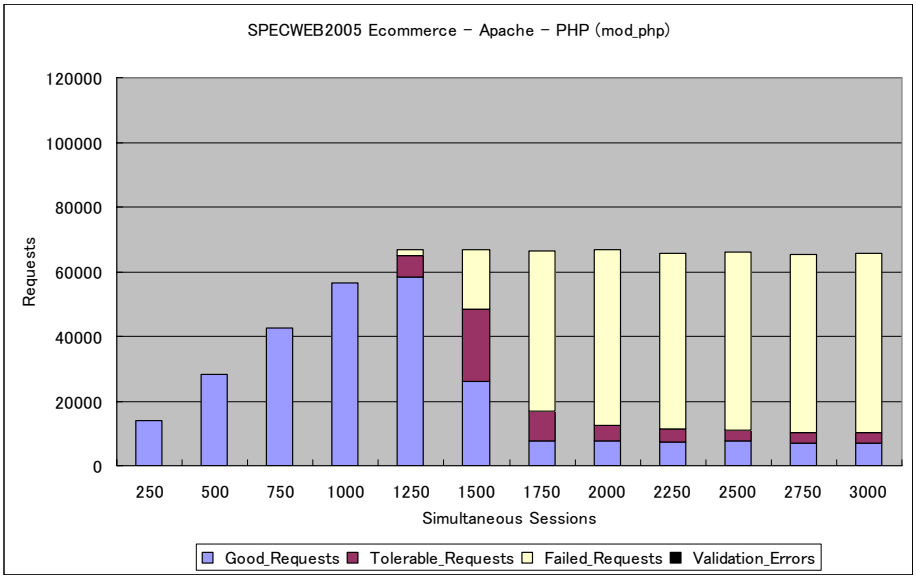


Fig. 12. SPECweb2005 Ecommerce Performance with mod_php and Apache

5.3 CPU Usage

Not surprisingly, using oprofile to profile CPU usage for each test scenario at the maximum throughput level shows that the ratio of CPU time spent in script engine vs. web server depends on both the test scenario and the web server configuration, as seen in Figures 13-15. This implies that improvements to either the language runtime, or the web server will result in performance increase. In Figure 14 we observe that encryption accounted for a large amount of web server CPU time when used (e.g., in the Banking scenario), and of course that scenarios with a higher percentage of

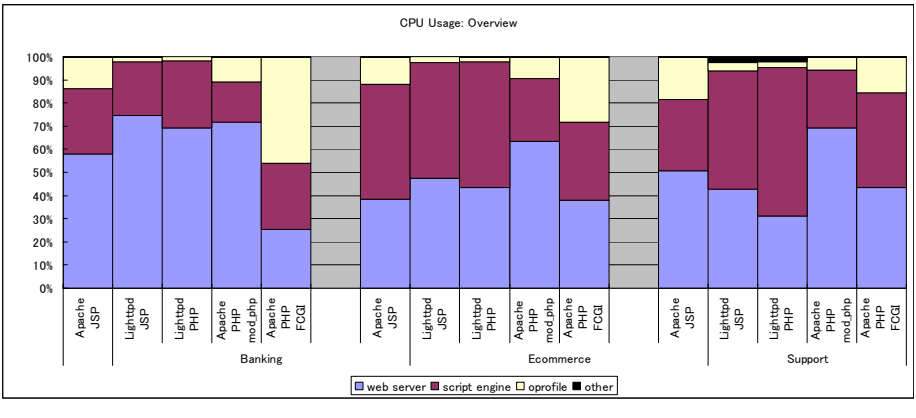


Fig. 13. High-Level View of CPU Usage for Each SPECweb Scenario

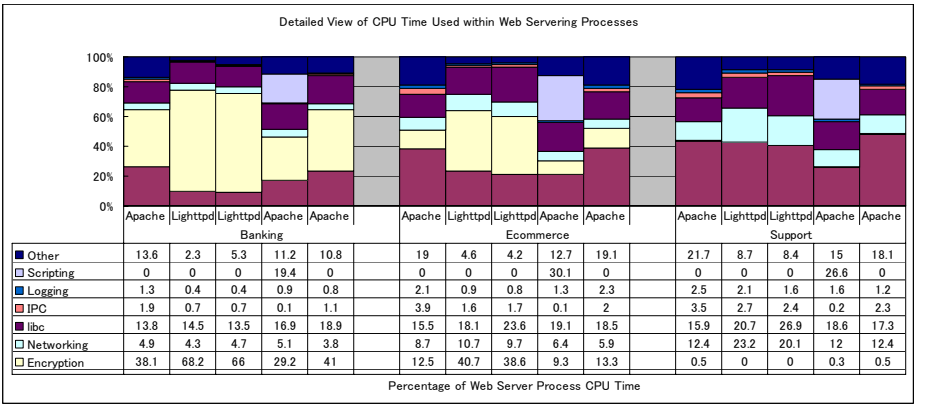


Fig. 14. Detailed View of CPU Time Used within Web Serving Processes

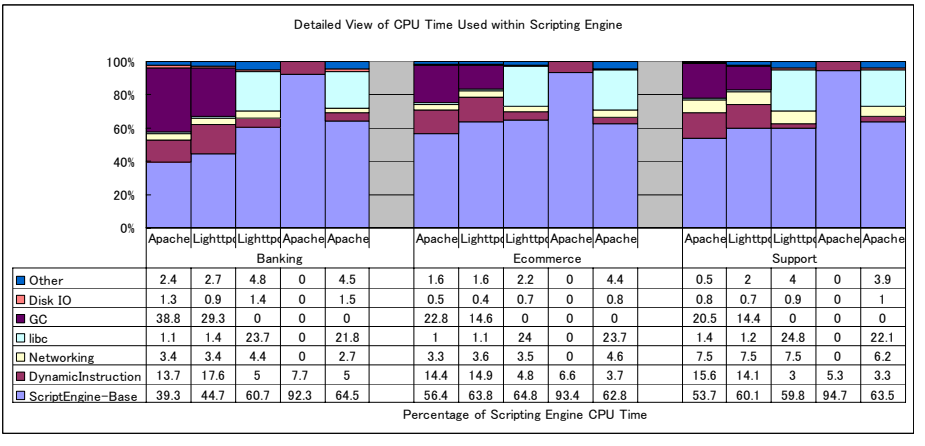


Fig. 15. Detailed View of CPU Time Used within Scripting Engine

dynamic data created by scripting engines tended to use more time in the script engine. The high percentage of SSL computation time spent in the Lighttpd as compared with Apache was puzzling until we identified that SSL connection negotiation data is not shared among multiple Lighttpd processes as it is with Apache. Data from `vmstat` show that the kernel accounted for 34-44% and user time accounted for 36-59% of CPU time. The seemingly high levels of system time are reasonable considering the disk and network I/O involved in running the SPECweb benchmark. At the function level, the `memcpy()` function call was observed as being a significant consumer of CPU in every configuration, implying that additional application of the zero-copy principal may be warranted [19].

6 Discussion

One of the first questions which comes to mind when reviewing the performance benchmark results is, "Why does JSP tend to perform better then PHP under high loads?" One major reason is the Java Just in Time (JIT) Compiler. Although JIT has been compared with PHP APC, APC is merely a bytecode cache which reduces the need for re-interpretation of source code, whereas JIT enables the execution of highly optimized local machine instructions. This is reflected in Figure 14, where Java with JIT shows the least time spent in the runtime engine. Another factor is that JSP realizes parallelization through the threading model, whereas the commonly used Apache worker/mod_php approach adopted in this testing realizes parallelization through the use of multiple processes. Thus under high CPU loads, one would expect less scheduling and context switch overhead with the threading model used with the JSP implementation.

Another seemingly anomalous point is that PHP used with Lighttpd outperformed JSP under high loads in the Support scenario, implying that PHP can handle I/O better than JSP. Initially, one would expect different performance characteristics of a program such as the PHP runtime which is written in low level C, and that of the Java based JSP environment. The difference in web server architectures also plays a factor, where the asynchronous event-handling approach used in Lighttpd appears preferable to Apache's multi-process "prefork" approach. The use of in-process language processing appears successful when reasonably lightweight, as is the case with mod_php. Likewise, external language processing as with Tomcat seems to be successful by avoiding replication of a heavy-weight JVM for each process. The external language processing approach via FCGI also appears highly successful with Lighttpd. The internal mod_php approach offers the advantage that data read from disk is immediately available to Apache, since the PHP engine runs in the same address space as the Apache daemon. However, the JVM used with JSP as well as PHP accessed via FCGI runs in a separate process and thus incurs domain socket communication overhead to transmit file data from one process to another, as well as potential inefficiencies from process context switching and coordination.

7 Related Work

Titchkosky and associates established that serving dynamic web content can reduce throughput by 8 times as compared with static web content [13], providing our team with encouragement to identify methods to reduce the negative performance impact of using scripted language dynamic web content. Ramana and Prabhakar analyzed the performance differences between PHP and compiled languages such as C, pointing out the relative performance downside of PHP [10], which corresponds with our tests on pure-script implemented benchmarks vs. scripts using standard class library or PHP extensions implemented in C language. The upside of our benchmarking is that we found the use of C-language PHP extensions for computationally intensive functions to enable PHP scripts to perform comparably with Java. Cecchet and colleagues analyze various middleware architectures based on technology such as Apache, PHP, Tomcat, MySQL, and JOnAS [3, 17], which helped guide our methodology. Warner and Worley

describe the importance of using technology such as PHP rather than just JSP for real-world benchmarking with SPECweb2005 [18]. We have contributed to this line of reasoning as we were motivated to write this paper since we have not seen data from an industry standard web server benchmark that provides a detailed comparison of the performance PHP and JSP as a web server dynamic scripting language.

8 Conclusion

When implementing a web server system which will never experience high load, or in which performance, throughput, and reliability under high load is not an issue, then the use of any of the analyzed languages or web servers will achieve similar performance results. If outstanding performance and throughput is the primary goal, then the use of JSP over PHP is advisable. However, if a 5-10% difference in throughput and performance is acceptable, then the implementer of a web system can achieve similar results using either PHP or JSP. In which case, other requirements such as developer language familiarity and programming efficiency, maintainability, security, reliability, middleware compatibility, etc. would be the deciding factors. It is also reassuring to developers of both language runtimes and web servers, that enhancements to either can offer performance improvements to the community.

Acknowledgements

We are appreciative of the many useful discussions with Graeme Johnson and Andrew Low, from the IBM Ottawa Software Lab, which have provided valuable direction. Mathematical guidance from Mei Kobayashi, and perceptive feedback from the Systems Department, both at the IBM Tokyo Research Laboratory resulted in a more consistent and rigorous analysis. We are also deeply indebted to the feedback and comments regarding PHP and SPECweb2005 testing which we received from the PHP team at IBM Hursley.

References

1. Apache Software Foundation (2008), `http://tomcat.apache.org`
2. Bergsten, H.: Java Server Pages. O'Reilly, Sebastopol (2003)
3. Cecchet, E., Chanda, A., Elnikety, S., Marguerite, J., Zwaenepoel, W.: Performance Comparison of Middleware Architectures for Generating Dynamic Web Content. In: 4th ACM/IFIP/USENIX International Middleware Conference (2003)
4. Chopra, V., Galbraith, B., et al.: Professional Apache Tomcat (2003) ISBN 0-764-5372-5
5. IBM (2006), `http://www-03.ibm.com/press/us/en/pressrelease/19822.wss`
6. Johnson, S., Huizenga, G., Pulavarty, B.: Performance Tuning for Linux Servers. IBM Press (2005) ISBN 0-131-44753-X
7. Lecky-Thompson, E., Eide-Goodman, H., Nowicki, S., Cove, A.: Professional PHP5. Wrox Press (2005) ISBN 0-764-57282-2

8. Petrini, F., Kerbyson, D., Pakin, S.: The case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In: Proceedings of IEEE/ACM SC (2004)
9. PHP Group (2008), http://www.php.net
10. Ramana, U., Prabhakar, T.: Some Experiments with the Performance of LAMP Architecture. In: Proceedings of the 2005 Fifth International Conference on Computer and Information Technology (2005)
11. Security Space (2007), http://securityspace.com
12. Standard Performance Evaluation Corporation (2008), http://www.spec.org
13. Titchkosky, L., Arlitt, M., Williamson, C.: A Performance Comparison of Dynamic Web Technologies. In: 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (2003)
14. Wainwright, P.: Professional Apache 2.0 (2002) ISBN 1-861-00822-1
15. Wu, A.W., Wang, H., Wilkins, D.: Performance Comparison of Alternative Solutions For Web-To-Database Applications. In: Proceedings of the Southern Conference on Computing (2000)
16. Garcia, D.F., Garcia, J.: TPC-W E-Commerce Benchmark Evaluation. IEEE Computer 36(2), 52–48 (2003)
17. Amza, C., et al.: Specification and implementation of dynamic Web site benchmarks. In: Proceedings of the 5th IEEE Workshop on Workload Characterization (2002)
18. Warner, S., Worley, J.: SPECweb2005 in the Real World: Using Internet Information Server (IIS) and PHP. In: 2008 SPEC Benchmark Workshop (2008)
19. Stancevic, D.: Zero Copy I: User-Mode Perspective. Linux Journal 3(105) (2003)