The Performance of Web-based 2-tier Middleware Systems

Johnson Dehinbo
Tshwane University of Technology, Pretoria, South Africa

Dehinbo.j@tut.ac.za

Abstract

The overall aim of this study is to determine the performance of selected web-based dynamic middleware systems that are used for designing and implementing dynamic web application systems. This is necessary in a world where more applications are moving to the web, and slow performance of such applications can discourage users, thereby reducing profit, and reduce programmers' productivity and quality of applications due to slow testing and execution.

Java Servlets, Java Server Pages (JSP), Microsoft Active Server Pages (ASP), and Personal Home Page (PHP) were used to perform some operations on the server, like retrieving all records from a database stored on the server. The time taken since the query is initiated from the browser, to the time the query result is displayed on the client browser were measured for each of the four middleware systems as an estimate for their performance. Records were increased in multiples of thousands to estimate scalability along with the performance. PHP proved to be more efficient and more scalable.

Keywords: Performance, Latency, Web-based, middleware, platforms

Introduction

There are many web based dynamic middleware systems for implementing the dynamic programs on the World Wide Web, and new designers have the big task of choosing the most appropriate implementation. The choice made may have an effect on the speed of execution of the web applications thereby affecting the performance or efficiency of the designed system.

Web application developers will not perform at their best capacity level if the middleware chosen by their management is slow during testing. For the industry, inadequate testing and debugging due to low speed of execution and access to stored objects could also lead to low productivity. It could as well lead to low quality of developed applications. The result is that some web applications development takes longer time than budgeted leading to inflated costs.

This is also applicable to students that need to test and debug their programs. A programming platform with low execution time will lead to frustration due to the slow speed of recurrent testing and debugging in the lab. This will have tremendous negative impact on adequate software testing and debugging, especially in a student computer laboratory session, where the time allocated is limited, thereby limiting students expertise of the subject.

Similarly, users will become frustrated if the web pages and desired information are not constantly available, or if

the system slowly responds to users' request. This problem originated from occasional low speed of access to stored objects possibly due to the implementation modes relating to the efficiency of web-based dynamic middleware systems being used to develop the web-based applications. Moreover, the low speed of execution of web applications leads to impatience by users, with disastrous impact for business, as further illustrated by Marshak (2003, p.1) which given by the following statement:

> "Impatience with poor performance is the most common reason that makes visitors terminate their visit at web sites. For e-commerce sites, such abandonment translates into lost revenue" (Marshak, 2003, p.1).

In earlier days, slow web sites were tolerated and expected. Even many users had slow connections to the network, so they hardly notice if the server was slow. Clearly the expectations have changed. Sites must be available 24 hours a day and 7 days a week because customers are on different parts of the world and peak time unknown. The system must be fast enough to satisfy increasingly demanding users, because competitive sites are "just a click away" (Treese and Stewart, 2003, p.173). This means that while customers visiting a traditional shop may feel reluctant to leave when not satisfied, customers using the web applications could easily run away. The above statement is confirmed by Singh (2002), which states that:

> "customers who feel they have lost control can simply leave the site without any embarrassment - unlike a user who is standing at a checkout till in a supermarket."

This is because no shop attendant is watching, and he/she doesn't have to walk some distance to the next shop. Therefore, as indicated by Marshak (2003, p.1), measuring the delay experienced by its customers is of high importance to a web site as these measurements are critical for analyzing the site behavior and to size its components.

The Research Question

Web application designers, especially the new and inexperienced ones have the big task of choosing the most appropriate web-based dynamic middleware system that will speed up their system development processes, and most importantly produce an efficient implementation. An important question, therefore, is:

> "Is there any one web-based dynamic middleware system that will be effective for designing robust, fast and responsive web applications fast?"

Objectives of the Study

In order to answer the research questions, there is the need to formulate the objectives in clear, measurable and achievable/manageable terms. The objectives of the study are therefore given below:

1. Review the evolution of web-based dynamic middleware systems
2. Develop experimentation to measure the latency of the designed web-based application systems in remote retrieval of records from database. This serves to estimate the throughput of the platform in the recurrent testing (compilation/interpretation-execution) in a practical class.
3. Estimate the scalability of the platforms.

The Importance and Use of the Study

While this study reviews and investigates the scalability and the overall performance of those web-based middleware systems, the purpose is not to persuade people that one middleware is bet-

ter than the other, but to help you make a more informed decision as to which middleware language is more efficient based on certain operating environment.

The main benefit of the study will be the educative analysis in choosing most appropriate web-based middleware platform to enhance the performance of developed applications leading to higher productivity in the software development and services industries. It will therefore be of benefit to institutions in deciding which web-based dynamic middleware systems to introduce to "beginning students" due to its potential to maximize limited time in the labs. The students themselves, especially the Information Technology students in tertiary institutions, will benefit from the study.

Therefore, the primary beneficiaries will be organizations that desire web applications, software organizations, web application developers, researchers, postgraduate students and individuals that utilize web-based dynamic middleware systems in the design of their web-based applications. Such people will definitely become empowered in making the best choice for their implementation situation.

The secondary beneficiaries will be users that desire efficient and fast transactions over the web, as well as individuals that desire a systematic mastery of web-based dynamic middleware systems. Therefore, the study will contribute immense knowledge to the field of web-based programming and distributed computing.

Literature Review

The Need for Analyzing Web Based Dynamic Middleware Systems

Prechelt (2000, p.1) pointed out that when it comes to the pros and cons of various programming languages, programmers and computer scientists alike usually hold strong opinions. However, analyzing programming languages, development platforms and tools is very important as illustrated below:

> "Comparisons across programming styles, or paradigms, are difficult to carry out, but are nevertheless important for understanding how different styles of programming affect the learning of novice programmers" (Wiedenbeck, Ramalingam, Sarasamma, & Corritore, 1999, p.5).

According to Lim (2002, p.2) information systems / computer science departments need to reexamine their curricula in order to prepare students to face the challenge of being productive in a computing world that is now swamped with web technologies. The author perceives that the choice of web-based dynamic middleware systems need to be backed with evidence from relevant literatures, information from practicing web developers and empirical experimental programming results. As confirmed by Apte, Hansen, and Reeser (2003, p.3) there is a real need to make a technology choice for developing software that would support a Web based service.

Unfortunately, Apte et al. (2003, p.3) noted that a study of existing literature showed varying conclusions about the superiority of one technology over the other. Moreover, Prechelt (2000, p.1) pointed out that when it comes to the pros and cons of various programming languages, programmers and computer scientists alike usually hold strong opinions.

Finally, Ashenfelter explains the need for analyzing web-based platforms in the statement below:

> "web development tool need to be analyzed in terms of its purpose (what it is designed to do), technology (ease of use, robustness, scalability, security, performance, etc.), support

> (portability, cost, ISP support), and how well it works in the real world" (Ashenfelter, 1999).

Performance, Latency and Throughput

The term "performance" as used in this study, refers to the total time in which the web-based programs are executed. For many systems, poor performance is often an inconvenience and perhaps a source of complaints, but the users keep using the system. In internet commerce systems, performance problems are much more than an inconvenience – they can be a disaster for a business, putting off customers and giving competitors an advantage (Treese & Stewart, 2002, p.177). So, building sites that are fast, reliable and scalable is probably the most challenging part of creating internet commerce systems (Treese & Stewart, 2002, p.173).

According to Marshak and Levy (2003, p.1), the central performance problem in the World Wide Web, in recent years, is user perceived latency. Treese and Stewart (2003, p.177) state that the performance of a system can be measured in many ways including using metrics that examine latency and throughput. Latency is a measure of how long it takes to complete a given operation e.g. how long it takes to download a web page. Throughput is a measure of how many operations can be completed in a given time e.g. how many web pages can the server deliver in an hour. This is a measure of transactions per second.

In the comparison of latency and throughput, Treese and Stewart (2003, p.177) further stated that while latency tells you about the experience of a particular user (on average), throughput tells you how many users the system can handle. As throughput increases for example, the latency as seen by any given user may increase. As illustrated by Treese and Stewart:

> "This is like driving in heavy traffic: the number of cars moving down the highway is greater than normal (i.e. greater throughput) but the average speed is lower (yielding higher latency)" (Treese and Stewart, 2003, p.177).

Therefore, it would be sufficient to measure the performance by using latency, which is estimated in this study as the time to retrieve stored objects.

Framework for Performance Comparisons

Renaud, Bishop, Lo, van Zyl and Worrall (2003) reported on the works of Hsier and Sivakumar (2001) and Shousha, Petriu, Jalnapurkar and Ngo (1998) stating that the measurement of software performance by and for experts is a well known task. From various other previous works, Renaud, et al. (2003) recalled that various metrics can be used to measure performance of algorithms in distributed systems, namely: response or waiting time, synch delay, number of messages exchanged, throughput, communication delay, node fairness, CPU cycle usage, and memory usage.

Since no single metric can be optimal for all applications, it is necessary to ensure that the developer can obtain metrics that reflect the need, priorities and workloads of the particular distributed system. The first four metrics were most suited to specifically measuring algorithm performance. The fifth metric is more dependent on network load than a specific algorithm, the sixth is difficult to quantify and the seventh and eighth produce measurement of debatable merit in judging algorithm efficacy (Renaud, et al., 2003).

For this study, the performance metric that will be used is the response or waiting time, which has also been used by other researchers in their studies such as Cooper (2001).

Other Related Studies

In the survey of middlewares by Cooper (2001), it was concluded that ColdFussion is fast to learn and fast to use, and CGI is also fast to learn. He then mentioned that Servlets are hard to learn and use - even by someone who already knew Java. We agree with this because in the study (Dehinbo, 2004b) Java has the highest line of code for a simple solution to the given problem. Bishop and Hurter (1999) also confirmed that a Java version of a server program in (Bishop, 1998) is nearly four times as long as its Perl's version.

In an empirical comparison of seven programming languages, Prechelt (2000, p.29) observed that designing and writing programs in the scripting languages namely Perl, Python, Rexx, or Tcl takes no more than half as much time as writing it in C, C++, or Java. Moreover, the resulting program is only half as long. He therefore concluded that the scripting languages offer reasonable alternatives to other full programming languages, and they may offer significant advantages with respect to programmer productivity, at least for reasonably small programs.

Marshak and Levy (2003, p.3) propose a new approach to estimate user perceived latency that is based on server side measurement. The approach uses a new technique in which a special tiny and practically unnoticeable zero sized inline HTTP object, called the sentry. It is placed at the end of the HTML document so that it does not add overhead, and it tracks the arrival time to the user.

In a study to compare the performance of middleware architectures for generating dynamic web content, Cecchet, Chanda, Elnikety, Marguerite, and Zwaenepoel (2003) evaluate three specific mechanisms namely PHP, Java servlets, and Enterprise Java Beans (EJB). The study measures the performance of these three architectures using two applications benchmarks: an online bookstore that stresses the server back-end and an auction site that stresses the server front-end. It was found out that EJB has lower performance than both PHP and Java servlets, with Java servlets also having lower performance than PHP (Cecchet, et al., 2003, p.1).

Cecchet attributes PHP's better performance to the fact that it executes as a module in the Web server, sharing the same process (address space), thereby minimizing communication overhead between the Web server and the scripts. This is unlike Java servlets, which run in a JVM as a separate process from the Web server and so can even be placed on a separate machine. However, it is observed that the flexible ability of Java servlets to execute on a separate machine from the Web server and their ability to perform synchronization leads to better performance when the front-end is the bottleneck (Cecchet, et al., 2003, p.5).

Hartman (2001) examined some tools for dynamic Web sites namely ASP, PHP and ASP.NET. He mentioned three factors that complicate choosing a scripting environment that will make the Web site to be fast, database-driven, reliable, and created on time and under budget. First, there is the issue of culture among developers.

> "No matter how rational their programming code might be, most programmer's choice of scripting technology has a lot to do with the ideological camps to which they belong. If they love to tinker with source code because it lets them develop solutions that are a tad more efficient than off-the-shelf products, and if their cubicles are embellished with defaced portraits of Bill Gates, it is a good bet that they will prefer to use PHP. If they love the convenience and efficiency of existing integrated technology solutions, they probably prefer to use ASP" (Hartman, 2001).

He also mentioned that he has encountered very few developers who are equally willing to use both, or who can talk about "the other" technology without a trace of disdain.

Hartman further states that the second factor that complicates choosing a scripting environment is that a Web site's future scalability and functional requirements, although hard to predict, are necessarily a part of the equation. The choice between JSP, PHP and ASP (or its successor ASP.NET) might restrict which servers and platforms the site could run on or impact the feasibility of developing future features, such as database-linked connectivity with extranet partner sites (Hartman, 2001).

Hartman's study came up with some conclusions and predictions, which includes the fact that ASP.NET, promises to be a faster and more efficient environment than ASP, and possibly PHP. However, this study is considering platforms that can be used by students both in the class and at home. On this note, ASP.NET is not affordable by students, and therefore we still use ASP rather than ASP.NET.

Moreover, another way in which this study differs from related studies is that it considers it very important to relate the performance of the platforms with similar architecture. We consider it necessary to exclude EJB because its architecture does not seem to be in the same category with PHP and Java servlets, being more of an enterprise multi-server platform. Instead, we include JSP and ASP, which both embeds code directly into an HTML page like PHP. Moreover, ASP have the backing of the VB Script language just like JSP and Java servlets have the powers of the Java language, and PHP also have full programming language capabilities.

Methodology

Experimental Approach to the Latency Estimation

The latency experimentation is required to estimate the execution speed for programs written using each of the web-based middleware platforms. This is needed in order to have information on the suitability of the platforms for the recurrent program testing that takes place in a practical laboratory session. In the experimental approach, a client-server system was set up for each of the middleware namely Java Servlet, ASP, JSP and PHP.

For this experimentation, programs were written to retrieve all records from a database stored on the server. The time taken since the query is initiated from the browser to the time the query result is displayed on the client browser was measured for each of the four web-based dynamic middleware platforms. This is used as an estimate for the performance or latency of web-based dynamic middleware systems. A platform that achieves fast execution of programs will shorten the 'write-compile-test-debug-recompile' cycle in the lab.

The System Configuration for the Latency Estimation

The minimum system configuration needed is any computer system capable of being connected to the Internet and being used as a web server. The server used is a Pentium IV 2.4M.Hz with 256MB RAM computer connected to the Internet via LAN card to the University Network. The following are the software configuration necessary:

Java Servlet & Java Server Pages Set-up:

Microsoft Windows XP Professional 2002, Microsoft Access 2000, J2SDK (Java 2), Tomcat Web Server and Servlet/JSP engine.

Active Server Pages (ASP) Set-up:

Microsoft Windows XP Professional 2002, Microsoft Access 2000, Visual Basic, IIS 5 (Internet Information Service 5).

Personal Home Page (PHP) Set-up:

Microsoft Windows XP Professional 2002, Microsoft Access 2000, PHP, IIS 5 (Internet Information Service 5).

The architecture adopted is a 2-tier system, with the client browser connecting to the server, which also hosts the database management system, as utilized in Dehinbo (2004a).

The Database Configuration for the Latency Estimation

In order to estimate the time taken to access and display data from the database, a database was created using Microsoft Access 2000 for all the platforms. The database consists of the following structure:

Studnumber (int), surname (text), initials (text), sex (text), diploma (text)
subject1 (int), subject2 (int), subject3 (int), subject4 (int), subject5 (int), subject6 (int).

We began the program execution with 10000 records because it is at this point that the estimated time goes above 1 second. This is necessary due to the limitation of the now ( ) function that doesn't measure in microseconds. We then double the size of the database to 20000, and then 40000 and 80000 records. This is simply because it is easier to duplicate the whole records than to enter new records.

The Procedure Followed to Obtain the Latency Data

For each platform, one program was written to access and display all the stored records in the database. One program is written as a Java Servlet while another one was written as a JavaServer page (jsp) program. For the ASP setup, the program is written using VBScript. The fourth program was written using PHP. Each of the programs extract the system time at the entry point of the program as "starttime" and extract as "stoptime" the system time at the end of the program. The difference between the "starttime" and the "stoptime" in seconds is taken as the estimated time for the database retrieval operation. This is used as an estimate of the latency value for the web based dynamic platform because most major web applications usually involve database access. The experiment was performed twice and we used the average value.

Latency Hypothesis

The null hypothesis is that there is no significant difference in the performance of each of the web based dynamic middleware systems. The alternate hypothesis is that there is significant difference in the performance of each of the web based dynamic middleware systems. Thus users can make informed choice on web based dynamic middleware systems that is very fast.

Establishing the Reliability of the Latency Experimentation

As observed by (Lerdorf & Tatroe, 2002, p.311), if we reload the scripts above several times, we'll see that the time taken may fluctuate a lot, thus affecting reliability of the experimentation. This is because, as stated by Lerdorf and Tatroe (2002, p.311), the danger of timing a single run of a piece of code is that you may not get a representative machine load – the server might be paging as a user starts, or it may have removed the source file from its cache.

Therefore, a way in which the reliability is increased is by following the approach presented by Lerdorf and Tatroe (2002, p.311), which states that the best way to get an accurate representation of the time it takes to do something is to time repeated runs and look at the average of those times.

Also, in other to ensure accuracy, the database for the web based dynamic middleware systems are having same structures and same number of records. Moreover, the programs were run with different number of records, to also test their scalability.

Again, the programs were run on the same computer to ensure same processor speed. The same computer serves as the client and the server, to minimize any possible effect of congestion on the university network. Moreover, the programs were executed after minor changes were made to the programs and the computer is rebooted, to ensure that they were re-compiled and not just re-loaded from the cache memory and also that the memory is free.

Scope and Assumption of the Latency Experimentation

The programming was limited to data access from relational database, as this is the most common and most important form of utilization of web based dynamic middleware systems. It is therefore assumed that the speed of access of database will be proportional to other form of processing commonly done.

Limitations of the Latency Experimentation

One problem that was encountered was the lack of functions to estimate time up to the microsecond level in ASP. The NOW() function only obtain system time to the latest second. Therefore, we decided to increase the records until the time taking is at least 1 sec. This made us to therefore start with 10,000 records for all the platforms.

Latency Results and Discussions

Sample Output Screen Captures

Each of the four programs written was executed on the browser. Sample outputs from the programs are given below in Figure 1 to 4:

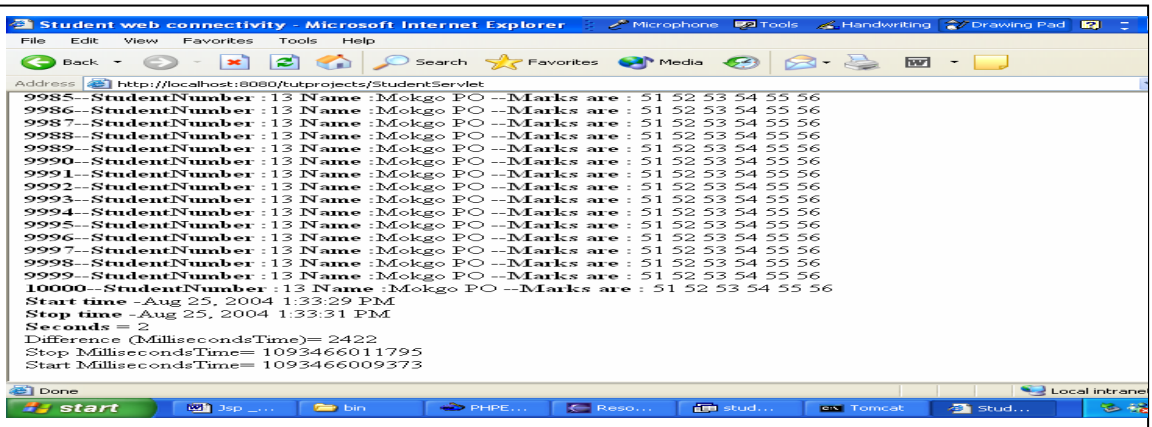

Figure 1. Java Servlet Retrieval Output

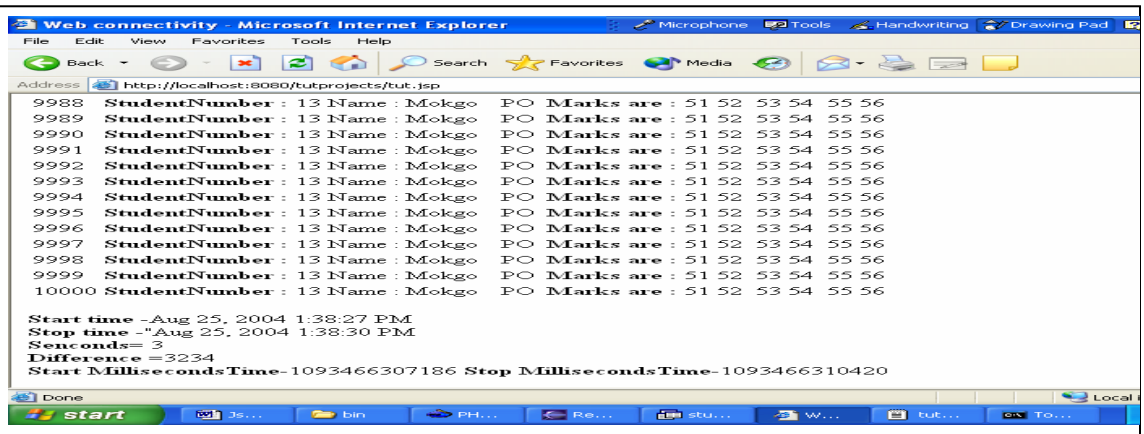

Figure 2. JavaServer Page Output

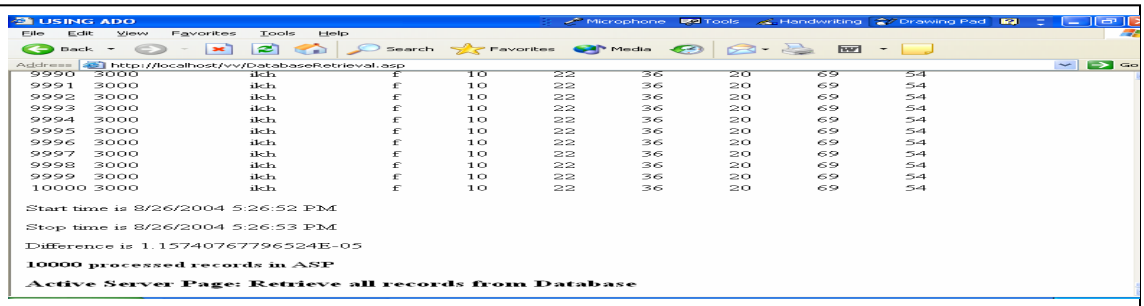

Figure .3. Active Server Pages Output

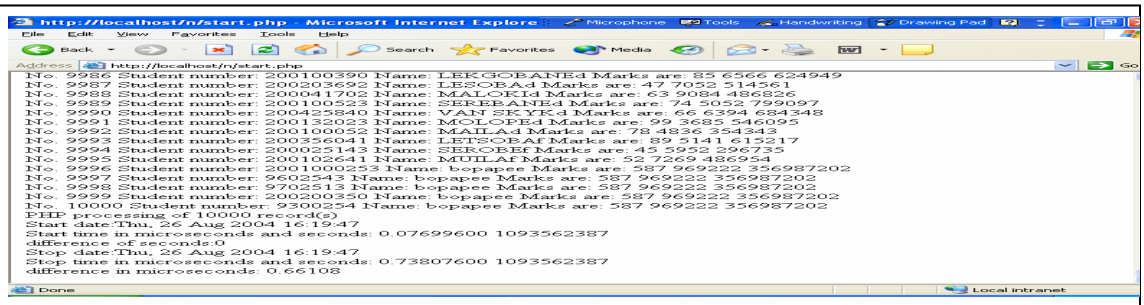

Figure 4. Personal Home Page Output

Latency with Scaling results

According to Treese and Stewart (2003, p.176) scaling is the question of how big a system can grow in various dimensions to provide more service. It can be measured by total number of users, the number of simultaneous users, the transaction volume etc. Treese and Stewart (2003, p.176) further mentioned that Scaling in one dimension typically affects other dimensions e.g. increasing the size of a database to handle more users may sometimes decrease the performance. Therefore,

we are going to measure the performance alongside with scaling, by estimating the response time while increasing the size of the database.

The estimated average time taken for the database retrieval in all the middleware platforms is presented in tabular form is given in the table 1 below. To estimate scalability as well, the records are doubled from 10000 to 20000 to 40000 and finally to 80000. This is simply because it is easier to duplicate the whole records than to enter new records. It is significant to note that ASP and JSP is unable to cope with 80,000 records using the configuration for this experiment. Though this can be solved by increasing the RAM, the fact is still noteworthy given that all the programs are executed on the same computer and that most students may not be able to afford large RAM.

Table 1. Average time taken for the database retrieval as number of records increases

Number of records / Time	Servlets	JSP	ASP	PHP
Time in seconds (10000 records)	2.0	3.0	1.15	0.66
Time in seconds (20000 records)	5.0	6.0	2.31	2.00
Time in seconds (40000 records)	9.0	15.0	4.69	5.0
Time in seconds (80000 records)	20.0			10.5

Discussion of the Latency with Scaling results

From Table 1 above, it is evident that, for the minimum 10,000 records implemented, PHP has the best performance followed by ASP, and followed by Java servlets. JSP has the worst performance. A pictorial view of this result is given in Figures 5 and 6.

This result is in agreement with Cecchet et al. (2003, p.5) which explains PHP's better performance as due to the fact that it executes as a module in the Web server, sharing the same process (address space), thereby minimizing communication overhead between the Web server and the scripts. This is unlike Java servlets that run in a JVM as a separate process from the Web server.

JSP's performance being lower than that of Java servlet could be explained due to the fact that JSP still have to be converted to Servlet before being executed. This explanation could also be responsible for the inability of the JSP to process 80,000 records, as the compilation is already taking part of the available memory and disk resources. Even though ASP is not converting to other temporary form, the interpretation of the codes in the same way as for JSP, unlike the com-

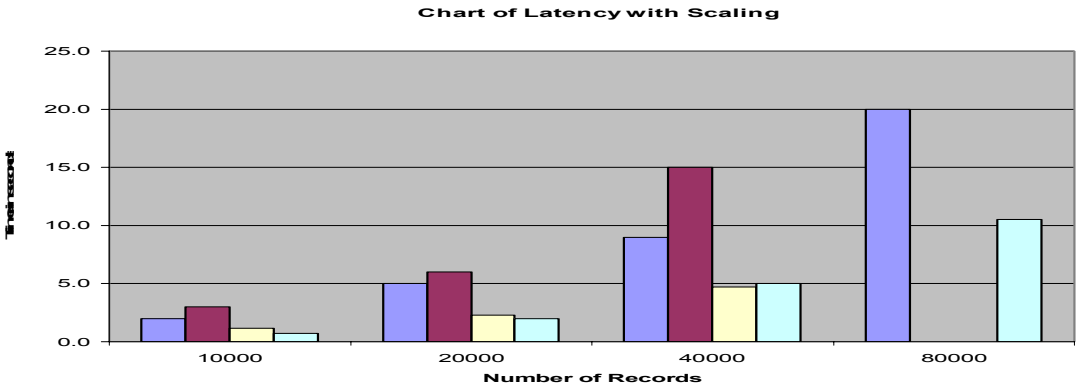

Figure 5. Chart for the Latency with Scaling

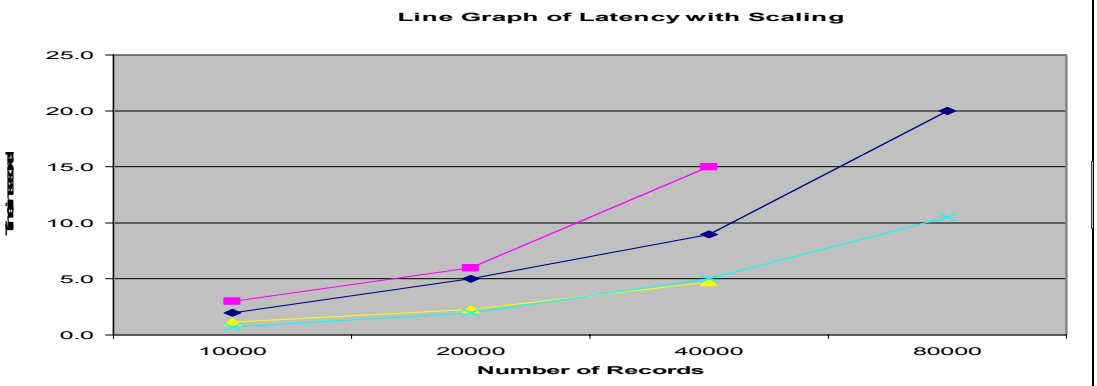

Figure 6. Line Graph for the Latency with Scaling

pilation in Java servlet could also be responsible for the inability of the ASP to process 80,000 records.

For the maximum records of 40,000 records executable by all the platforms using the resources of this experiment's configuration, ASP seems to be trying to outperform PHP. This could be due to the mode of fetching the results. While ASP processes the whole results before display, PHP displays them as it retrieves them sequentially. This would incur some fetch overhead as the number of records increases.

Summary

This study examines the latency experimentation that measured the time for the execution of programs using the specified platforms. The essence is to determine platforms that will shorten the compile-test-debug-recompile cycle in software development, especially in an academic practical laboratory.

We combined the latency experiment with scalability by estimating the performance for 10000 records, 20000, 40000 and 80000 records. The result showed that PHP has the best performance for 10000 records, while ASP tries to outshine PHP for 40,000 records. Given that ASP is unable to cope with 80,000 records using the configuration for this experiment, it is evident that on the average, PHP has the best performance.

Conclusion

Taking a critical look at this study, it aims at finding ways of "designing fast web applications fast". The study is concerned with a measure of how fast it takes to execute developed web applications. Developing fast applications fast will increase productivity and profit. It will also reduce the time spent in testing the developed applications, as well as the quality of the developed applications due to possibility of extensive testing. This is based on the premise that since there is the need for recurrent testing in a practical software development session, a platform that has minimum latency for executing developed applications will be more desirable.

The study concluded that PHP has the best performance on the average. This is in line with other studies such as Cecchet, et al. (2003). Moreover, with respect to scalability, PHP still proved to be very scalable. Therefore, there is significant difference in the performance of the dynamic web-based middleware systems.

References

Apte, V., Hansen, T., & Reeser, P. (2003). Performance comparison of dynamic web platforms. *Computer Communications*, *26* (8), 888-898.

Ashenfelter, J.P. (1999). *Choosing a database for your web site*. NY, USA: Wiley Computer Publishing.

Bishop, J. & Hurter, R. (1999). Competitors to Java: Scripting languages. *South African Computer Lecturers Association (SACLA) conference*. June 1999. Golden Gate, South Africa.

Cecchet, E., Chanda, A., Elnikety, S., Marguerite, J. & Zwaenepoel, W. (2003). Performance comparison of middleware architectures for generating dynamic web content. *Lecture Notes in Computer Science: Middleware, 2672*, 242-261.

Cooper, R. (2001). Software for managing web sites. *South African Institute of Computer Scientists and Information Technologists (SAICSIT) Annual conference*. September 2001. Pretoria, South Africa.

Creswell, J. W. (2003). *Research design: Qualitative, quantitative and mixed methods approaches* (2nd ed.). USA: Sage Publications.

Dehinbo, J. (2004a). The impact of web-based middleware systems on training and assessment through inhouse developed online testing system. *Proceedings of the Informing Science + Technology Education (InSITE) conference*, Rockhampton, Australia.

Dehinbo, J. (2004b). Determining a suitable programming language for the B.Tech. degree. *Proceedings of the Southern Africa Computer Lecturers Association (SACLA) conference*, Durban. South Africa.

Hartman, H. (2001). Tools for dynamic Web sites: ASP vs PHP vs ASP.NET. *Seybold Report Analysing Publishing Technologies*, *1* (12).

Lim, B. L. (2002). Teaching web development technologies: Past, present, and (near) future. *Journal of Information Systems Education, 13* (2), 117-123.

Leedy, P. D. & Ormrod, J. E. (2001). *Practical research: Planning and design* (7th ed.). USA: Merrill Prentice Hall.

Lerdorf, R. & Tatroe, K. (2002). *Programming PHP: Creating dynamic web pages.* USA: O'Reilly & Associates.

Marshak, M. & Levy, H. (2003). Evaluating web user perceived latency using server side measurements. *Computer Communications, 26* (8), 872-887.

Mouton, J. (2001). *How to succeed in your master's & doctoral studies: A South African guide and resource book*. South Africa: Van Schaik.

Prechelt, L. (2000). An empirical comparison of seven programming languages. *Computer, 33* (10), 23-29.

Renaud, K., Lo, J., Bishop, J., Lo, J. van Zyl, P & Worrall, B. (2003). *Algon: A framework for supporting comparison of distributed algorithm performance.* Italy: PNDP.

Singh, S. & Kotze, P. (2002). Towards a framework for e-commerce usability. *Proceedings of the annual research conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT)*. South Africa: PE.

Treese, G. W. & Stewart, L. C. (2003). *Designing systems for Internet commerce* (2nd ed.). USA: Addison-Wesley.

White, C. J. (2003). *Research: An introduction for educators*. South Africa: Pierre Van Ryneveld.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S. & Corritore, C.L. (1999). A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers, 11* (3), 252-282.

Biography

Johnson Dehinbo is a senior lecturer at the Department of Computer Studies, Tshwane University of Technology, Pretoria, South Africa (recent merging of Technikon Pretoria, Technikon Northern Gauteng, and Technikon North West, to form a dynamic University of Technology). Mr Dehinbo joined the Technikon Northern Gauteng as a lecturer in 1997. Mr Dehinbo has previously worked as a Computer Programmer / Analyst at the International Institute of Tropical Agriculture, Ibadan, Nigeria from 1991 to 1996, and as a Graduate Assistant at the Ogun State University, Ago-Iwoye, Nigeria from 1990 to 1991.

His area of research interests includes web-based application development, database, e-commerce and their impact on educational systems.