Proceedings of the Southern Conference on Computing
The University of Southern Mississippi, October 26-28, 2000

Performance Comparison of Alternative Solutions For Web-To-Database Applications

AMANDA W. WU*, HAIBO WANG# AND DAWN WILKINS+

Abstract. Currently, there are many approaches on developing web-to-database projects. These approaches operate in a very similar way on handling database operations, but result in different performance. In this paper, three alternative approaches (CGI, Java Servlets and PHP) were applied on a same project to investigate their performance difference. Benchmark results will be provided for comparison.

1. Introduction

Common Gateway Interface (CGI) is a traditional way to handle web-to-database applications. The performance of CGI is stable and it has been a reasonable choice to develop web projects. The main drawback of the CGI technique is overall inefficiency in handling concurrent client requests [8]. This downside of CGI limits its usage in many web applications. As an alternative solution, Professional Home Page (PHP) can be built as Apache modules and run inside Apache instead of implementing as a CGI solution. The use of modules improves performance by adding multithreading and persistent database connection features. On the other hand, Java Servlet approach has become a popular solution for e-commerce applications. Servlets have rich features that provide strong influence on tuning the performance of web applications. Keeping database connections persistent and multithreading are the important features of the Java servlet solution. Among these three approaches, PHP has become more and more attractive on small to medium sized web applications. Compared with the Java Servlet solution, it is easier to develop. Compared with the CGI solution, it gains better performance. We were unable to locate any benchmark examples to clarify their differences in performance. This paper considers the details of how these approaches differ on handling multiple concurrent users, and investigates their performance differences by benchmarking various versions of a sample web-to-database program.

2. Theoretical Background

Benchmark experiments were implemented on three alternative solutions: the CGI solution, the Java servlet solution and the PHP3 solution. Both non-persistent and persistent database connection models have been tested on the servlet solution and the PHP3 solution.

Due to the fact that the majority of CGI programs are written in Perl [7], in this paper Perl was used to develop the CGI sample program. Perl stands for "Practical Extraction and Report Language". It was originally developed by Larry Wall to handle large data sets and to create reports. With its roots in data processing, Perl's file handling is advanced and it makes manipulating large files a trivial task [8].

2.1 CGI Life Cycle

2.1.1 What is CGI and what is a CGI program?

It is well known that CGI stands for Common Gateway Interface. Many languages can be used to develop CGI programs, such as C, C++, Java, Perl, Visual Basic, PHP, etc. When these languages are implemented as CGI solutions, their programs will have the extension ".cgi", and will be placed under some special directory typically called "cgi-bin". These programs communicate with web servers via their special library files. Those library files act as parsers to translate their coding information into a format that web servers can understand.

2.1.2 How does a CGI program operate?

It is also not a secret that CGI's life cycle is an obvious cause of its performance problem. A CGI program needs to create a separate process for each user request, and this process will be terminated as soon as the data transfer is completed. Spawning a separate program instance for each client request takes extra time. The operating system has to load the program, allocate memory for the program, and then de-allocate and unload the program from memory [8]. While the operating system is performing housekeeping, nothing else can run. This heavy weight context switch prevents CGI programs from improving their efficiency on handling concurrent client requests, thus they are not suitable for high-traffic applications that need to process a large number of client requests.

---

* MCSR, University of Mississippi, University, MS 38677, ww1@olemiss.edu
# MCSR, University of Mississippi, University, MS 38677, chwang@olemiss.edu
+ Department of Computer Science, University of Mississippi, University, MS 38677, dwilkins@cs.olemiss.edu

2.2 Servlet Life Cycle

2.2.1 What are Java Servlets?

Java Servlets are server side Java code run in a server application to answer client requests. Servlets are not tied to a specific client-server protocol but they are most commonly used with HTTP and the word "Servlet" often refers to "HTTP Servlet".

Servlets make use of the Java standard extension classes in the packages *javax.servlet* and *javax.servlet.http*. Since servlets are written in the highly portable Java language and follow a standard framework, they provide a means to create sophisticated server extensions in a server and operating system independent way.

2.2.2 How does the life cycle of a servlet differ from the life cycle of a CGI program?

When a servlet program is called the very first time, it is loaded into the memory. After the request is processed, the servlet remains in memory and will not be unloaded from memory until the web server is shut down [8]. Normally, a web server will be running all the time. So after a servlet is activated, it keeps running in the background and lives as long as the web server lives. We use the term "Cold Servlet" to refer to a servlet that has not already been loaded into memory, and "Hot servlet" to refer to one already loaded.

The first time a servlet is called and loaded into memory, the init() method of the servlet is called. This method is not executed for future requests to the servlet since then it is "hot". Thus, subsequent requests to the servlet can simply result in method call to the servlet class without the need to reload the program once again. Implementing servlets as Java classes has the benefit that they can easily remain in memory and be efficiently called upon time and time again.

Most web servers provide an option to load some frequently used servlet programs when the web server is started up in order to gain a performance advantage (see Figure 1). In this way, servlets may already be in the memory for the first client request, thus first time access overhead can be avoided. Loading the servlets at start up ensures that response time for all requests is kept to a minimum, as opposed to waiting for the servlet to be loaded on the initial request [8].

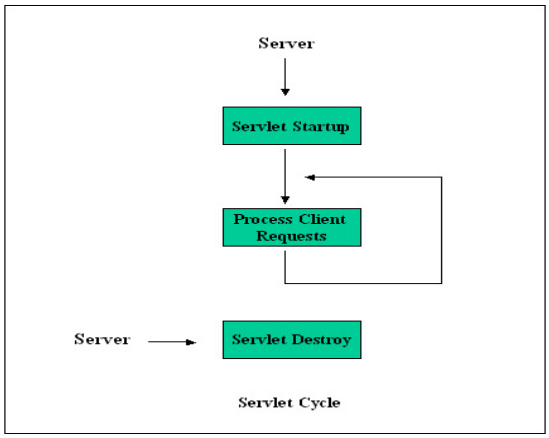

Figure 1. Servlet life cycle

2.2.3 Do all servlet programs perform better than CGI programs?

No. There are always exceptions to the rules. Cold servlets actually perform worse than CGI. Some server-side include servlets are good examples. They will be unloaded from memory immediately after completing a client request [8]. In this case, a servlet is loading and unloading on demand, this reduces a servlet to a CGI model with even worse performance than CGI due to the start up overhead. When a servlet is activated, it calls the init( ) method to prepare all the parameters it needs. It also needs to load the Java Virtual Machine (JVM) in order to get the run time environment. All this is overhead. Cold servlets follows these procedures every time when they are called, since JVM will time out after a certain period depending on the web server settings. But CGI programs only need to load an external interpreter in order to execute, they will not have so much overhead to worry about.

2.2.4 Do we need cold servlets?

Usually, cold servlets are impractical and not recommended since time will be lost while loading and unloading servlets, which makes them less different from the CGI solutions. The problem is, once a servlet is loaded into the memory, it runs in the background without being noticed. This takes system resources. As more and more servlet programs are activated and remain in the memory, the web server might eventually run out of resources and crash. In this case, to keep the system performance up, we might consider keeping only those servlets expected to be frequently used in the memory.

This would keep the shared memory limit down and prevent inefficient use of the system resources. Some servlets will need to be run only on request to protect system resources, unless a web server has enough resources to support all the servlets running.

2.2.5 Java servlets and high-traffic web applications

After considering the life cycle difference between servlets and CGI, it is not hard to understand why servlets are better at handling a large amount of concurrent user requests. E-commerce applications such as online shopping applications will achieve a performance gain with the use of Java servlets.

Multithreading is the main advantage of servlet solutions over CGI solutions. A servlet does not need to use a separate process to handle each client request, it simply spawns another thread within the same process. This makes it possible for hundreds and even thousands of users to access the same servlet simultaneously without bringing down the server [8].

2.2.6 Java servlets can even provide persistent database connection

The servlet life cycle allows for extremely fast database access as well. Most web applications are database driven. A database is an excellent choice to manage data resources. Unfortunately, database operations are the most expensive methods used in web programs. A performance bottleneck often comes right at the beginning when a database connection is opened [5]. CGI programs do not have a choice except to open and close a database connection every time they process a request. No exceptions. Servlets are smarter, they recycle database connections by using a connection pool to avoid the overheads. When a servlet is activated, it can request several database connections in the init() method call, and keep those connections in the connection pool. When a request comes in, the servlet checks to see if there is a connection available. If there is one available, it assigns the connection to the request. If it runs out of connections, it simply creates one more and assigns it to the request. When the request has been processed, it returns the connection to the pool and this connection will be available for the next incoming request. In this way, servlets provide their own way to keep database connections persistent.

A database connection can be recycled and kept persistent by Java servlets as long as the servlet is running and the database connection does not time-out. Database connections can be timed-out after a certain time limit depending on different database server configurations. For MySQL which was used in this project, the database connection time-out limit was set to 8 hours (28800 seconds), which also is the default time-out value.

For high-traffic systems, it is very advantageous to create a large number of connection objects during servlet initialization and then dynamically assign incoming requests to open connection objects [3]. Although persistent connections do not provide any faster data transfer speed than non-persistent connections, they do provide faster database connect speed and result in a substantial savings in processing overhead. This is an important feature of servlet tuning for web-to-database application performance.

2.3 PHP3

PHP is a server-side, cross-platform, HTML embedded scripting language. It was developed late in 1994 by Rasmus Lerdorf to keep track of visitors to his on-line resume. The first public version was known as the Personal Home Page Tools (PHP Tools). It consisted of a simple interpreter engine that understood a few special macros and a number of common utilities. The interpreter was revised later and turned into an open source and renamed as Professional Home Page (PHP). Today, PHP3 ships with a number of commercial products such as Stronghold web server and RedHat Linux. It has become a very attractive tool for web-to-database application development.

PHP provides a programming approach similar to Active Server Page (ASP), but its broad support of databases gives it an edge over VBScript. PHP code can be embedded directly into a HTML page and executes on the server. PHP can also adapt Object Oriented Programming approach to create large PHP projects [2]. PHP can either be built as an Apache module or as a binary that can run as a CGI. It combines with Apache web server and MySQL and becomes a current popular web project solution.

2.3.1 PHP3 built as an Apache module

When built as an Apache module, PHP is especially lightweight and speedy. Without any process creation overhead, it can return results quickly. It offers excellent connectivity to many databases. It can even provide persistent database connections to improve the program efficiency and increase throughput by taking advantage of Apache's multiprocessing feature.

Apache is a multiprocessing web server, which uses the parent process to coordinate its child processes on handling requests. When a request comes in, the parent process routes the request to one of its available child processes, and the request will be serviced by this child process. Different child processes might service subsequent requests from the same

client. Database connections are shared among the child processes and thus kept persistent. For sites that do not need a large, robust web application, Apache and PHP make an effective combination with superior reliability.

2.3.2 PHP3 built as a binary to run as CGI

If PHP3 is implemented as a CGI solution, there will be no gain from a persistent connection. Since no connection persists once the script has finished. A separate instance of the PHP3 interpreter is created and destroyed for every PHP3 page request to the web server. When the instance is destroyed after every request, all allocated resources, such as link to a server, are closed.

3. Experimental Approach

Today's web applications are designed for the purpose of resource sharing. They are expected to have the ability to handle multiple concurrent user requests. When an application is in its design stage, it is very important to choose an appropriate solution to meet the development requirements. Performance is one of the most important issues that needs to be considered.

How much difference does it make when choosing different solutions for an application? In order to answer this question, this project set up a benchmark experiment to investigate the performance differences. In the experiment, a sample program was selected from a web-to-database application developed using CGI solution. This sample program was redeveloped using Java servlets and using PHP3. ApacheBench (ab) [1], which is described in section 3.4, was used to simulate multiple concurrent users making requests on the different versions of the sample program. The time requested to process the request was obtained in order to compare the performance of these alternative approaches.

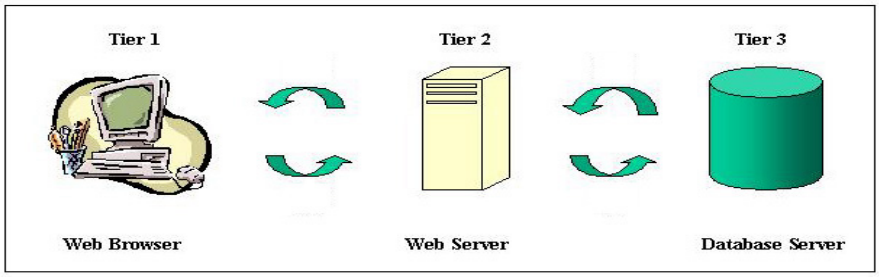

Figure 2. Three-Tier client/server architecture of the "Teacheval" project

This sample CGI program was selected from a web-to-database application called "Online UM Teacher Evaluation Results", known as "Teacheval". The overall design of this application is a 3-tier client/server architecture [6]: web browser (client), web server and backend database server (see Figure 2). The backend database server is hosted by a Sun Enterprise 450 server known as "sumac", the web server is hosted by another Sun Enterprise 450 server known as "cedar". The application is similar to the requirements to many real-world web-to-database applications with a backend database server separated from the web server to ensure that users have no direct access to the data. Client requests will go across the network and reach the web server. The web server communicates with the database server and processes the request and sends information back to the client.

3.1 Database Table Structure

Five tables from the "Teacheval" database were involved in the sample program used in this project: evals, deptinfo, schoolinfo, courses, and terminfo. Figure 3 shows the relationship of these five tables.

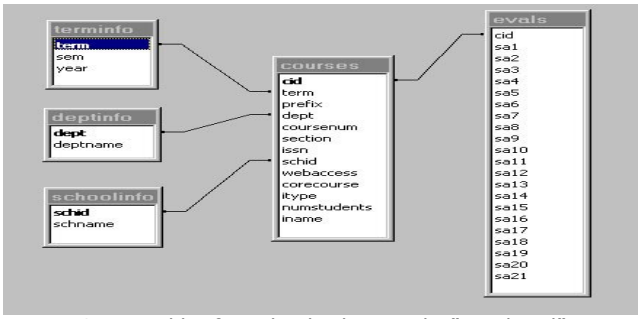

Figure 3. Five tables from the database in the "Teacheval" project

"evals" table: stores 27307 entries of answers to 21 questions, "cid" is a foreign key.
"courses" table: stores 1358 header sheet records, "cid" is the primary key.
"terminfo" table: stores 13 records of term, year and semester information.
"deptinfo" table: stores 92 department ID and department names.
"schoolinfo" table: stores 9 school ID and school names

3.2 Test Platforms

The benchmark experiment was carried on a SGI O2 workstation with one processor, known as "rain". This machine is running apache and IRIX 6.5. There are only a few small unofficial applications running on this web server. Its performance can be considered as in an ideal situation.

3.3 Sample program used in benchmark experiment

The sample program generates the following yellow box (see Figure 4).

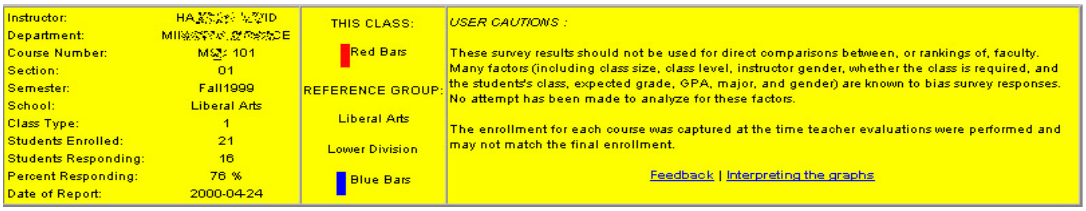

Figure 4. Output of the sample benchmark program

Inside this program, two queries are used to retrieve information from the database: the first simple query randomly selects one course ID from the courses table:

```
$rows = int (rand 1000);
$sql_statement = "SELECT cid FROM courses where term='001' AND  webaccess<>'1' LIMIT $rows, 1";
```

The second query is a complex query, which joins all five tables to select all the information that is needed to fill out the left part of the yellow box:

$sql_statement = "SELECT courses.prefix, courses.coursenum, courses.section, courses.iname, courses.itype, courses.numstudents, count(evals.cid), deptinfo.deptname, terminfo.sem, terminfo.year, schoolinfo.schname FROM courses, deptinfo, terminfo, schoolinfo, evals where courses.cid=|'$vals[0]|' AND deptinfo.dept = courses.dept AND terminfo.term =courses.term AND schoolinfo.schid = courses.schid AND evals.cid = courses.cid GROUP BY courses.cid";

In order to compare the performance of different solutions, this yellow box was implemented as five different versions:

1. ***Version 1:*** CGI with Perl.
2. ***Version 2:*** Java servlet without persistent database connection. In this version, methods of opening database connection and closing database connection were put in the *doGet( )* block. Thus, the database connection will be opened and closed on each request. This operates in this same manner as CGI.
3. ***Version 3:*** Java servlet with one persistent database connection. In this version, opening a database connection was put in the *init( )* block, and closing a database connection method is called only when the web server is shut down. Thus the database connection can be kept persistent for all incoming requests.
4. ***Version 4:*** PHP3 without persistent database connection. This version used *mysql_connect("host", "username", "password")* and *mysql_close( )* as the database connection open and close methods. Thus, the database connection will be opened when a request comes in, and will be closed when the request has been processed. This also operates in the same manner as CGI.
5. ***Version 5:*** PHP3 with persistent database connections. This version used *mysql_pconnect ("host", "username", "password")* as the open database connection method. This similar to Java servlets' connection pool feature. By using this method, PHP3 pools the database connections. When a request is coming in, it first tries to find a persistent connection that is already opened with the same host, username and password. If one is found, an identifier for this existing connection is returned instead of opening a new connection. The connection will not be closed when the execution of the script ends. Instead, it remains open for the next incoming requests.

3.4 Benchmark Tool

These five versions of the test programs have been tested by simulating multiple concurrent users making requests to them using ApacheBench (ab).

Ab is a built-in application in the Apache web server. It is a benchmarking tool that can be used to compare the performance of the different versions of the yellow box. It was designed to test how a web server performs when users are making requests to a supplied program. Ab is very convenient and is very easy to use. In particular, it shows how many requests per second an Apache server is capable of serving to that particular program. If two different versions of the sample program are run on a same web server, then the difference of the web server processing on these two versions shows their performance difference.

Among the results ab provides, the only number we care about is the "requests per second", as long as no requests were unable to be processed by the web server. The usage of ab to benchmark a program is:

ab -n [*number of requests made by each user*] -c [*number of concurrent users*] –w [*completed url of the program*]

In this project, multiple users (from 1 to 50) are simulated concurrently requesting the different versions of sample programs. Each of the simulated users makes 1 request to a sample program in every test.

4. Comparison

4.1 What difference does a persistent database connection make on a web-to-database application?

Four versions (2-4) of the sample program were tested to compare their performance.

4.1.1 Java servlet versions

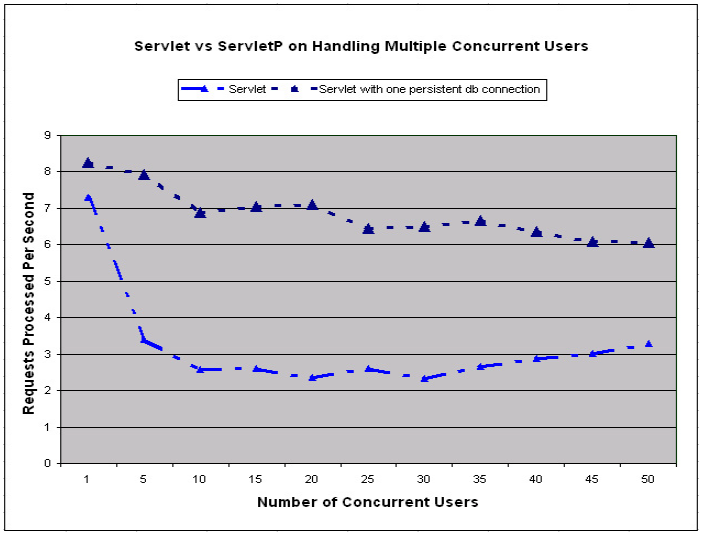

Figure 5. Performance difference between version 2 and version 3

Figure 5 shows the benchmark results of two Java servlet versions. Follow the two curves, we can see how the web server reacts on these two versions of the program when the number of concurrent user increases. The vertical distance shows the performance difference between these two versions.

Version 3 keeps a persistent database connection, and shares it between the incoming requests. Thus the processing speed is relatively stable. As the requests traffic becomes more and more heavy, the number of requests that can be processed per second drops slightly. This performance is gained between these two versions from saving the time of reopening the database connections, which is the most time-consuming operation.

Version 2 operates in an open-transfer-close manner. Thus a small increase of traffic load results in greatly decreasing processing speed. The request processing speed seems to be stable in the range of 5 to 30 concurrent requests, and then tends to increase slightly. This is an interesting result that might be caused by the cache feature of servlets. One of the features of a servlet that can be used to tune the performance is to use cache. Sometimes the cache is so large the web server has to be restarted to get a fresh copy of the servlet output.

4.1.2 PHP3 versions

From Figure 6, it seems there is not much difference on the processing speed between these two versions in the cases of light traffic and very heavy traffic. In the beginning, almost no difference can be seen between two versions. Since both

of the versions need to open a connection on the first request. When the number of concurrent requests increases to the range of 5-40, having a persistent database connection shows its difference in performance. But the performance of both versions tends to decrease when the number of concurrent users exceeds 40. PHP3 is an interpreted language, thus it has limitations on execution speed.

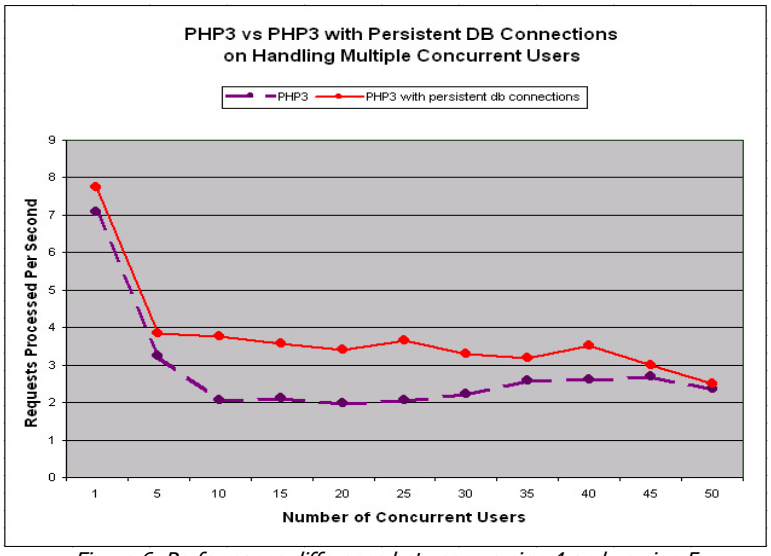

Figure 6. Performance difference between version 4 and version 5

4.2 How bad is a CGI compared to a servlet program?

Figure 7 shows the benchmark results between Java servlets (version 2 and version 3) and CGI (version 1) on handling multiple concurrent users. Notice that all three versions give better throughput when handling small number of concurrent users making requests due to the lighter load of the web server. At this time, the web server has enough ability and resources to handle small number of requests, and processes the requests very fast. When the number of concurrent users goes up, the web server becomes more heavily loaded. Processing time increases significantly.

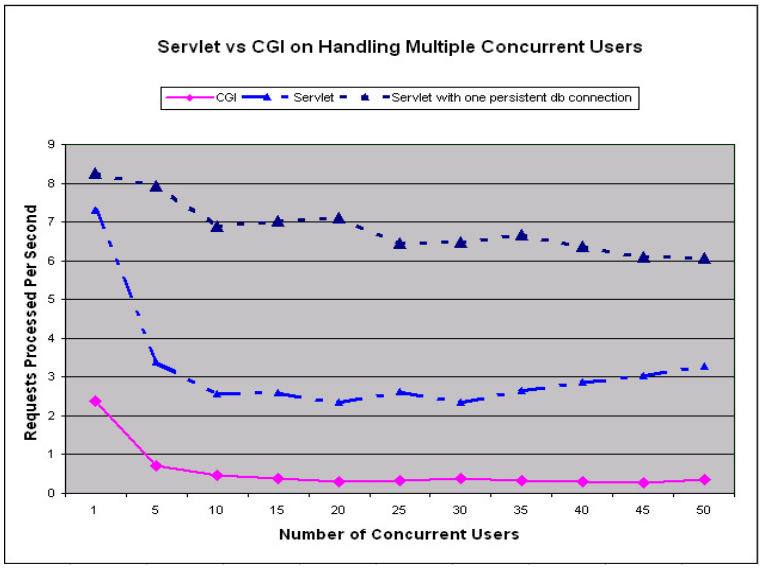

Figure 7. Performance difference between serlvets and CGI tested on "rain"

The CGI version performs better only when handling less than five concurrent users. When the number of concurrent users increases, the performance degrades.

In both tests, it is apparent that there is always a large processing speed difference between the servlet versions and the CGI version. So we can easily draw a conclusion that Java servlet is a much better solution than CGI.

4.3 Does PHP3 really qualify as a better alternative solution to CGI?

The PHP3 interpreter can be run as a web-server module under Apache. This makes a significant different when compared to a CGI solution. Running under Apache as an Apache module will be faster than running an external process to the interpreter code. This contrasts with many other interpreted languages where the entire interpreter is loaded, executed, and then terminated every time a browser accesses a web page containing the code on the server.

Figure 8 shows the difference in processing speed of the three versions (CGI, PHP3 without persistent database connections and PHP3 with persistent database connections). Although PHP3 versions do not show as much difference as the servlet versions do, the PHP3 versions are still significantly better than the CGI version.

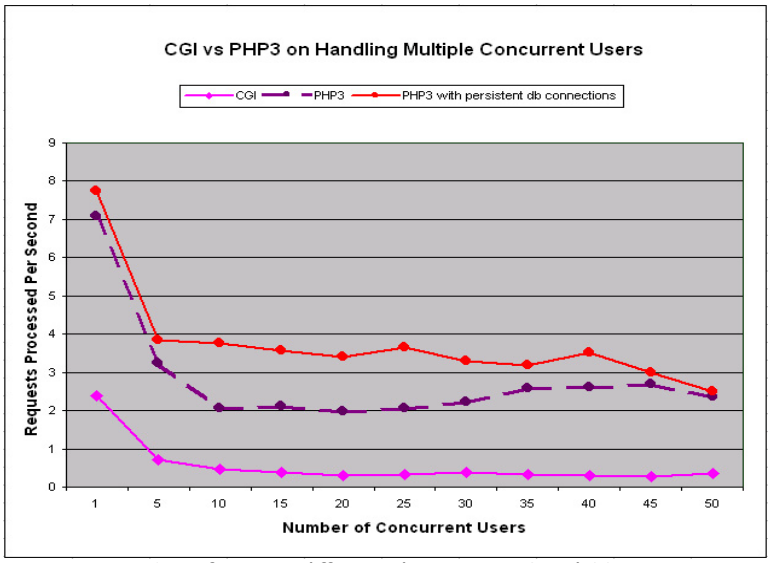

Figure 10. Performance difference between PHP3 and CGI versions

Both PHP3 and Perl are interpreted languages, but the difference is, the CGI version has to load the interpreter engine into web server, and then unloads it when the program exits. But PHP3 code is embedded directly into HTML code, and run inside Apache, thus it performs much better than CGI.

4.4 Between servlet and PHP3, which one is better?

Both servlet and PHP3 are considered better alternative solutions to CGI based on performance. Which one is actually better? Since servlets are server side Java code. Java is "compiled" to bytecode and then interpreted by JVM. While PHP3 is an interpreted language, and thus the servlet version should run faster than the PHP3 version. To discover the real difference, benchmark tests were run on versions 2 through 5.

4.4.1 Two solutions using non-persistent database connection model

In this case, both solutions have to reopen and close database connections. From Figure 19, we can see the two performance curves are almost overlapping for a while, and then begin to show their difference when the request traffic reaches a large number (30 in this case) with the servlet version increasing and the PHP3 version decreasing on the processing speed.

4.4.2 Two solutions using persistent database connection model

If the persistent database model is applied on both solutions, which one will benefit more? Figure 10 tells it all.
When handling the light traffic cases, the servlet version has some overhead that reduces its advance compared to the PHP3 version. Since a servlet has to load JVM in order to get a run time environment. JVM is considered an external process to Apache. It will take extra system resources to load it. JVM can be timed out in a certain time interval depending on the web server setting. This will decrease the processing speed a little bit. While PHP3 runs as an Apache module, which can be preloaded into memory when the server is started. Server children will be able to share the code space used by these modules. This virtually eliminates or reduces the difference between these two versions. In very heavy traffic cases, PHP3 version is getting close to its limits.

When the persistent database connection model is applied, the servlet version starts to show it advantage over PHP3. The more heavily loaded the situation, the more advantageous the servlet appears. This difference become significant when the traffic load becomes large.

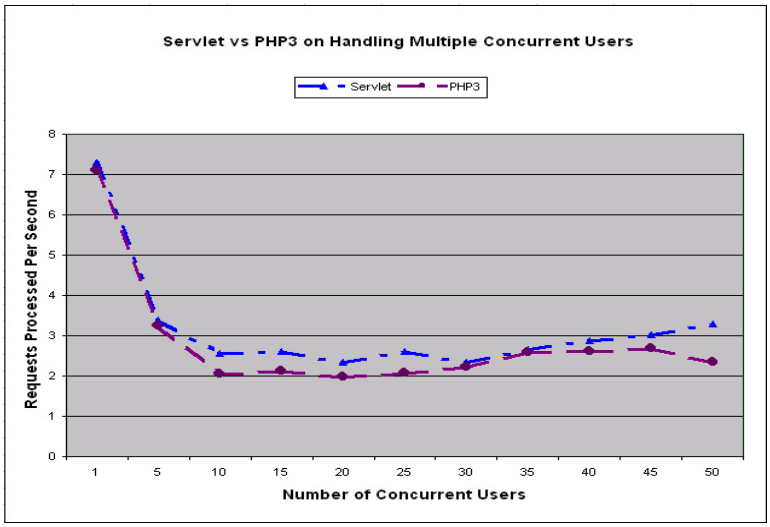

Figure 9. Performance comparison of servlet and PHP3 using non-persistent database connection model

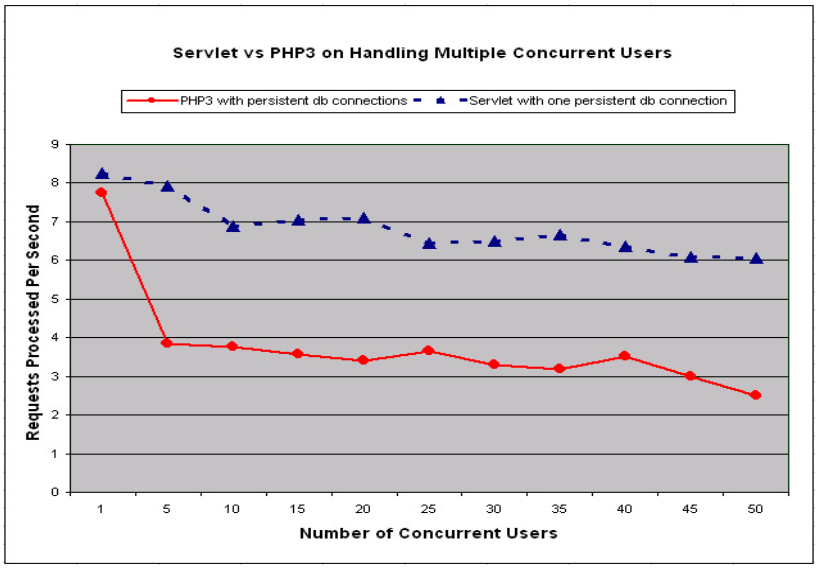

Figure 10. Performance comparison of servlet and PHP3 using persistent database connection model

4.5 Which version is the best solution?

Putting all the benchmark results of five versions together, Figure 11 shows a big picture of performance difference on these different approaches.

The solution of Java servlet with persistent database connection has the best performance, while PHP3 using persistent database connections performs fairly well when compared to the CGI solution.

Although the non-persistent database connection models using Java servlet or PHP3 pay some performance penalty, they still shows an advantage over the CGI solution. It is quite obvious that the CGI solution lacks efficiency when trying to handle high-traffic applications.

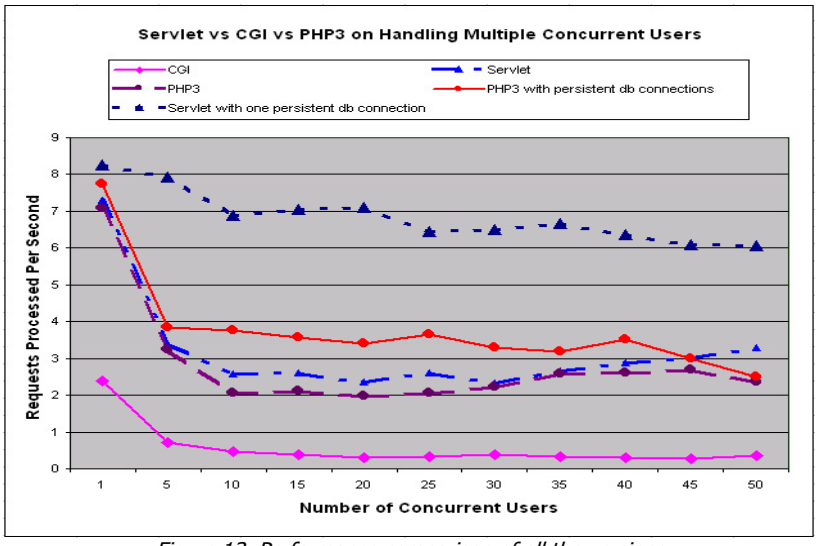

Figure 13. Performance comparison of all the versions

5. Discussion

Java servlets have the advantage of Java's Object Oriented nature, thus they begin to show their important role on web application projects. With their multi-threading feature, Java servlets are an excellent choice to meet the requirement of e-commerce (such as online shopping) applications. These applications need to handle client requests in a highly interactive mode and they must process multiple client requests in a fast and efficient way.

In this project, benchmark tests were only run on data retrieval (read) operations. Due to the fact that the database allows multiple read operations at the same time, a single-thread model was not applied in this project in order to provide a fair comparison base for all versions. If single-thread model is applied on the Java servlet solution, the processing speed would decrease due to the requested synchronization. Currently, there is no such single-thread model concept for the CGI solution and PHP3 solution, which makes the Java servlet solution one more step advanced than the other two solutions. Having the ability to operate in a thread safe mode is important for database updating.

The combination of PHP, Apache and MySQL is very attractive for current web-to-database projects. For these applications, performance is not a critical issue. Using PHP3, a web project can be developed in a quick and dirty way with the benefit of easy coding and better performance than CGI. But the tradeoff is, the current version of PHP uses different database functions and methods for handling different databases, which makes it harder to switch database system. Java servlets can use JDBC to overcome this restriction. In addition, CGI also can solve this problem by using a generic DBI.

CGI solutions are appropriate for small applications with a limited amount of client access. These low-traffic applications can be developed using CGI solutions quickly with the tradeoff being a the performance penalty. In the situation of processing a huge amount of raw data, many direct system calls must be made, and in these cases CGI with Perl may show its power. This is also one of the reasons why mod_perl comes along to make use of the advantages of Perl, and to avoid the disadvantages of being run as a CGI.

References

[1] *" http://www.sgi.org"*

[2] Argerich, L., "Object Oriented Programming in PHP: The way to large PHP projects*"* *http://www.phpbuilder.com/columns/luis20000420.php3*

[3] Ayers, D., Bergsten, H., Bogovich, M., etc., "Professional Java Server Programming" 176, (1999).

[4] Gundavaram, S., "CGI Programming on the World Wide Web" 6, (1996).

[5] Hunter, J., Crawford, W., "Java Servlet Programming" 259, (1998).

[6] Myers, T., Nakhimovsky, A., "Professional Java XML Programming with Servlets and JSP" 11, (1999).

[7] Patchett, C., Wright, M., "The CGI/Perl Cookbook" 16, (1998).

[8] Williamson, A. R., "Java Servlet by Example" 4-20, (1999)