A Comparison of Software Architectures for E-business Applications

Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite and Willy Zwaenepoel
Department of Computer Science
Rice University

Abstract

As dynamic content has become more prevalent on the Web, a number of standard mechanisms have evolved to generate such dynamic content. We study three specific mechanisms in common use: PHP, Java servlets, and Enterprise Java Beans (EJB). PHP and Java servlets require a direct encoding of the database queries in the application logic. EJB provides a level of indirection, allowing the application logic to call bean methods that then perform database queries. Unlike PHP, which typically executes on the same machine as the Web server, Java servlets and EJB allow the application logic to execute on different machines, including the machine on which the database executes or a completely separate (set of) machine(s).

We present a comparison of the performance of these three systems in different configurations for two application benchmarks: an auction site and an online bookstore. We choose these two applications because they impose vastly different loads on the sub-systems: the auction site stresses the Web server front-end while the online bookstore stresses the database. We use open-source software in common use in all of our experiments (the Apache Web server, Tomcat servlet server, Jonas EJB server, and MySQL relational database).

The computational demands of Java servlets are modestly higher than those of PHP. The ability, however, of locating the servlets on a machine different from the Web server results in better performance for Java servlets than for PHP in the case that the application imposes a significant load on the front-end Web server.

The computational demands of EJB are much higher than those of PHP and Java servlets. As with Java servlets, we can alleviate EJB's performance problems by putting them on a separate machine, but the resulting overall performance remains inferior to that of the other two systems.

Rice University Computer Science Technical Report TR02-389

1. Introduction

Web content is increasingly generated dynamically, a departure from the early days of the Web when virtually all content consisted of static HTML or image files. Dynamic Web content is typically generated by a front-end Web server and a back-end database (see figure 1). The (dynamic) content of the site is stored in the database. The application logic provides access to that content. The client sends an HTTP request to the Web server containing the appropriate URL and some parameters. The Web server causes the application logic to be executed. Typically, the application logic issues a number of queries to the database and formats the results as an HTML page. The Web server finally returns this page as an HTTP response to the client.

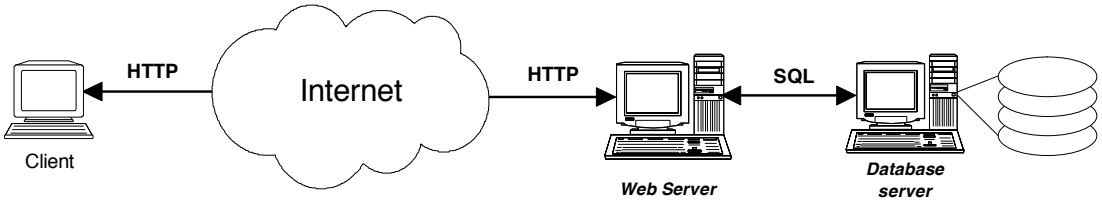

Figure 1. Typical Configuration of a Dynamic Content Web Site

The application logic may take various forms, including scripting languages such as PHP [13], that execute as a module in the Apache Web server [3], Microsoft Active Server pages [10] that are integrated with Microsoft's IIS server [11], and Java-based systems that execute in a separate Java virtual machine. We concentrate here on three software systems for writing application content: PHP, Java servlets [6], and Java servlets used in combination with Enterprise Java Beans (EJB) [1].

PHP is a scripting language in which SQL queries can be embedded. Similarly, Java servlets allow SQL queries to be embedded in Java code. In both PHP and Java servlets, the application programmer writes the SQL queries. The situation is different with EJB. A number of beans are defined, which, roughly speaking, represent items in the database. The Java servlets call bean methods, which then in turn issue SQL queries to the database. EJB therefore provides a level of indirection between the application and the database. PHP executes as a module in the Web server. The two Java-based systems execute in a separate Java virtual machine. This allows an extra degree of freedom in configuring the system: the servlets can be located on a machine different from the Web server machine, either on the database machine or on a separate machine.

Although the computational demands of Java servlets are higher than those of the corresponding PHP scripts, we demonstrate that this extra degree of freedom can be used to improve the performance of Java servlets compared to PHP. In particular, we show that for applications that put significant load on the front-end Web server, better performance can be achieved by locating Java servlets on the database machine or on a separate machine. The introduction of EJB, however, adds a degree of inefficiency that cannot be overcome by putting the servlets or the EJBs on a separate machine.

We use two benchmarks to demonstrate these results, an auction site modeled after eBay.com [5] and an online bookstore modeled after TPC-W [20]. We use a 1.33GHz AMD Athlon with 768MB memory and a 60GB disk for each machine. The machines are connected to each other and to a set of machines running client emulation software by a switched 100Mbps Ethernet. The auction site puts more load on the Web server, while for the online bookstore the database CPU is the bottleneck. In all but one of the experiments we find the memory, the disk or the network not to be a bottleneck.

The outline of the rest of this paper is as follows. Section 2 provides some background on PHP, Java servlets, and EJB. Section 3 describes the two benchmarks that we use to evaluate these systems. Section 4 describes our experimental environment and our measurement methodology. Sections 5 and 6 discuss the results of our experiments with the auction site and the online bookstore benchmarks, respectively. Section 7 discusses related work. Section 8 concludes the paper.

2. Background

2.1. PHP (Hypertext Preprocessor)

PHP [13] is a scripting language that can be seen as an extension of the HTML language: PHP code can be directly embedded into an HTML page. Built as an HTTP server module, PHP is executed within a Web server process and does not incur any inter-process communication overhead. When the HTTP server identifies a PHP tag, it invokes the PHP interpreter that executes the script. Requests to the database are performed using an ad hoc interface.

2.2. Java HTTP Servlets

An HTTP servlet [6] is a Java class that can be dynamically loaded by a servlet server and runs in a Java Virtual Machine (JVM). After the initial load, the server invokes the servlet using a lightweight method invocation, but since the JVM is a separate process from the Web server, inter-process communication takes place for each request. Concurrent requests are handled by separate threads. Servlets access the database using the standard JDBC interface, supported by all major databases.

2.3. Enterprise Java Beans

The purpose of an Enterprise Java Beans (EJB [1]) server is to abstract the application business logic from the underlying middleware. There are two types of EJB: entity beans that map data stored in the database (usually one entity bean instance per database table row), and session beans that are used to perform temporary operations (stateless session beans) or represent temporary objects (stateful session beans). The EJB server is responsible for providing services such as database access (JDBC), transactions (JTA), security (JMS), naming (JNDI) or management support (JMX).

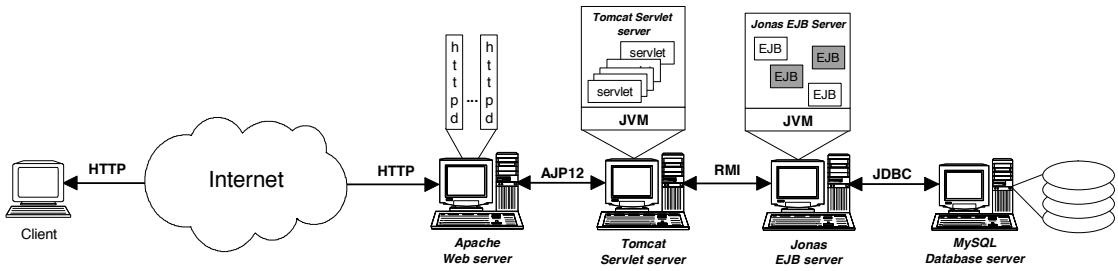

Figure 2. Enterprise Java Bean integration in the e-business architecture

Figure 2 shows an example of an architecture including an EJB server. First, a client sends a request to the HTTP server. The HTTP server invokes the servlet server using a well-defined protocol (AJP12). The servlet queries the EJB server (using RMI) to retrieve the information needed from the database in order to generate the HTML reply. The EJB server calls the database.

2.4. Summary

PHP scripts are easy to write and reasonably efficient, but the database interfaces are ad hoc. This makes code maintenance awkward, because new code needs to be written for each new database to which the scripts need to access. PHP scripts execute in the same process (address space) as the Web server, thereby minimizing communication overheads between the Web server and the scripts.

HTTP Java servlets interface to the database using JDBC. This makes them easily portable between databases. Contrary to the PHP interpreter, the servlet server runs in a JVM as a separate process from the Web server. On the plus side, this implies that it can be placed on a machine different from the Web server to balance the load. On the minus side, servlets incur the overhead of the JVM. In addition, they incur the cost of inter-process communications with the Web server, especially when they execute on a separate machine.

The EJB architecture abstracts the application logic from any specific platform, protocol, or middleware infrastructure. In particular, the bean methods can be called through a Web interface or from an arbitrary program.

3. Benchmarks

We describe the two benchmarks we use to compare PHP, HTTP Java servlets, and EJB. We chose one benchmark that puts significant load on the Web server (an auction site) and another benchmark that puts significant load on the database (an online bookstore).

3.1. Auction Site Benchmark

Our auction site benchmark implements the core functionality of an auction site: selling, browsing and bidding. We do not implement complementary services like instant messaging or newsgroups. We distinguish between three kinds of user sessions: visitor, buyer, and seller. For a visitor session, users need not register but are only allowed to browse. Buyer and seller sessions require registration. In addition to the functionality provided during visitor sessions, during a buyer session users can bid on items and consult a summary of their current bids, rating and comments left by other users. Seller sessions require a fee before a user is allowed to put up an item for sale. An auction starts immediately and lasts typically for no more than a week. The seller can specify a reserve (minimum) price for an item.

The database contains seven tables: users, items, bids, buy_now, comments, categories and regions. The users table records contain the user's name, nickname, password, region, rating and balance. Besides the category and the seller's nickname, the items table contains the name that briefly describes the item and a more extensive description, usually an HTML file. Every bid is stored in the bids table, which includes the seller, the bid, and a max_bid value used by the proxy bidder (a tool that bids automatically on behalf of a user). Items that are directly bought without any auction are stored in the buy_now table. The comments table records comments from one user about another. As an optimization, the number of bids and the amount of the current maximum bid are stored with each item to prevent many expensive lookups of the bids table. This redundant information is necessary to keep an acceptable response time for browsing requests. As users only browse and bid on items that are currently for sale, we split the item table in a new and an old item table. The very vast majority of the requests access the new items table, thus considerably reducing the working set used by the database.

Our auction site defines 26 interactions that can be performed from the client's Web browser. Among the most important ones are browsing items by category or region, bidding, buying or selling items, leaving comments on other users and consulting one's own user page (known as myEbay on eBay [5]). Browsing items also includes consulting the bid history and the seller's

information. We define two workload mixes: a browsing mix made up of only read-only interactions and a bidding mix that includes 15% read-write interactions. The bidding mix is the most representative of an auction site workload.

We sized our system according to some observations found on the eBay Web site. We always have about 33,000 items for sale, distributed among eBay's 40 categories and 62 regions. We keep a history of 500,000 auctions in the old-items table. There is an average of 10 bids per item, or 330,000 entries in the bids table. The buy_now table is small, because less than 10% of the items are sold without auction. The users table has 1 million entries. We assume that users give feedback (comments) for 95% of the transactions. The new and old comments tables contain about 31,500 and 475,000 comments, respectively. The total size of the database, including indices, is 1.4GB.

3.2. Online Bookstore Benchmark

The TPC-W benchmark from the Transaction Processing Council [20] is a transactional Web benchmark specifically designed for evaluating e-commerce systems. Our online bookstore benchmark is modeled after TPC-W. It implements all the functionality specified in TPC-W that has an impact on performance, including transactional consistency and support for secure transactions. It does not implement some functionality specified in TPC-W that has an impact only on price and not on performance, such as the requirement to provide enough storage for 180 days of operation.

All persistent data, with the exception of the images used with each book, is stored in the database. The database contains eight tables: customers, address, orders, order_line, credit_info, items, authors, and countries. The order_line, orders and credit_info tables store information about orders that have been placed, in particular: the book ordered, the quantity and discount (table order_line), the customer identifier, date of order, information about the amount paid, shipping address and status (table orders), and credit card information such as type, number and expiry date (table credit_info). The items and authors tables contain information about the books and their authors. Customer information, including real name and user name, contact information (email, address), and password, is obtained via a customer registration form and maintained in the customers and address tables.

The inventory images, totaling 183 MB of data, are resident on the Web server. We implemented the 14 different interactions specified in the TPC-W benchmark specification. Of the 14 scripts, 6 are read-only, while 8 cause the database to be updated. The read-only interactions include access to the home page, listing of new products and best sellers, requests for product detail, and two interactions involving searches. Read-write interactions include user registration, updates of the shopping cart, two interactions involving purchases, two involving order inquiry and display, and two involving administrative tasks. We use the same distribution of script execution as specified in TPC-W. An interaction may also involve requests for multiple embedded images, each image corresponding to an item in the inventory. With one exception, all interactions query the database server.

We implement a Payment Gateway Emulator (PGE) that represents an external system that authorizes payment of funds during purchasing interactions [20, clause 6.4]. The Web server contacts the PGE using an SSL session to send the credit card information. The PGE replies with a message containing the authorization number. The PGE is not part of the benchmarked system.

TPC-W specifies three different workload mixes, differing in the ratio of read-only to read-write scripts. The browsing mix contains 95% read-only scripts, the shopping mix 80%, and the ordering mix 50%. We have experimented with two different database sizes, 350MB and 3.5GB.

4. Hardware and Software Environment

4.1. Client Emulation Implementation

We implement a client-browser emulator. A session is a sequence of interactions for the same customer. For each customer session, the client emulator opens a persistent HTTP connection to the Web server and closes it at the end of the session. Each emulated client waits for a certain think time before initiating the next interaction. The next interaction is determined by a state transition matrix that specifies the probability to go from one interaction to another one.

The think time and session time for all benchmarks are generated from a negative exponential distribution with a mean of 7 seconds and 15 minutes, respectively. These numbers conform to clauses 5.3.1.1 and 6.2.1.2 of the TPC-W v1.65 specification. We vary the load on the site by varying the number of clients. We have verified that in none of the experiments the clients are the bottleneck.

4.2. Application Logic Implementation

In PHP and Java servlets, the application programmer is responsible for writing the SQL queries. In order to arrive at a fair comparison, we use exactly the same queries to the database in both environments

EJB containers automatically manage bean persistence, relieving the programmer of writing SQL code. We use these facilities to implement entity beans with container-managed persistence. However, as we use the EJB 1.1 CMP model, we use session beans for complex queries that require joins on several database tables. The Java servlets are used only as the presentation tier as defined in [1], to generate the HTML reply from the information retrieved from the bean.

4.3. Software Environment

We use Apache v.1.3.22 as the Web server, configured with the PHP v.4.0.6 module, mod_ssl version 2.8.5 and openSSL 0.9.5a. We increase the maximum number of Apache processes to 512. We observe that with that value, the number of Apache processes is never a limit on performance.

The servlet server is Jakarta Tomcat v3.2.4 [19], running on Sun JDK 1.3.1. The EJB server is JOnAS v2.4 [7], an Open Source Java implementation of the EJB specification. We use this EJB server because, to the best of our knowledge, it is the fastest open source implementation, and its performance has been found to be comparable to some commercial implementations [14]. JOnAS is also integrated in Lutris Enhydra Application Server [9] and Libelis Orcas [8].

We use MySQL v.3.23.43-max [12] as our database server with the MyISAM non-transactional tables. The MM-MySQL v2.04 type 4 JDBC driver is used for both the servlet and EJB servers.

All machines run the 2.4.12 Linux kernel.

4.4. Hardware Platform

The Web server and the database server run on an AMD Athlon 1.33GHz CPU with 768MB SDRAM, and a Maxtor 60GB 5,400rpm disk drive. A number of 800MHz AMD Athlon machines run the client emulation software. We use enough client emulation machines to make sure that the clients do not become a bottleneck in any of our experiments. All machines are connected through a switched 100Mbps Ethernet LAN.

4.5. Measurement Methodology

Each experiment is composed of 3 phases. A warm-up phase initializes the system until it reaches a steady-state throughput level. We then switch to the steady-state phase during which we perform all our measurements. Finally, a cool-down phase slows down the incoming request flow until the end of the experiment. For all experiments with a particular application we use the same length of time for each phase, but the duration of each phase is different for different applications. The online bookstore uses 1 minute, 10 minutes and 10 seconds for the warm-up, the steady-state and the cool-down phase, respectively. The auction site uses 5, 30 and 5 minutes. These lengths of time were chosen based on observation of when the experiment reaches a steady state, and the length of time necessary to obtain reproducible results.

To measure the load on each machine, we use the *Sysstat* utility [18] that collects CPU, memory, network and disk usage from the Linux kernel every second. The resulting data files are analyzed post-mortem to minimize system perturbation during the experiments.

4.6. Configurations

We use a total of five configurations to evaluate the different e-business software architectures defined in Section 2. (1) PHP running on the same machine as the Web server, (2) Java servlets running on the same machine as the Web server, (3) Java servlets running on the same machine as the database, (4) Java servlets running on a dedicated machine, and (5) Web server, servlet server, EJB server, and database server each running on different machines.

5. Experimental Results for Auction Site

5.1. Configurations using PHP and Java Servlets

Figure 3 reports the throughput in interactions per minute as a function of number of clients for the browsing mix workload, using PHP and Java servlets, respectively. Running the Java servlets on the Web server machine results in the lowest performance, with a peak throughput of 6,840 interactions per minute, reached for 700 clients. PHP peaks at 8,520 interactions per minute with 800 clients. Even better results, however, are achieved by running the Java servlets on the database machine or on a dedicated machine. These configurations result in 10,200 and 12,000 interactions per minute, for 1,000 and 1,200 clients, respectively.

Figure 5 shows the throughput in interactions per minute as a function of number of clients for the bidding mix, using PHP and Java servlets. The ordering of the different configurations in terms of performance is different. In particular, putting Java servlets on the database machine is now the worst performer, reaching a peak of 6,480 interactions per minute with 700 clients. Servlets on the Web server achieves a peak 7,380 interactions per minute with 700 clients. PHP peaks at 9,780 clients per second with 1,100 clients. The best configuration remains the one in which the servlets run on a dedicated machine, with 10,440 interactions per minute at 1,200 clients.

These results can be explained by looking at figure 4 and figure 6 that report the CPU utilization for each of the machines in the various configurations, for the browsing and the bidding mix, respectively. Let us first compare the configurations in which the application logic, whether implemented by PHP or by Java servlets, runs on the Web server. For both the browsing and the bidding mix, the Web server CPU is the bottleneck with 100% CPU utilization. PHP is more efficient than Java servlets for this application. We attribute this difference in part to the extra communication between the Web server and the servlet server that, unlike PHP, execute in separate processes. While the database machine CPU is not a bottleneck in these configurations, its utilization is substantially lower for the browsing mix than for the bidding mix. In particular, for the Java servlets configuration, it is 21% for the browsing mix and 45% for the bidding mix.

This difference explains why moving the servlets to the database machine results in substantially better performance for the browsing mix than for the bidding mix. The third set of bars in figure 4 and figure 6 shows that, for both mixes, when the servlets execute on the database machine, the database machine CPU becomes 100% utilized. The resulting bottleneck is, however, much more severe for the bidding mix than for the browsing mix. This can be derived from the fact that the Web server CPU utilization is much higher for the browsing mix (81%) than for the bidding mix (34%). The configuration with the servlets executing on the same machine as the database leads to a relatively well balanced CPU load on the two machines for the browsing mix, and therefore good performance, but a poorly balanced situation for the bidding mix, and therefore inferior performance. Finally, when a dedicated machine is used for the servlets, the best performance is achieved for both mixes. The benefit of an extra CPU outweighs the extra communication costs resulting from putting the servlet server on a separate machine. Communication between the servlet server and the database is modest at an average of 1.8Mb/s.

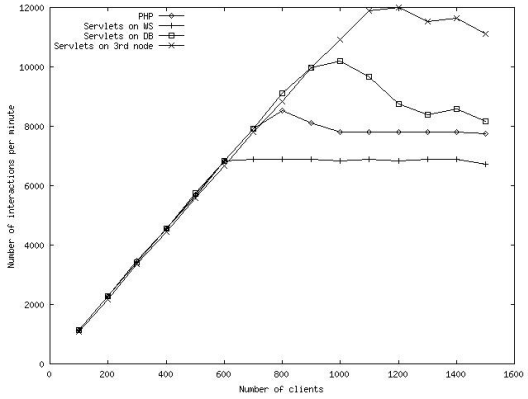

Figure 3. Auction site throughput in interactions per minute as a function of number of clients for the browsing mix using PHP and Java servlets.

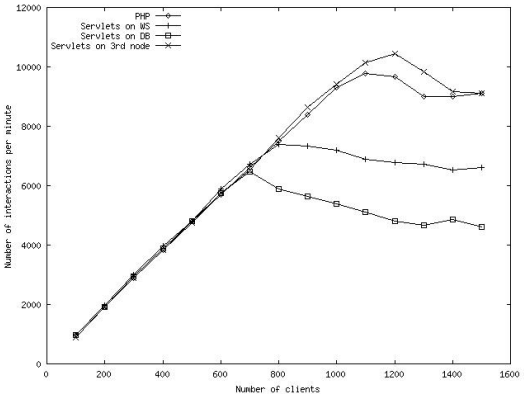

Figure 5. Auction site throughput in interactions per minute as a function of number of clients for the bidding mix using PHP and Java servlets.

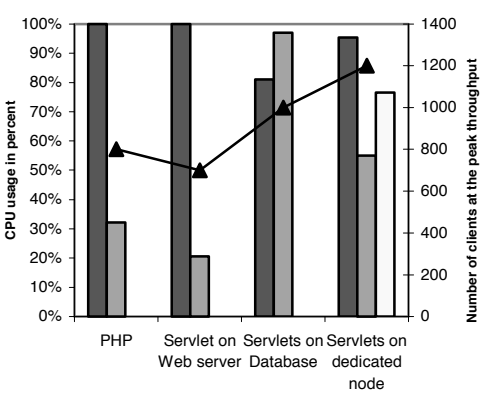

Figure 4. Number of clients and processor usage in percent at the peak throughput for the browsing mix using PHP and Java servlets.

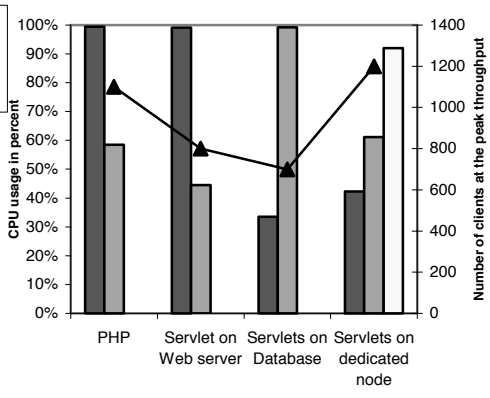

Figure 6. Number of clients and processor usage in percent at the peak throughput for the bidding mix using PHP and Java servlets.

With one exception, none of the other resources (memory, disk, network) are the bottleneck for any of these configurations. The one exception is the case of the browsing mix in the configuration with a dedicated machine for the servlet server. In this configuration the network traffic on the Web server reaches 94Mb/s (80Mb/s to clients and 14Mb/s from the servlet server).

Memory and disk usage are never the bottleneck in any of these experiments. For instance, for the bidding mix, which has the highest memory and disk requirements, we observe a maximum memory usage of 110MB, 95MB, and 390MB on the Web server, the servlet server and the database server, respectively. Although the database is much larger than that, most accesses are to records relating to new auctions, which is a small subset. Disk usage is initially high, to load these records into memory, but then drops off to an average of 0.4 MB/s.

5.2. Configurations using PHP, Java Servlets and EJB

Figure 7 and figure 8 compare the throughput of EJB in interactions per minute as a function of number of clients with that of PHP and Java servlets. In this comparison, the servlet server and the EJB server execute on a dedicated machine. For the browsing mix, EJB throughput initially grows linearly with the number of client, but stagnates around 80 clients to reach its peak at 850 interactions per minute with 100 clients, and remains constant after that. The curve for the bidding mix is similar, but achieves a slightly higher peak at 1,051 interactions per minute with 140 clients. In the linear part of both curves, the throughput of PHP and Java servlets is identical to that of EJB. Beyond that point, however, throughput of PHP and Java servlets continues to increase to achieve their peaks at client loads and throughput value an order-of-magnitude higher.

Figure 9 clearly shows that the CPU on the EJB server is the bottleneck resource with an average 99% utilization for both mixes. CPU utilization on all other machines is very modest: 23% for the browsing mix and 32% for the bidding mix on the servlet server, 17% for both mixes on the database server, and 4% and 6% for the browsing and bidding mix on the Web server.

As before, memory usage and disk usage are not bottlenecks. The EJB server uses about 190MB of memory. Network bandwidth is not a bottleneck either, although it is interesting to note that a very large number of small packets are exchanged between the EJB server and the database server (an average of 2,000 packets per second for a total bandwidth of 0.5Mb/s). This large number of small messages results from accesses to fields in the beans that require a single value from a row in the database.

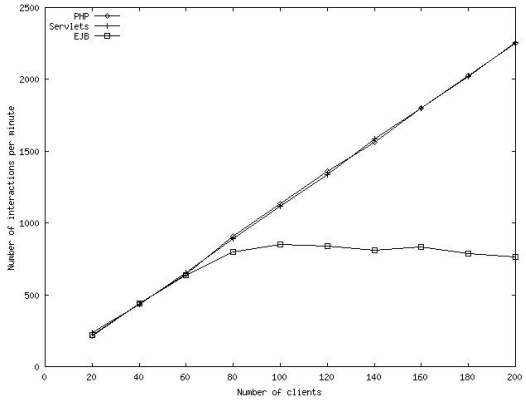

Figure 7. Auction site throughput in interactions per minute as a function of number of clients for the browsing mix using PHP, Java servlets and EJB.

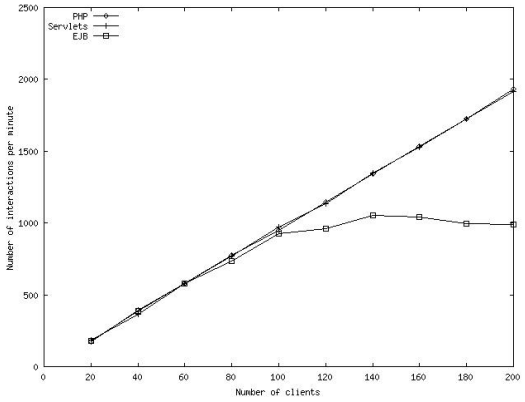

Figure 8. Auction site throughput in interactions per minute as a function of number of clients for the bidding mix using PHP, Java servlets and EJB.

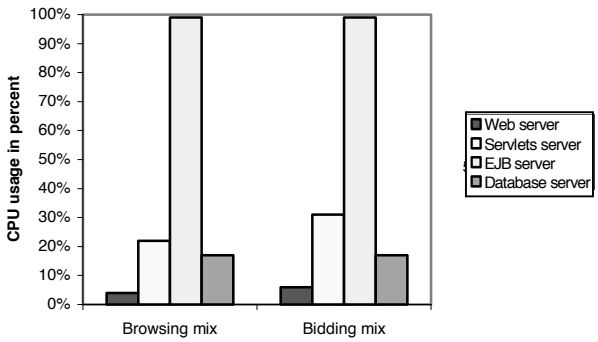

Figure 9. Auction site processor usage in percent at the peak throughput for the browsing and bidding mixes using EJB.

We have experimented with co-locating the EJB server with the servlet server or the database server, and found that both these configurations result in lower performance. Thus, reducing or avoiding network communication with the EJB server does not have a beneficial impact since CPU usage bounds the throughput.

5.3. Summary

PHP consumes less CPU time than servlets. We attribute this primarily to the fact that it executes in the same process as the Web server. While an advantage in terms of excution overhead, it restricts PHP to being co-located with the Web server on the same machine. If the Web server is the bottleneck and substantial unused CPU time is available on the database machine, then with the same number of machines, better overall performance is achieved by moving the servlet server to the database machine. Java servlets can also take advantage of the availability of a dedicated machine, leading to even better performance. EJB offers the most flexible architecture, allowing the Web server, the servlet server, the EJB server, and the database to execute on four different machines. The overhead of the EJB server is, however, so high that, even with the extra machine for the EJB server the overall performance remains much below that of PHP and Java servlets.

6. Online Bookstore

6.1. Configurations using PHP and Java Servlets

Figure 10 reports the online bookstore throughput in interactions per minute as a function of the number of clients for the shopping mix, which is the most representative mix for this benchmark, using PHP and Java servlets. The throughput is approximately the same for all three configurations (Web server with PHP, Web server with Java servlets on the same machine, and Web server and Java servlets on separate machines). In particular, initially, with a small number of clients, the throughput is identical. The three curves reach very slightly different peaks at approximately the same number of clients, and then start dropping off. In particular, locating the Java servlets on the Web server results in the lowest performance with a peak throughput of 497 interactions per minute reached with 70 clients. Next, locating the Java servlets on a dedicated machine peaks at 504 interactions per minute with 70 clients, which is also close to the peak of 532 interactions per minute obtained by PHP with 70 clients.

Figure 11 shows that the CPU utilization on each of the machines for each of the configurations, and figure 12 shows the CPU utilization on the Web server and the database server as a function of time, at the peak throughput for the configuration using PHP. The results are similar for the other configurations. The conclusion to draw from the combination of these two figures is that the

database CPU is the bottleneck. Although the average database CPU utilization ranges from 65% to 82% without reaching 100% in any of the configurations, figure 12 shows that the database CPU utilization in fact oscillates between 0% and 100%, resulting in an average utilization below 100%. The CPU on the Web server, however, never reaches 100%. The oscillations in the database CPU utilization are a result of the fact that for this application the different database queries differ in execution time by two orders of magnitude. Our conclusion that the database CPU is the bottleneck is further corroborated by the data shown in figure 13, which shows that the database CPU utilization beyond the peak throughput point for PHP (at 90 clients). Here, the average database CPU utilization approaches 100%. MySQL performance drops sharply when the load is that high, and this drop is reflected in the decline of the overall throughput in figure 10.

Memory is never the bottleneck resource, and remains stable after the initialization phase at 65MB and 110MB for the Java servlet and the database server, respectively. Memory usage on the Web server increases over time as the static images get read into the Linux buffer cache. The memory footprint of the Web server user processes remains as low as 70MB.

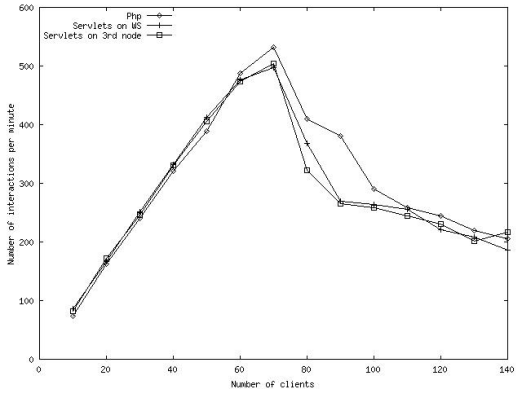

Figure 10. Online bookstore throughput in interactions per minute as a function of number of clients using PHP and Java servlets.

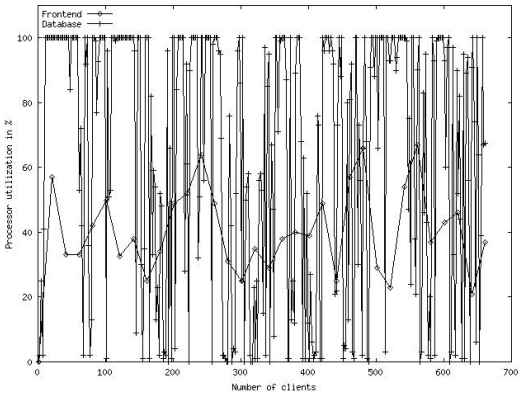

Figure 12. Online bookstore processor usage on the Web and database servers at the peak throughput for the shopping mix using PHP.

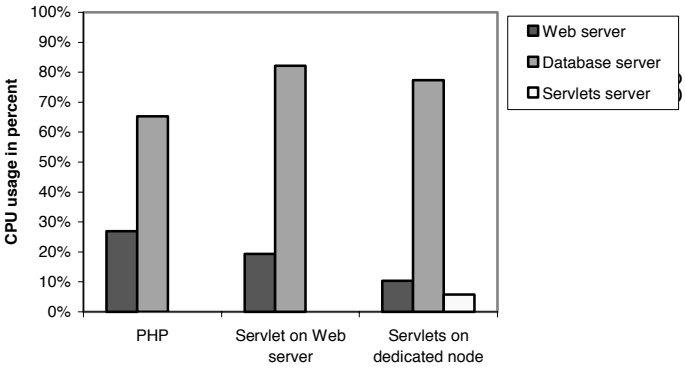

Figure 11. Online bookstore processor usage in percent at the peak throughput using PHP and Java servlets.

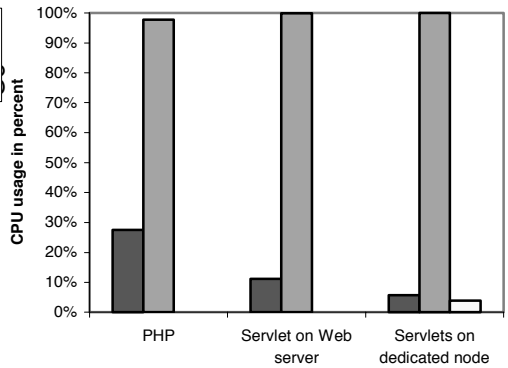

Figure 13. Online bookstore processor usage in percent with 90 clients using PHP and Java servlets.

Network bandwidth is not a bottleneck either for this experiment. The traffic related to the generation of the dynamic pages, between the Web server, the servlet server, and the database

server, is very low. Most of the traffic flowing from the Web server to the clients stems from the static images. At its maximum, it is around 2.5Mb/s.

The browsing and ordering mixes exhibit similar behavior to the shopping mix. The throughputs for PHP and Java servlets (in two different configurations) are very close when the database is lightly loaded and then vary slightly with the database approaching saturation. The reason is the same as for the shopping mix. The database CPU remains the bottleneck for these workload mixes, albeit with somewhat different values of peak throughput and number of clients at which peak throughput is achieved. All results reported in this section are for the smaller database. The database CPU remains the bottleneck, and the comparisons remain the same for the experiments with the larger database.

6.2. Configuration including PHP, Java Servlets and EJB

Figure 14 compares the throughput of the EJB implementation of the online bookstore with the PHP and Java servlets on a dedicated node versions. The throughput is given in interactions per minute as a function of number of clients. The EJB version peaks at 92 interactions per minute with 20 clients whereas the PHP and Java servlets implementation throughput increases linearly to much larger values with much larger numbers of clients.

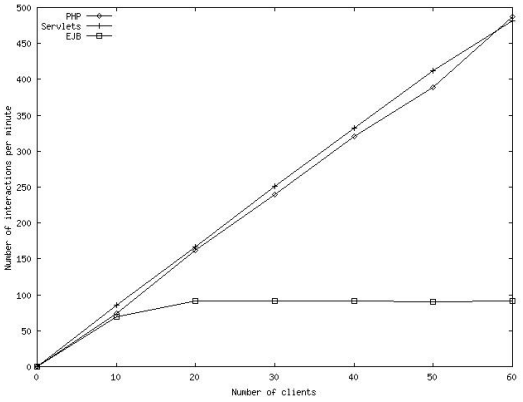

Figure 14. Online bookstore throughput in interactions per minute as a function of number of clients using PHP, Java servlets on a dedicated node and EJB.

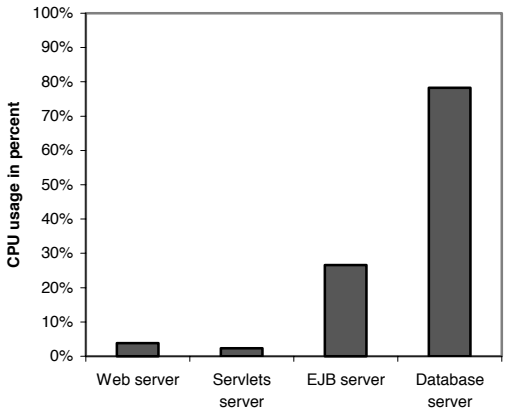

Figure 15. Online bookstore percentage CPU utilization at the peak throughput using EJB.

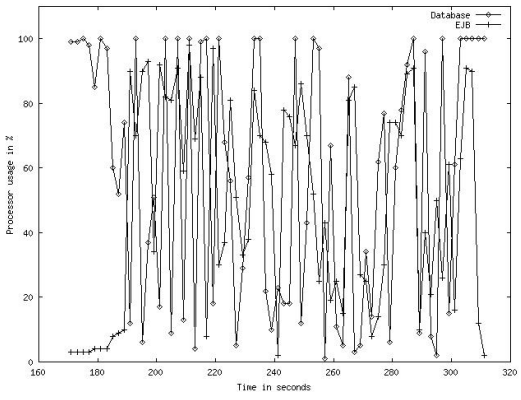

Figure 16. Online bookstore database and EJB server percentage CPU utilization as a function of time during steady state using EJB.

Unlike for the auction site, the CPU on the EJB server is not the sole bottleneck. As shown by figure 15, the CPU utilization is higher on the database server (78%) than on the EJB server (27%). Figure 16 shows the CPU utilization on the EJB server and the database server over time. When the database CPU is 100% utilized, the EJB server CPU is almost idle. When CPU usage drops on the database, CPU utilization on the EJB server rises up to 100%. Both the database and the EJB servers CPU are alternatively the bottleneck resources limiting the throughput to less than 100 interactions per minute.

With this very low throughput, network activity is not significant and can in no case become the bottleneck. Memory usage on each server is low as well, 50MB on the servlet server, 70MB on the Web server, 105MB on the EJB server, and 155MB on the database server.

6.3. Summary

With the database CPU being the bottleneck for the online bookstore, the differences between PHP and Java servlets is minimal. In this case, it is not possible to use the fact that servlets can be offloaded from the Web server to a different machine to increase throughput. The throughput of EJB remains problematic. Although the EJB server CPU and the database server CPU alternate in terms of being the bottleneck, overall performance remains considerably lower than with PHP or with Java servlets.

7. Related Work

Cain et al. [4] present a detailed architectural evaluation of TPC-W implemented using Java servlets. They investigate the impact of Java servlets on the memory system, the branch predictor, and the effectiveness of coarse-grain multithreading. Our study is aimed at studying the overall system and at studying differences between different software and system architectures.

Wu et al. compare PHP, Java servlets and CGI as approaches for web-to-database applications [21]. Their benchmark test is restricted to data retrieval (read) operations, while we use more realistic benchmarks with various. They only use a configuration where the Java servlet server runs on the Web server, and even with this configuration, servlets outperform the two scripting languages. However, they use PHP3 while we use PHP v4.0.6 that includes a lot of improvements. We have shown that Java servlets have a slightly larger overhead than PHP, but the flexibility of servlets allows architectures where the load is balanced among several servers.

The functionalities of Java servlets, PHP and CGI/Perl are compared in Sun's white paper [15]. They analyze the server-side mechanisms provided by each architecture. They conclude that Perl or PHP can help meet short-term goals but present the long-term benefits of using Java servlets for Web-based development, such as platform- and server-independent methods, and portable and reusable logic components. We propose a complementary comparison, focusing on performance in addition to functionality, and also including EJB.

The ECperf specification [16] is a first attempt at standardizing the evaluation of EJB servers. Only one result [17] has been published as of this writing, using an 8-way and a 4-way SMP node for the J2EE server and the database server, respectively. The reported throughput of 5961.77 BBops/min (Benchmark Business OPerationS per minute) is modest in view of the hardware platform used to achieve it. This confirms the large software overhead of the currently available implementations of EJB.

In our own earlier work [2], we analyze implementations of three benchmarks (an online bookstore, an auction site, and a bulletin board site) using PHP with the goal of discovering the bottlenecks in each benchmark. In this paper, we extend this work to a comparison of PHP with Java servlets and EJB on two of the benchmarks. The Web server CPU is the bottleneck for the

bulletin board. Therefore, we expect the comparison for the bulletin board to come out similar to the one for the auction site.

8. Conclusions

We compare three common systems for generating dynamic content for e-business sites: PHP, Java servlets, and Enterprise Java Beans (EJB). In terms of portability, Java servlets provide independence from the particular database used by performing all database operations through JDBC. The database interfaces in PHP are ad hoc. EJB goes one step further in that the bean methods are independent from the middleware platform, and can be called by any application, be it a Web server, servlets, or a completely different application.

In terms of programmability, the number of lines of code in our implementation with Java servlets is higher for the auction site than in the PHP implementation, and about the same for the online bookstore. The presence of the Java tools and the safety properties of the language help in debugging, but the safety properties also necessitate many re-cast's, reflecting well-known trade-offs between typed and untyped (scripting) languages. EJB is easy to use, in that it does not require SQL queries to be written, but our implementation requires more lines of (Java) code than servlets because of the many interfaces that need to be implemented.

In terms of performance, if the database is the bottleneck, then there is no difference between PHP and Java servlets. If the front-end Web server and/or the business logic are the bottleneck, then the Java servlets can be re-located to the database machine or to a separate machine to increase performance over PHP. If the Web server and the business logic run on the same machine, PHP is somewhat faster than Java servlets. Under all circumstances, EJB is considerably slower than PHP and Java servlets, even when an extra machine is dedicated as an EJB server.

References

[1] Rahim Adatia et al. – Professionnal EJB – *Wrox Press, ISBN 1-861005-08-3*, 2001.
[2] Cristiana Amza, Emmanuel Cecchet, Anupam Chanda, Alan L. Cox, Sameh Elnikety, Romer Gil, Julie Marguerite, Karthick Rajamani and Willy Zwaenepoel – Bottleneck Characterization of Dynamic Web Server Benchmarks – Rice University Computer Science Techical Report TR02-388.
[3] The Apache Software Foundation – http://www.apache.org/.
[4] Harold W. Cain, Ravi Rajwar, Morris Marden and Mikko H. Lipasti – An Architectural Evaluation of Java TPC-W – *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, 2001.
[5] eBay – http://www.ebay.com/.
[6] Jason Hunter and William Crawford – Java Servlet Programming 2nd edition – *O'Reilly, ISBN 0-596-00040-5*, 2001.
[7] JOnAS Open Source EJB Server – http://www.objectweb.org.
[8] Libelis Orcas – http://www.libelis.com.
[9] Lutris Enhydra Application Server – http:/www.lutris.com.
[10] Microsoft Active Server Pages – http://www.asp.net.
[11] Microsoft Internet Information server – http://www.microsoft.com/iis.
[12] MySQL Reference Manual v3.23.36 – http://www.mysql.com/documentation/.
[13] PHP Hypertext Preprocessor – http://www.php.net/.
[14] Guillaume Rivière – JOnAS performance results compared to JBoss and WebLogic – Evidian Technical Report, 2001.
[15] Sun Microsystems - Comparing Methods For Server-Side Dynamic Content White Paper – *http://java.sun.com*, 2000.

[16] Sun Microsystems – ECperf specification - http://java.sun.com/j2ee/ecperf/, 2001.
[17] Sun Microsystems – ECperf Benchmark for SUN Fire V880 running Borland AppServer 4.5 and Sun Enterprise 450 running Oracle 8i - http://ecperf.theserverside.com/ecperf/, 2001.
[18] Sysstat package – http://freshmeat.net/projects/sysstat/.
[19] Jakarta Tomcat servlet container – http://jakarta.apache.org/tomcat/.
[20] Transaction Processing Performance Council– http://www.tpc.org/.
[21] Amanda Wu, Haibo Wang and Dawn Wilkins – Performance Comparison of Alternative Solutions For Web-To-Database Applications – *Proceedings of the Southern Conference on Computing*, 2000.