Performance Comparison of Middleware Architectures for Generating Dynamic Web Content

Emmanuel Cecchet[1], Anupam Chanda[2], Sameh Elnikety[3], Julie Marguerite[1], and Willy Zwaenepoel[3]

[1] INRIA, Projet Sardes, 655, Avenue de l'Europe, 38330 Montbonnot St Martin, France
{Emmanuel.Cecchet,Julie.Marguerite}@inrialpes.fr
[2] Rice University, 6100 Main Street, MS-132, Houston, TX, 77005, USA
anupamc@cs.rice.edu
[3] Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland
{sameh.elnikety,willy.zwaenepoel}@epfl.ch

Abstract. On-line services are making increasing use of dynamically generated Web content. Serving dynamic content is more complex than serving static content. Besides a Web server, it typically involves a server-side application and a database to generate and store the dynamic content. A number of standard mechanisms have evolved to generate dynamic content. We evaluate three specific mechanisms in common use: PHP, Java servlets, and Enterprise Java Beans (EJB). These mechanisms represent three different architectures for generating dynamic content. PHP scripts are tied to the Web server and require writing explicit database queries. Java servlets execute in a different process from the Web server, allowing them to be located on a separate machine for better load balancing. The database queries are written explicitly, as in PHP, but in certain circumstances the Java synchronization primitives can be used to perform locking, reducing database lock contention and the amount of communication between servlets and the database. Enterprise Java Beans (EJB) provide several services and facilities. In particular, many of the database queries can be generated automatically.
We measure the performance of these three architectures using two application benchmarks: an online bookstore and an auction site. These benchmarks represent common applications for dynamic content and stress different parts of a dynamic content Web server. The auction site stresses the server front-end, while the online bookstore stresses the server back-end. For all measurements, we use widely available open-source software (the Apache Web server, Tomcat servlet engine, JOnAS EJB server, and MySQL relational database). While Java servlets are less efficient than PHP, their ability to execute on a different machine from the Web server and their ability to perform synchronization leads to better performance when the front-end is the bottleneck or when there is database lock contention. EJB facilities and services come at the cost of lower performance than both PHP and Java servlets.

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 and image files. Dynamic content is used in many online services that need access to current information such as e-commerce and electronic banking. Also, it is used to customize the look-and-feel of Web pages according to the preferences of each user.

Dynamic Web content is generated by a combination of a Web server, a dynamic content generator, and a back-end database (see figure 1). The Web server serves all static content and forwards requests for dynamic content to the dynamic content generator. The dynamic content generator executes the code that captures the business logic of the Web site and issues queries to the database, which stores the dynamic state of the site.

In more detail, when a Web server receives an HTTP request for dynamic content, it forwards the request to the dynamic content generator. The dynamic content generator executes the corresponding code, which may need to access the database to generate the response. The dynamic content generator formats and assembles the results into an HTML page. Finally, the Web server returns this page as an HTTP response to the client.

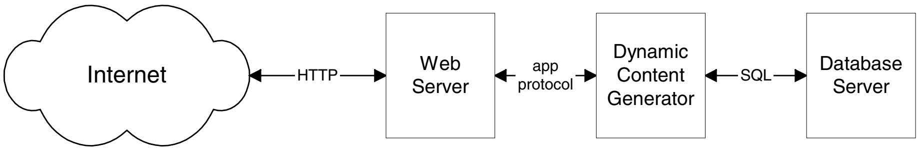

Fig. 1. Typical configuration of a dynamic content Web site.

The implementation of the application logic may take various forms, including scripting languages such as PHP [14] that execute as a module in a Web server such as Apache [4], Microsoft Active Server pages [11] that are integrated with Microsoft's IIS server [12], Java servlets [8] that execute in a separate Java virtual machine, and full application servers such as an Enterprise Java Beans (EJB) server [22]. This study focuses on three software systems for generating dynamic Web content: PHP, Java servlets, and EJB.

PHP is a scripting language in which SQL queries can be embedded. Similarly, Java servlets allow SQL queries to be embedded in the Java code. In both PHP and Java servlets, the application programmer writes the SQL queries. With EJB a number of beans are defined. Session beans implement the business logic of the application, and entity beans implement the persistence services. Roughly speaking, each entity bean corresponds to a database table, an entity bean instance corresponds to a row in the table, and an entity bean member to a column. To access the persistent state, bean methods are called, which in turn issue SQL queries to the database. In contrast to PHP and Java servlets, the SQL queries issued by the beans are generated automatically[1]. Most commonly, the bean methods are called from Java servlets, which in this case only implement the presentation logic of the site.

If only the dynamic content generator accesses the database, both Java servlets and EJB can use Java synchronization mechanisms to offload some of the synchronization and locking typically performed by the database. This can lead to improved performance in the presence of database lock contention.

[1] With container-managed persistence. The alternative, bean-managed persistence, in which the application programmer writes the queries explicitly, is not considered in this paper.

PHP executes as a module in the Web server, sharing the same process address space. Both servlets and EJB execute in a separate Java virtual machine, requiring interprocess communication between the Web server and the dynamic content generator. This separation, however, allows an extra degree of freedom in configuring the system, in that the servlets and the EJB can be deployed on a separate machine from the Web server.

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 server front-end, better performance can be achieved by locating Java servlets on a separate machine. The introduction of EJB adds significant overhead that cannot be alleviated by putting the EJB and the associated servlets on separate machines.

To evaluate the performance of these architectures, we use two benchmarks: an online bookstore modeled after the TPC-W specification [20] and an auction site modeled after eBay.com [7]. We perform our experiments on commodity hardware. Each machine contains a 1.33GHz AMD Athlon, 768MB main memory and 60GB disk. The machines are connected to each other and to a set of machines running client emulation software by a switched 100Mbps Ethernet. For the online bookstore, the database server is the bottleneck, and the auction site saturates the server front-end. In all the experiments, the memory, disk and the network are never the performance bottleneck except for one configuration.

The remainder of the paper is organized as follows. Section 2 provides necessary 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 online bookstore and the auction site benchmarks, respectively. Section 7 discusses related work, and Section 8 presents our conclusions.

2 Background

2.1 PHP (Hypertext Preprocessor)

PHP [14] 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. PHP support generally takes the form of a server module that is integrated into the HTTP Web server. PHP is executed within the Web server process and does not incur any interprocess communication overhead. When the HTTP Web server identifies a PHP tag, it invokes the PHP interpreter module that executes the script. Requests to the database are explicit and are performed using an ad hoc interface.

2.2 Java HTTP Servlets

An HTTP servlet [8] is a Java class that can be dynamically loaded by a servlet engine and runs in a Java Virtual Machine (JVM). After the initial load, the servlet engine invokes the servlet using local calls, but since the JVM is a separate process from the Web server, interprocess communication takes place for each request. Servlets access the database explicitly, using the standard JDBC interface, which is supported by all major databases.

Servlets can use all the features of Java. In particular, they can use Java built-in synchronization mechanisms to perform locking operations[2]. If only the servlet engine accesses the database, locking in the servlet engine can replace some locking in the database, which may reduce database lock contention and communication between the servlet engine and the database.

2.3 Enterprise Java Beans

The purpose of an Enterprise Java Beans (EJB) server is to abstract the application business logic from the underlying middleware. An EJB server provides a number of services such as database access (JDBC), transactions (JTA), messaging (JMS), naming (JNDI) and management support (JMX). The EJB server manages one or more EJB containers. The container is responsible for providing component pooling and lifecycle management, client session management, database connection pooling, persistence, transaction management, authentication, and access control.

We use two types of EJB in our implementations: entity beans that map data stored in the database (usually one entity bean instance per database table row), and session beans that are used either to perform temporary operations (stateless session beans) or represent temporary objects (stateful session beans). As with Java servlets, Java's synchronization mechanisms can be used to offload locking from the database to the application server.

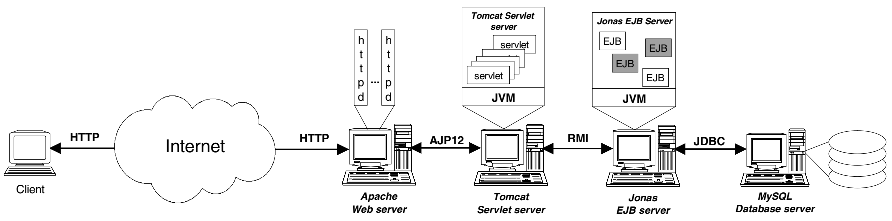

Fig. 2. Using an Enterprise Java Bean server to generate dynamic Web content.

Figure 2 shows an example of an architecture including an EJB server. Java servlets are usually used with EJB to call the bean methods. First, a client sends a request to the HTTP server. The HTTP server invokes the servlet engine using a well-defined protocol (AJP12). The servlet communicates with the EJB server (using RMI) to retrieve the information needed in order to generate the HTML reply. The EJB server in turn calls the database to maintain the state of the beans. These calls are generated automatically.

[2] It is possible to do locking using PHP versions 3 and 4 on Unix-like operating systems that support System V Semaphores. We do not consider this possibility in the paper because this feature is not available on all platforms.

2.4 Summary

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

Java servlets access the database using JDBC. This makes them easily portable between databases. Contrary to the PHP interpreter, the servlet engine runs in a JVM as a separate process from the Web server. Therefore, servlets can be placed on a machine different from the one running the Web server. This flexibility can be used to improve load balancing. Also, servlets can use all the Java language features, especially its synchronization mechanisms. Servlets, however, incur the cost of interprocess communications with the Web server.

The EJB architecture offers a level of indirection as it abstracts the application logic from any specific platform or infrastructure. In particular, database accesses are generated automatically as part of the bean methods.

3 Benchmarks

We describe the two benchmarks, which we use to compare PHP, Java servlets, and EJB. We choose these two benchmarks because the first one stresses the database, while the second one stresses the Web server.

3.1 Online Bookstore Benchmark

The online bookstore benchmark implements the TPC-W specification [20], a transactional Web benchmark for evaluating e-commerce systems. Our online bookstore implementation 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.

The database stores all persistent data except for the images, which are stored in the file system of the Web server. The database manages eight tables: *customers*, *address*, *orders*, *order_line*, *credit_info*, *items*, *authors*, and *countries*. The *order_line*, *orders* and *credit_info* tables store the details of the orders that have been placed. In particular, *order_line* stores the book ordered and the quantity and discount. *Orders* stores the customer identifier, the date of the order, information about the amount paid, the shipping address and the status. *Credit_info* stores credit card information such as its type, number and expiry date. 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, are maintained in the *customers* and *address* tables.

We implemented the 14 different interactions specified in the TPC-W benchmark specification. Of the 14 interactions, six are read-only and eight have update queries that change the state of the database. The read-only interactions include access to the home page, new products and best-sellers listings, requests for product detail, and two search interactions. Read-write interactions include user registration, updates to the shopping cart, two purchase interactions, two involving order inquiry and display, and two administrative updates. We use the same Markov model for the distribution of interactions as specified in TPC-W. Interactions may also involve requests for embedded images corresponding to an item in the inventory as well as navigational buttons and logos. All interactions access the database server to generate dynamic content, except for one interaction that involves only static content.

TPC-W specifies three different workload mixes, differing in the ratio of read-only to read-write interactions. The browsing mix contains 95% read-only interactions, the shopping mix 80%, and the ordering mix 50%. The shopping mix is considered the most representative mix for this benchmark. The database scaling parameters are 10,000 items and 288,000 customers. This corresponds to a database size of 350MB, which fits entirely in the main memory of database server. The images stored in the Web server file system use 183MB of disk space.

3.2 Auction Site Benchmark

Our auction site benchmark implements the core functionality of an auction site: selling, browsing and bidding. It does 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, their rating and the 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 nine tables: *users*, *items*, *old_items*, *bids*, *buy_now*, *comments*, *categories*, *regions* and *ids*. 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* and *old_items* tables contain the name that briefly describes the item and a more extensive description, normally 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 on the *bids* table. This redundant information is necessary to keep an acceptable response time for browsing requests. As users browse and bid only on items that are currently for sale, we split the *items* table in separate *items* and *old_items* tables. The vast majority of requests access the new items table, thus considerably reducing the database working set.

Our auction site defines 26 interactions that can be accessed 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 [7]). Browsing items also includes consulting the bid history and the seller's information. We define two workload mixes: a browsing mix made up of 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 size 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 any auction. The *users* table has 1 million entries. We assume that users give feedback (comments) for 95% of the transactions. The comments table contain about 500,000 comments. The total size of the database, including indices, is 1.4GB.

4 Hardware and Software Environment

4.1 Client Emulation Implementation

We implement a client-browser emulator. A client session is a sequence of interactions for the same client. For each client 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.

The think time and session time for both 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 [20]. 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. To make the comparison fair, we use exactly the same queries to the database in both environments. The only exception is when locking in Java servlets is used. In this case, we remove some "LOCK TABLES" and "UNLOCK TABLES" SQL statements.

With EJB, we separate the presentation logic that remains in the Java servlets from the business logic that is implemented by EJB. The Java servlets are used only as the presentation tier as defined in Adatia et al. [1], to generate the HTML reply from the information retrieved from the bean. We use the session façade pattern [2] represented by figure 3 to implement the business logic. The main business logic resides in stateless session façade beans that access entity beans with container-managed persistence. The entity beans access the database. This design uses the relevant features of the EJB container, and at the same time provides the best performance compared to other designs with entity beans [6].

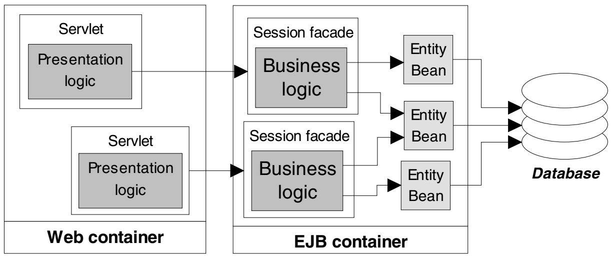

Fig. 3. Session façade design pattern.

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. With that value, the number of Apache processes is never a limit on performance.

The servlet engine is Jakarta Tomcat v3.2.4 [19], running on Sun JDK 1.3.1. The EJB server is JOnAS v2.5 [9], 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 [6]. JOnAS is also integrated in production application servers such as the Lutris Enhydra Application Server [10].

We use MySQL v.3.23.43-max [13] as our database server with the MyISAM 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

We use four server machines. Each machine has 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 experiment. All machines are connected through a switched 100Mbps Ethernet LAN.

4.5 Measurement Methodology

Each experiment is composed of 3 phases. A ramp-up phase initializes the system until it reaches a steady-state throughput level. We then switch to the measurement phase during which we perform all our measurements. Finally, a ramp-down phase sustains the same request rate as the measurement phase to allow for differences in client machines clocks and to allow for all pending requests to terminate. 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 the two applications. The online bookstore uses 1, 20 and 1 minutes for the ramp-up, measurement, and ramp-down phases, respectively. The auction site uses 5, 30 and 5 minutes. These lengths of time are chosen based on observation of the length of time necessary to reach a steady state and to obtain reproducible results.

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

4.6 Configurations

We experiment with four different configurations, shown in Figure 4.

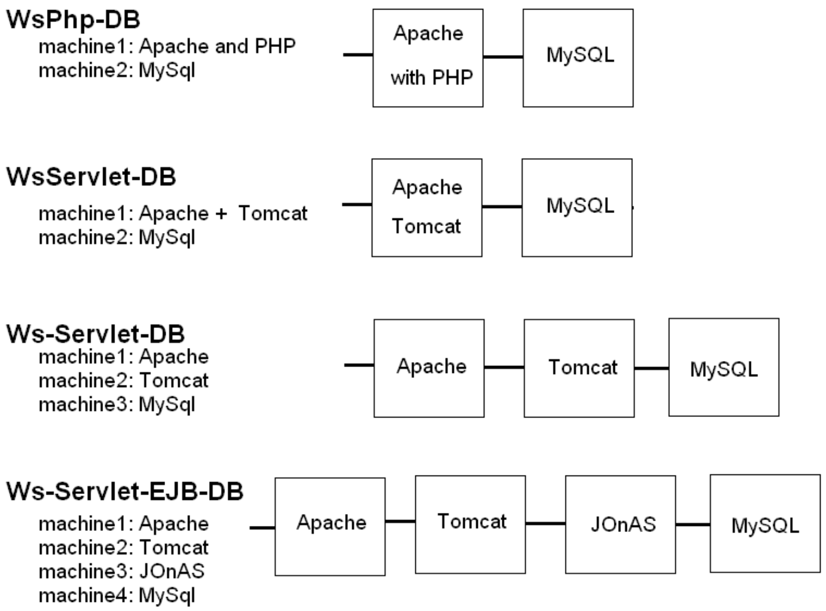

Fig. 4. The four software and hardware configurations evaluated.

We always run the database on a separate machine. PHP is implemented as a server module, so it needs to run on the same machine as the Web server. We refer to the PHP configuration as WsPhp-DB. Servlets can be located on the Web server machine or on a separate machine. We refer to these two configurations as WsServlet-DB and Ws-Servlet-DB, respectively. The configurations WsPhp-DB, WsServlet-DB and Ws-Servlet-DB contain exactly the same database queries. The configurations WsServlet-DB(sync) and Ws-Servlet-DB(sync) perform locking in the servlet engine. They contain the same database queries as WsServlet-DB and Ws-Servlet-DB, except that many of queries that explicitly acquire and release database locks (e.g., “LOCK TABLES” and “UNLOCK TABLES” SQL statements) are removed. For the EJB configuration (Ws-Servlet-EJB-DB), we use four machines, one each for the Web server, the servlet engine, the EJB application server and the database server.

5 Experimental Results for the Online Bookstore

This benchmark contains complex database queries, which stress the database server. For all configurations, the bottleneck is the database server. The bottleneck results either from CPU saturation or from database lock contention. As shown in the next subsections, for this benchmark the database interface (i.e., the set of queries issued to the database) is the key factor that affects the performance. Configurations that have the same database interface have approximately the same throughput.

5.1 Shopping Mix

Figure 5 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 workload for this benchmark.

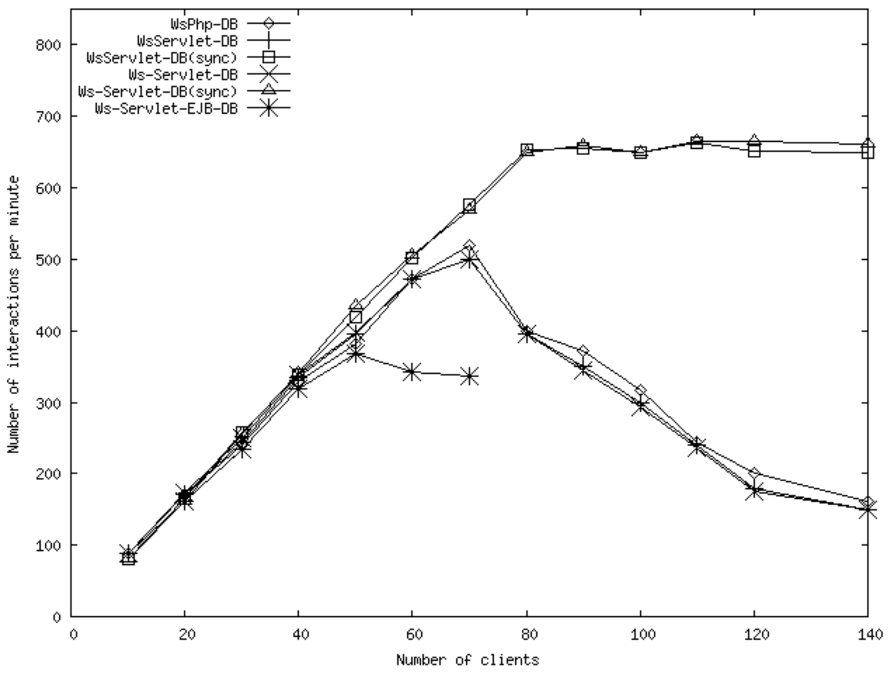

Fig. 5. Online bookstore throughput in interactions per minute as a function of number of clients for the shopping mix.

The PHP configuration WsPhp-DB gives a peak throughput of 520 interactions per minute. As the load increases beyond the peak point, the performance drops because of database lock contention.

The Java servlet configurations WsServlet-DB and Ws-Servlet-DB give approximately the same throughput as the PHP configuration, because these three configurations have the same database interface and contain exactly the same queries. Moving

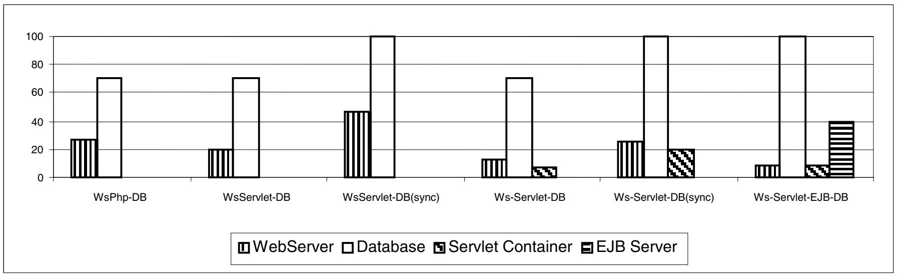

Fig. 6. Online bookstore percentage CPU utilization at the peak throughput for the shopping mix.

servlets to another machine as in Ws-Servlet-DB does not produce any performance gain because the servlets issue the same set of queries and database lock contention remains the bottleneck.

The servlets configurations WsServlet-DB(sync) and Ws-Servlet-DB(sync), which perform some of the locking using the Java synchronization mechanisms, give better performance because of the reduction in database lock contention. The peak throughputs are 663 and 665 interactions per minute, respectively. The bottleneck resource at the peak is the CPU of the database server, which is 100% utilized throughout the peak plateau.

The EJB configuration Ws-Servlet-EJB-DB performs the worst, because it requires too many short queries to maintain the state of beans.

The above results can be further explained by examining Figure 6, which shows the CPU utilization for all configurations and for each machine. For the configurations WsPhp-DB, WsServlet-DB and Ws-Servlet-DB, the CPU utilization of the database is around 70%. It does not reach 100% due to database lock contention. For the configurations WsServlet-DB(sync) and Ws-Servlet-DB(sync), performing some locking operations in the servlets engine alleviates database lock contention and the CPU utilization of the database server reaches 100%. For the EJB configuration Ws-Servlet-EJB-DB, the CPU of the database machine reaches 100%, because the application server issues many short queries to maintain the state of the entity beans.

The other resources (e.g., memory, disk bandwidth, network, process limit) are not the bottleneck for any of the configurations. Memory usage on the database remains constant at 410MB. On the Web server memory usage increases over time as the static images are read into the Linux buffer cache. The memory footprint of the Web server user processes remains as low as 70MB. The traffic between the Web server, the servlet engine, and the database server is very low. Network traffic is the heaviest between the Web server and the clients, but it remains less than 3.5Mb/s. Most of this traffic stems from the static images. Steady-state disk I/O is less than 20 transfers per second for all machines.

5.2 Browsing Mix

Figure 7 shows the throughput using the browsing mix of the online bookstore benchmark. The curves are lower than those in Figure 5, because this workload contains more read queries, which are generally more complex than the update queries.

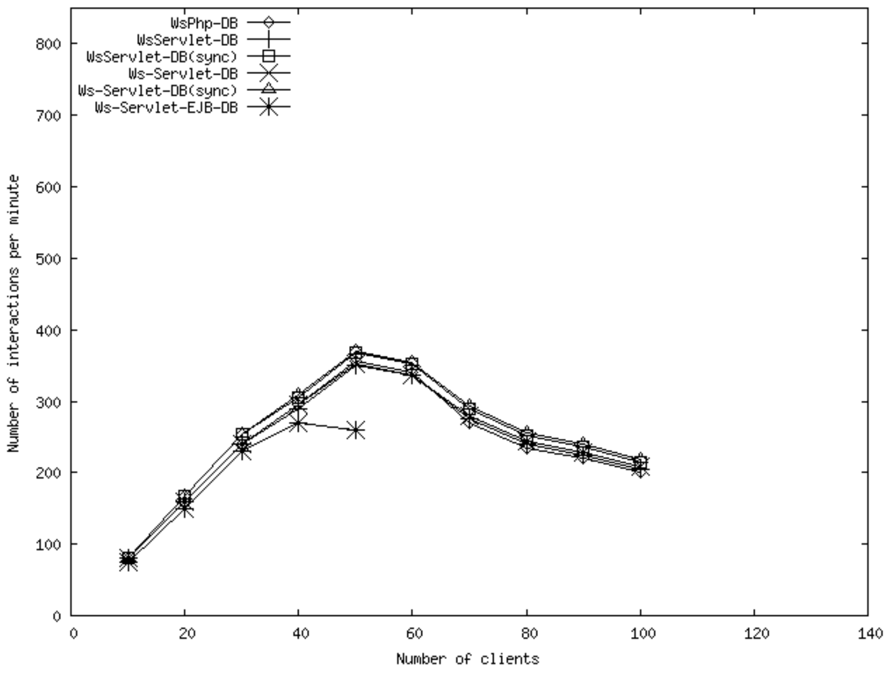

Fig. 7. Online bookstore throughput in interactions per minute as a function of number of clients for the browsing mix.

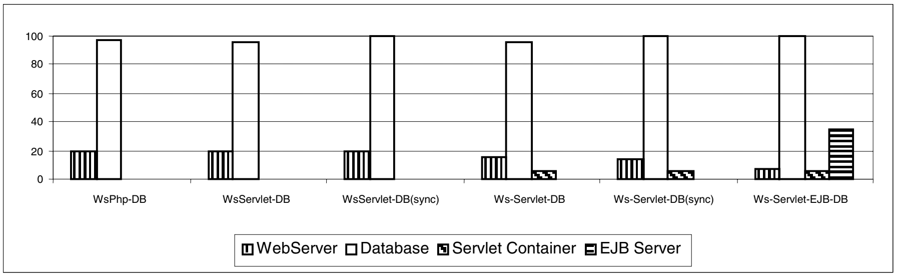

Fig. 8. Online bookstore percentage CPU utilization at the peak throughput for the browsing mix.

The bottleneck for this mix is invariably the CPU of the database server. There is no lock contention in the database because of the dominance of the read queries. Hence, performing the locking operations in the servlet engine does not yield any noticeable performance gain. For this reason, all configurations, except WS-Servlet-EJB-DB, have the same performance. The performance of Ws-Servlet-EJB-DB is low for the same reason as in the shopping mix.

Figure 8 shows the CPU utilization at the peak throughput for all machines in the different configurations. The figure confirms that the CPU of the database server is the bottleneck for all configurations.

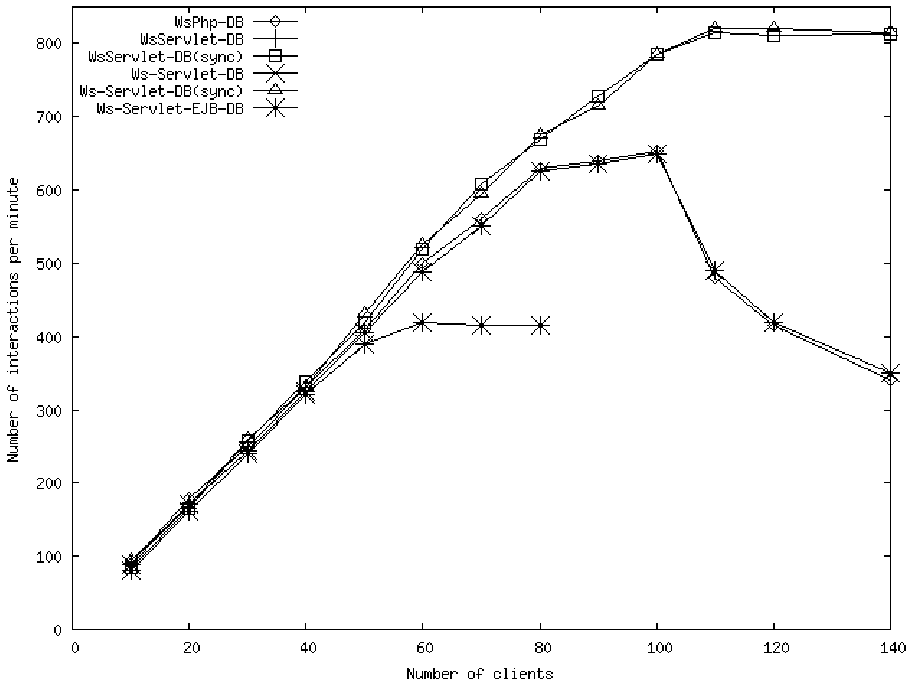

Fig. 9. Online bookstore throughput in interactions per minute as a function of number of clients for the ordering mix.

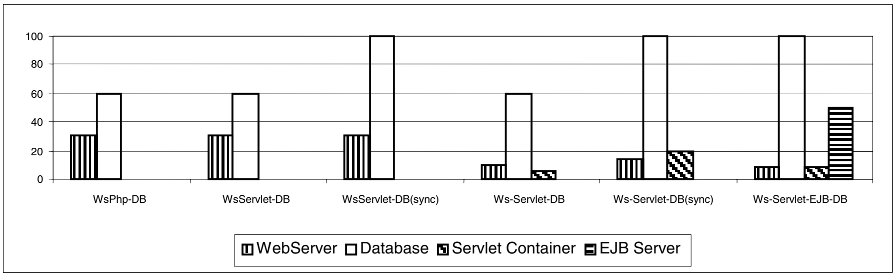

Fig. 10. Online bookstore percentage CPU utilization at the peak throughput for the ordering mix.

5.3 Ordering Mix

The ordering mix contains shorter update queries than the shopping mix and as a result gives higher throughput. Figure 9 depicts the throughput for different configurations, and Figure 10 shows the CPU utilization at the peak point for each configuration.

The configurations WsPhp-DB, WsServlet-DB and Ws-Servlet-DB have approximately the same throughput curves, because the bottleneck is database locking. Figure

10 shows that for these configurations the database server CPU utilization remains around 60%.

Performing the locking operations on the servlet engine, as in WsServlet-DB(sync) and Ws-Servlet-DB(sync), gives much better performance. It reduces the database lock contention and allows the CPU of the database server to reach 100% utilization.

As expected, the performance of the EJB configurations Ws-Servlet-EJB-DB is considerably lower than other configurations.

5.4 Summary

With the database being the bottleneck for the online bookstore, there is little difference between PHP and Java servlets when they use exactly the same queries. Therefore, offloading the servlets to a new machine does not increase throughput. Using Java synchronization mechanisms, it is possible for servlets to perform some locking operations, which reduces lock contention in the database and gives better performance for workloads with a moderate-to-high fraction of writes. The throughput of EJB is lower than with PHP or with Java servlets.

6 Experimental Results for the Auction Site

This benchmark contains mostly short database queries. For example, many update queries correspond to inserting a new bid, buying an item, or leaving a comment. Similarly, the read queries return a list of items that meet specific criteria, show the details of an item, show the history and the comments of a user, or retrieve the status of the user's active bids. This benchmark stresses the dynamic content generator, which communicates with the database and assembles the response. Our measurements show that the bottleneck is the CPU of the dynamic content generator for all configurations except in one case. This benchmark contrasts different implementations of dynamic content generation.

6.1 Bidding Mix

Figure 11 reports the throughput in interactions per minute as a function of number of clients for the bidding mix of this benchmark.

First, we compare WsPhp-DB, WsServlet-DB and WS-Servlet-DB. The PHP configuration WsPhp-DB peaks at 9,780 clients per minute with 1,100 clients. Servlets on the Web server, WsServlet-DB, achieves a lower peak of 7,380 interactions per minute with 700 clients. The best configuration among these three configurations is the one in which the servlets run on a dedicated machine, Ws-Servlet-DB, with 10,440 interactions per minute at 1,200 clients.

These results can be explained by looking at figure 12, which reports the CPU utilization of different machines at the peak throughput for each configuration. When the dynamic content generator runs on the Web server, as in WsPhp-DB and WsServlet-DB, the Web server CPU is the bottleneck with 100% CPU utilization. PHP is more efficient than Java servlets and gives around 33% more peak throughput. We attribute this difference in part to the overhead of communicating between the Web server and the servlet engine, which execute in separate processes. Profiling measurements indicate that, on average, the cost of sending one character of dynamic con-

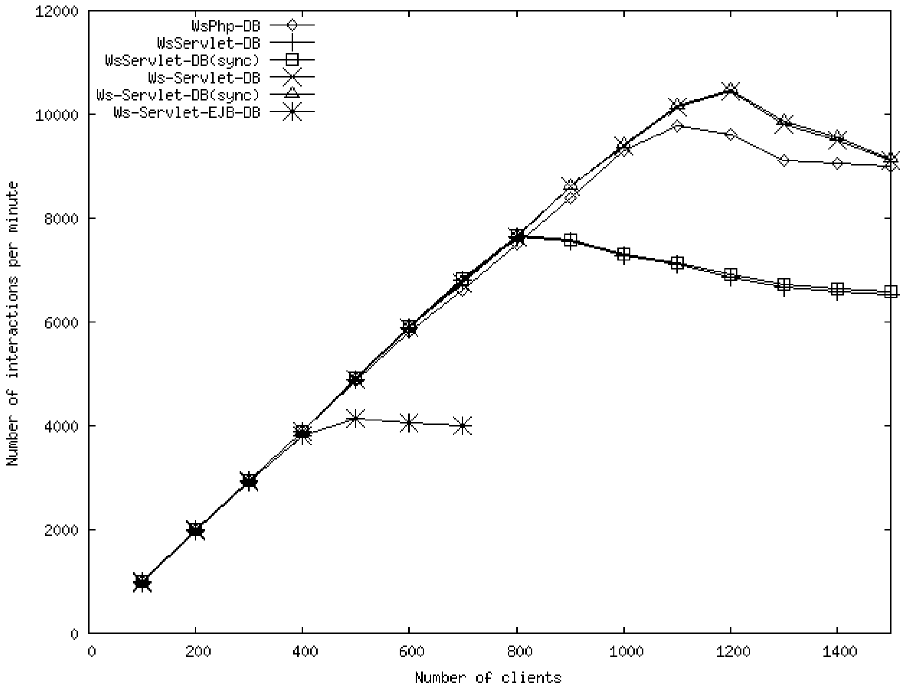

Fig. 11. Auction site throughput in interactions per minute as a function of number of clients for the bidding mix.

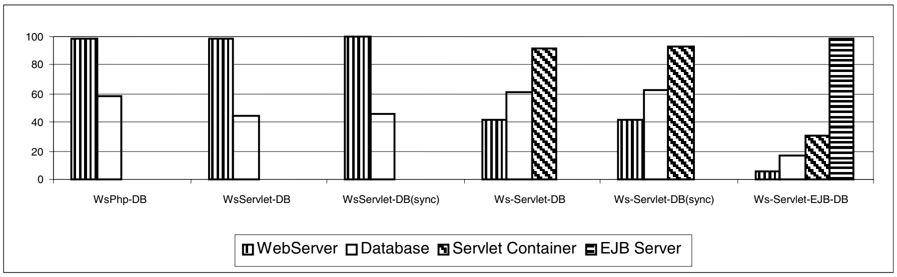

Fig. 12. Auction site percentage CPU utilization at the peak throughput for the bidding mix.

tent between the servlet engine and the Web server is 191 microseconds. In contrast, PHP does not incur such costs. Also, servlets use a type 4 JDBC driver that is written in Java and interpreted. PHP uses a native code database driver, which is presumably faster than the type 4 JDBC driver. Finally, when a dedicated machine is used for the servlets as in Ws-Servlet-DB, the best performance is achieved. The benefit of an extra CPU outweighs the extra communication costs resulting from putting the servlet engine on a separate machine.

Second, consider the configurations WsServlet-DB(sync) and Ws-Servlet-DB(sync). Because the queries in this benchmark are short, the database machine

CPU is at most 62% utilized for all configurations. Also, there is no lock contention in the database. Thus, performing locking in the Java servlets does not increase the throughput. This explains why in figure 11 the throughput curve for WsServlet-DB(sync) coincides with the curve for WsServlet-DB, and Ws-Servlet-DB(sync) coincides with Ws-Servlet-DB.

Finally, the EJB configuration Ws-Servlet-EJB-DB initially exhibits a linear increase in throughput with the number of clients, but stagnates around 500 clients to reach its peak at 4136 interactions per minute. Figure 12 clearly shows that the CPU on the EJB server is the bottleneck resource with average 99% utilization. CPU utilization on all other machines is very modest: 32% on the servlet engine, 17% on the database server, and 6% on the Web server.

None of the other resources (memory, disk, and network) forms a bottleneck for any configuration. For instance, we observe a maximum memory usage of 110MB, 95MB, and 390MB on the Web server, the servlet engine and the database server, respectively. Although the database is much larger than the physical memory of the database machine, most accesses are to records relating to new auctions, which is a small subset. Disk usage is initially high in order to load these records into memory, but then drops off to an average of 0.4 MB/s. The communication between the servlet engine and the database is modest at an average of 1.8Mb/s.

For the EJB configuration Ws-Servlets-EJB-DB, the EJB server uses about 190MB of memory. Although network bandwidth is not a bottleneck, 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 to be read or updated in the database.

6.2 Browsing Mix

The browsing mix only contains read-only queries. The majority of these queries are short. This results in making the dynamic content generator or the Web server the bottleneck resource rather than the database server.

Figure 13 reports the throughput in interactions per minute as a function of number of clients for the browsing mix workload. The curves follow similar trends as those for the bidding mix. In particular, the PHP configuration WsPhp-DB gives around 25% better peak throughput than the corresponding servlet configuration WsServlet-DB. Moving servlets to a dedicated machine gives the best performance.

Performing locking in the Java servlets does not yield any increase in the throughput. Therefore, the configuration WsServlet-DB(sync) has identical throughput to WsServlet-DB, and Ws-Servlet-DB has identical throughput to Ws-Servlet-DB. The EJB configuration Ws-Servlet-EJB-DB shows the lowest throughput.

Similar to the shopping mix, the disk and memory are never a bottleneck. Also, the network bandwidth is the bottleneck only in the case discussed above.

Figure 14 depicts the CPU utilization of different machines at the peak points for each configuration. The bottleneck resource is the CPU of the server running the dynamic content generator, except for the configurations Ws-Servlet-DB and Ws-Servlet-DB(sync). For these two configurations, the Java servlets run on a dedicated machine. They achieve the highest throughput of 12,000 interactions per minute at 12,000 clients. The CPU of the Web server approaches 100% because of the network

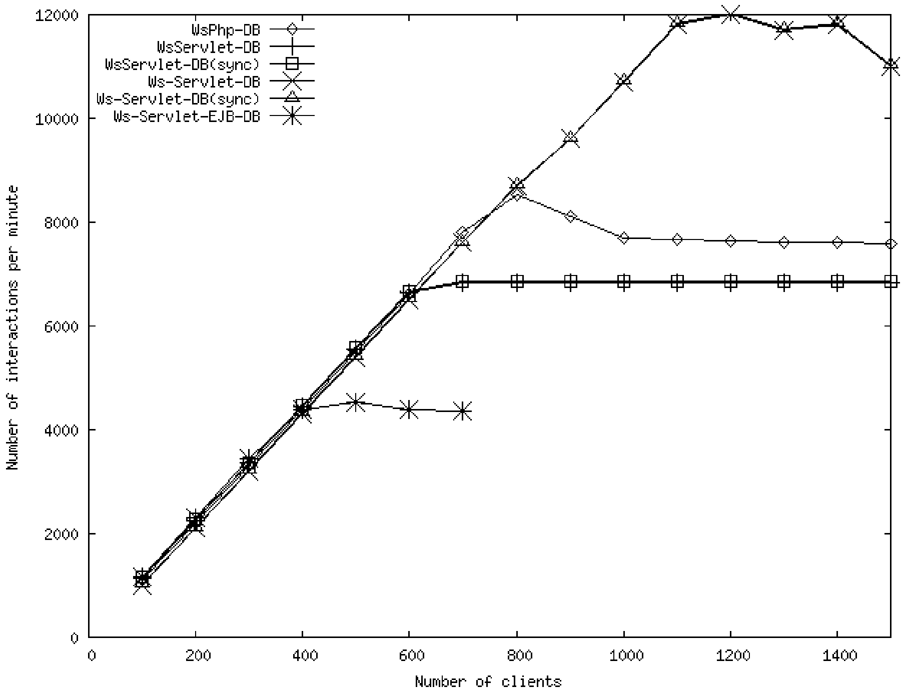

Fig. 13. Auction site throughput in interactions per minute as a function of number of clients for the browsing mix.

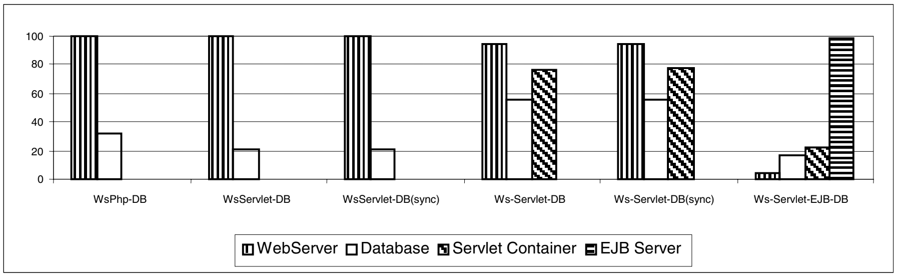

Fig. 14. Auction site percentage CPU utilization at the peak throughput for the browsing mix.

traffic on the Web server. In this configuration the network traffic on the Web server reaches 94Mb/s (80Mb/s to clients and 14Mb/s from the servlet engine).

6.3 Summary

PHP consumes less CPU time than servlets. We attribute this primarily to the fact that it executes in the same process and address space as the Web server. Although an advantage in terms of execution overhead, it restricts PHP to being co-located with the Web server on the same machine. If the Web server is the bottleneck, then better overall performance is achieved by moving the servlet engine to a dedicated machine. Performing locking in the servlet engine does not improve the throughput if there is no database lock contention. EJB offers the most flexible architecture, but even using four machines to run the Web server, the servlet engine, the EJB server and the database, the EJB performance is below that of PHP and Java servlets.

7 Related Work

Cain et al. [5] 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 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. They only use a configuration where the Java servlet engine 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. Our Java servlets environment has a larger overhead than PHP, but the flexibility of servlets allows configurations 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, and also including EJB.

The ECperf specification [16] was a first attempt at standardizing the evaluation of EJB servers. Since then, it has been replaced by SPECjAppServer2002 (Java Application Server). SPECjAppServer2002 is a client/server benchmark for measuring the performance of Java Enterprise Application Servers using a subset of the J2EE APIs in a complete end-to-end Web application [17]. The results from SPECAppServer2002 report moderate throughputs (in BOPS or Business OPerations per Second) 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 [3], 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. We do not use the third benchmark, the bulletin board, in this study because the Web server CPU is the bottleneck for the bulletin board. Therefore, we expect the results for the bulletin board to be similar to the auction site results.

In other earlier work [6] we study the scalability of EJB applications using the auction site benchmark with different enterprise bean types and design patterns. In this paper, we use the session façade design pattern with stateless session beans and entity beans using local interfaces. This design offers the best tradeoff between using EJB services and obtaining high performance.

8 Conclusions

We compare three middleware architectures for generating dynamic content: PHP, Java servlets, and Enterprise Java Beans (EJB). PHP is tied to the Web server. The database interfaces in PHP are ad hoc and have to be written for each database. Java servlets run independently from the Web server. They provide independence from the particular database used by performing all database operations through JDBC. EJB goes one step further and uses a component approach that is platform-independent. EJB splits the business logic and the presentation logic in separate tiers.

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 to structure the application logic into enterprise beans. Tools are available, however, that automate the generation of large portions of the EJB code.

In terms of performance, PHP scripts are more efficient than Java servlets. PHP scripts are, however, tied to the Web server and provide limited functionality and runtime support. Java servlets run in a different process from the Web server. This flexibility can be exploited to off-load the servlets to another machine to give better performance when the Web server is the bottleneck. Servlets can use all the Java language features and runtime support. In particular, if servlets are the only application that accesses the database, they can offload some locking operations from the database. This improves performance if there is database lock contention. Enterprise Java Beans offer the most flexible architecture. The EJB server offers many services to the enterprise beans, which capture the application logic. Using EJB represents a trade-off: Expressing the application logic in terms of enterprise beans offers important software engineering qualities such as modularity, portability, and maintainability, but the performance of EJB is lower than both Java servlets and PHP.

References

1. Rahim Adatia et al. – Professional EJB – *Wrox Press, ISBN 1-861005-08-3*, 2001.
2. Deepak Alur, John Crupi and Dan Malks – Core J2EE Patterns – *Sun Microsystems Press, ISBN 0-13-064884-1*, 2001.
3. Cristiana Amza, Emmanuel Cecchet, Anupam Chanda, Alan Cox, Sameh Elnikety, Romer Gil, Julie Marguerite, Karthick Rajamani and Willy Zwaenepoel – Specification and Implementation of Dynamic Web Site Benchmarks – *IEEE 5th Annual Workshop on Workload Characterization (WWC-5)*, Austin, TX, USA, November 2002.
4. The Apache Software Foundation – http://www.apache.org/.

5. 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.
6. Emmanuel Cecchet, Julie Marguerite and Willy Zwaenepoel – Performance and scalability of EJB applications – *Proceedings of OOPSLA'02*, 2002.
7. eBay – http://www.ebay.com/.
8. Jason Hunter and William Crawford – Java Servlet Programming 2nd edition – *O'Reilly, ISBN 0-596-00040-5*, 2001.
9. JOnAS Open Source EJB Server – http://www.objectweb.org.
10. Lutris Enhydra Application Server – http:/www.lutris.com.
11. Microsoft Active Server Pages – http://www.asp.net.
12. Microsoft Internet Information Server – http://www.microsoft.com/iis.
13. MySQL Reference Manual v3.23.36 – http://www.mysql.com/documentation/.
14. PHP Hypertext Preprocessor – http://www.php.net/.
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. SPECjAppServer2002 Design Document - http://www.specbench.org/jAppServer2002/docs, 2002.
18. Sysstat package – http://freshmeat.net/projects/sysstat/.
19. Jakarta Tomcat Servlet Engine – 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.
22. Sun Microsystems – Enterprise Java Beans Specifications – http://java.sun.com/j2ee/.