Bottleneck Characterization of Dynamic Web Site Benchmarks

Cristiana Amza, Emmanuel Cecchet, Anupam Chanda, Alan L. Cox, Sameh Elnikety, Romer Gil, Julie Marguerite, Karthick Rajamani and Willy Zwaenepoel
Department of Computer Science
Rice University

Abstract

The absence of benchmarks for Web sites with dynamic content has been a major impediment to research in this area. We describe three benchmarks for evaluating the performance of Web sites with dynamic content. The benchmarks model three common types of dynamic-content Web sites with widely varying application characteristics: an online bookstore, an auction site, and a bulletin board. For each benchmark we describe the design of the database, the interactions provided by the Web server, and the workloads used in analyzing the performance of the system.

We have implemented these three benchmarks with commonly used open-source software. In particular, we used the Apache Web server, the PHP scripting language, and the MySQL relational database. Our implementation is therefore representative of the many dynamic content Web sites built using these tools. Our implementations are available freely from our Web site for other researchers to use.

We present a performance evaluation of our implementations of these three benchmarks on contemporary commodity hardware. Our performance evaluation focused on finding and explaining the bottleneck resources in each benchmark. For the online bookstore, the CPU on the database was the bottleneck, while for the auction site and the bulletin board the CPU on the front-end Web server was the bottleneck. In none of the benchmarks was the network between the front-end and the back-end a bottleneck. With amounts of memory common by today's standards, neither the main memory nor the disk proved to be a limiting factor in terms of performance for any of the benchmarks.

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. A number of scripts executing in the Web server provide access to that content. The client sends an HTTP request to the Web server containing the URL of the script and some parameters. The Web server executes the script, which issues a number of queries to the database and formats the results as an HTML page. This page is then returned to the client as an HTTP response.

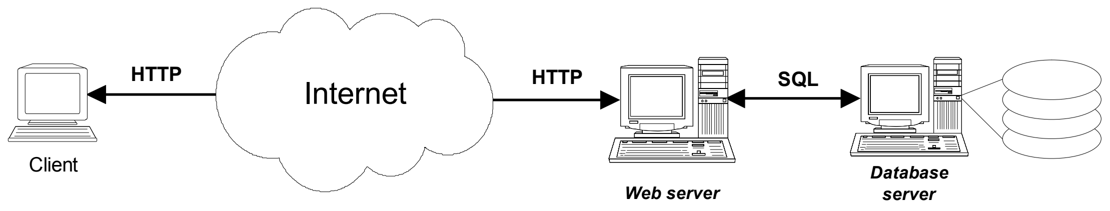

Figure 1. Typical Configuration of a Dynamic Content Web Site

To study the architecture and the performance of dynamic Web sites, benchmarks are needed that are representative of some of the common applications for such Web sites, yet simple enough to be understood and manipulated with ease. TPC-W [18] provides a specification for benchmarking e-commerce applications. It models an online bookstore, such as amazon.com. This paper proposes new specifications for two different types of dynamic content sites: auction sites and bulletin boards. Our benchmark for auction sites is modeled after eBay [8]. For bulletin boards, our benchmark is modeled after the Slashcode [16], used in many bulletin board sites, including its originator site Slashdot [15].

We describe the specification and the implementation of each of the benchmarks and the workloads we use to evaluate the benchmarks. Our implementations use three popular open-source Web server development packages: the Apache [1] Web server, the PHP [12] Web-scripting language, and the MySQL [9] relational database. This software setup has become one of de facto standards on the Web. The most recent Netcraft survey [10] showed that 61% of all Web sites are running Apache. About 40% of these sites had the PHP module compiled in. While we recognize that different software platforms may lead to different performance characteristics, our implementation and the resulting performance results are representative of what can be achieved by a software platform in frequent use. The choice of an open-source platform also allows easy use of our benchmarks by other researchers. To further this goal, we have made the source code of our implementation of the benchmarks available on our Web site.

We then used these benchmarks to assess the bottlenecks for servers implementing online stores, auction sites and bulletin boards. In all our experiments, the Web server and the database ran on a separate machine. In particular, we used a 1.33GHz AMD Athlon with 768MB memory and a 60GB disk for each machine. The two machines were connected to each other and to a set of machines running client emulation software by a switched 100Mbps Ethernet. For the online bookstore the CPU on the database server is the bottleneck. In contrast, for the auction site and the bulletin board the Web server CPU is the bottleneck. In none of the experiments we found the memory, the disk or the network to be a bottleneck. We also comment on the effect of enforcing various degrees of (transactional) consistency in the benchmarks.

The rest of this paper is structured as follows. Sections 2 to 4 describe the benchmark specifications, our implementations, and the workloads used for each of them. Section 5 describes our experimental environment, both in terms of software and hardware. Sections 6 to 8 discuss the results for each of the benchmarks. We cover related work in section 9, and conclude in section 10.

2. Online Bookstore Benchmark

The TPC-W benchmark from the Transaction Processing Council [18] 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) which represents an external system that authorizes payment of funds during purchasing interactions [18, 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 use two database sizes, a large one (3.5GB) and a small one (350MB) that fits entirely in memory on our experimental platform. These sizes include the necessary database indices.

3. 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 [8]). 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.

4. Bulletin Board Benchmark

Our bulletin board benchmark is modeled after an online news forum like Slashdot [15]. We originally considered using the Perl-based Slashcode [16], which is freely available, but we concluded that the code was too complex to serve as a benchmark and that possible performance differences between Perl and PHP, used in our other benchmarks, would make it too difficult to draw general conclusions about all benchmarks. Instead, we implemented the essential bulletin board features of the Slashdot site in PHP. In particular, as in Slashcode, we support discussion threads. A discussion thread is a logical tree, containing a story at its root and a number of comments for that story, which may be nested. Users have two different levels of authorized access: regular user and moderator. Regular users browse and submit stories and comments. Moderators in addition review stories and rate comments.

The main tables in the database are the users, stories, comments, and submissions tables. The users table contains each user's real name and nickname, contact information (email), password, level of authorized access, and rating. The stories table contains each story's title and body, the nickname of the story's author, the date the story was posted, the number of comments at the outermost nesting level, and the category the story fits under. The categories table contains the same categories as the Slashdot site. The comments table contains the comment's subject and body, the nickname of the comment's author, the date the comment was posted, the identifier of the story or the parent comment it belongs to, and a comment rating. Each submitted story is initially placed in the submissions table, unless submitted by a moderator. We maintain a moderator_log table, which stores the moderator ratings for comments. Regular user ratings are computed based on the ratings of the comments they have posted.

For efficiency reasons, we split both the stories and comments tables into separate new and old tables. In the new stories table we keep the most recent stories with a cut-off of one month. We keep old stories for a period of two years. The new and old comments tables correspond to the new and old stories respectively. The majority of the browsing requests are expected to access the new stories and comments tables, which are much smaller and therefore much more efficiently accessible. A daemon is activated periodically to move stories and comments from the new to the old tables as appropriate.

We have defined 24 Web interactions. The main ones are: generate the stories of the day, browse new stories, older stories, or stories by category, show a particular story with different options on filtering comments, search for keywords in story titles, comments and user names, submit a story, add a comment, review submitted stories and rate comments at the moderator level. Full text search is currently not supported. Without additional support, it requires a prohibitive processing time in a general-purpose relational database. Typically, an external search engine would be used to perform this task.

We use two workload mixes: a browsing mix and a submission mix. The browsing mix is a read-only workload that does not allow users to post stories or comments. The submission mix contains 85% read-only interactions, with the remaining 15% being story and comment submissions and moderation interactions.

We generate the story and comment bodies with words from a given dictionary and lengths between 1KB and 8KB. Short stories and comments are much more common, so we use a Zipf-like distribution for story length [3]. The database contains 2 years of stories and comments. We use an average of 15 to 25 stories per day and between 20 and 50 comments per story, as we observed on Slashdot. We emulate 500,000 total users, out of which 10% have moderator access privilege. With these parameters, the database size is 439MB. We also created a larger database of 1.4GB containing more old stories and comments. The results are very similar as the majority of the requests access the new stories and comments.

5. Hardware and Software Environment

5.1. 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 were the bottleneck.

5.2. 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. We use MySQL v.3.23.43-max as our database server. We use MyISAM and BDB tables as non-transactional and transactional database tables, respectively. All machines run the 2.4.12 Linux kernel.

In MySQL, as soon as overload occurs, performance degrades sharply. Moreover, performance degrades faster when the number of update queries in the workload becomes larger. Transactions are a new and relatively less stable addition to MySQL. Transactions exacerbate the performance degradation under heavy contention. The effects of these limitations in MySQL will be seen in several experiments reported in sections 6 to 8.

5.3. Transactional semantics

We study two levels of transactional consistency.

The first level provides transactional isolation, but does not provide for transactional durability and atomicity. For all interactions (PHP scripts) that contain update queries, we insert database lock operations that obtain all locks necessary for the queries in the script (for both read and write operations) before the first query. Scripts that contain only read-only queries do not obtain locks. Unless mentioned otherwise, this level of consistency is the default used in our experiments.

The second level provides full ACID transaction guarantees. BDB tables are extensions to MySQL that support full transaction semantics. For all interactions (PHP scripts) we insert a begin_transaction before the first database query is issued and a end_transaction after the last query is issued. If a script fails, an abort can be issued to release any held database locks and undo all modifications to the database. We refer to this second level of consistency as "with transactions" in the experiments in sections 6 to 8.

5.4. 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.

5.5. 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 30 seconds for the warm-up, the steady-state and the cool-down phase, respectively. The auction site uses 5, 30 and 5 minutes, and the bulletin board 2.5, 15 and 2.5 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 [17] 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.

6. Bottleneck Analysis for Online Bookstore Benchmark

Figure 2 shows the throughput, in interactions per minute, as the number of clients increases, for each of the three workload mixes and for the small database. The peak throughputs are 356, 515, and 654 interactions per minute, for the browsing, shopping, and ordering mix, respectively. Figure 3 to figure 5 show, for the different mixes, the average CPU utilization on the Web server and the database server as the number of clients increases.

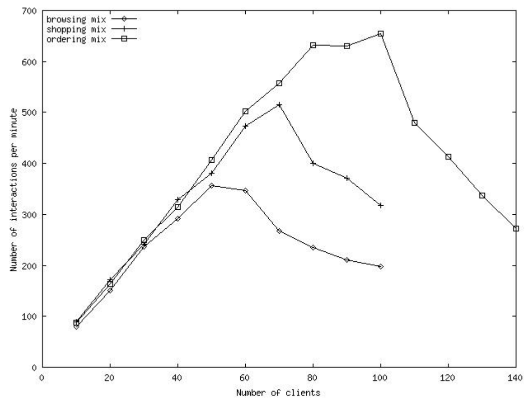

Figure 2. Online bookstore throughput in interactions per minute as a function of number of clients.

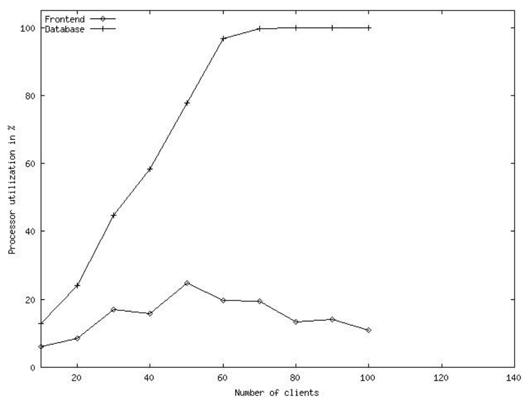

Figure 3. Online bookstore percentage CPU utilization as a function of number of clients for the browsing mix.

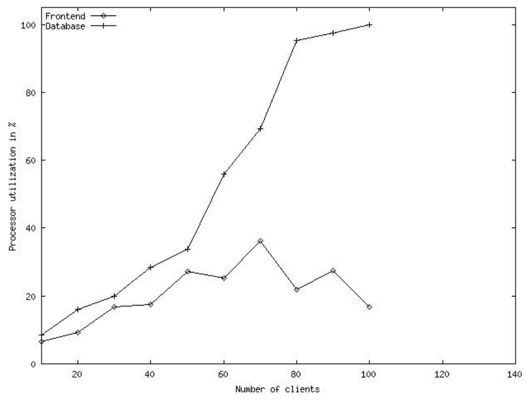

Figure 4. Online bookstore percentage CPU utilization as a function of number of clients for the shopping mix.

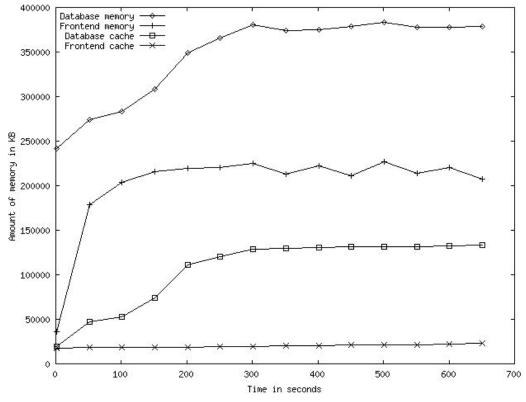

Figure 6. Online bookstore memory usage in KB as a function of time at the peak throughput for the ordering mix.

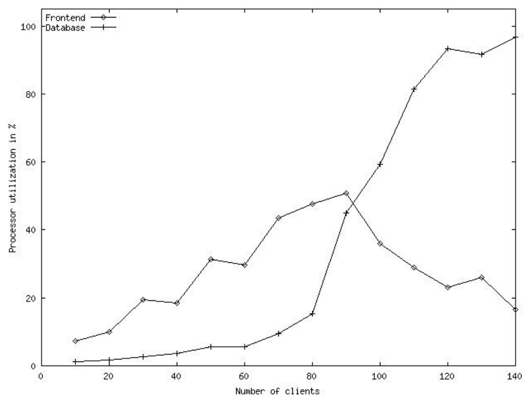

Figure 5. Online bookstore percentage CPU utilization as a function of number of clients for the ordering mix.

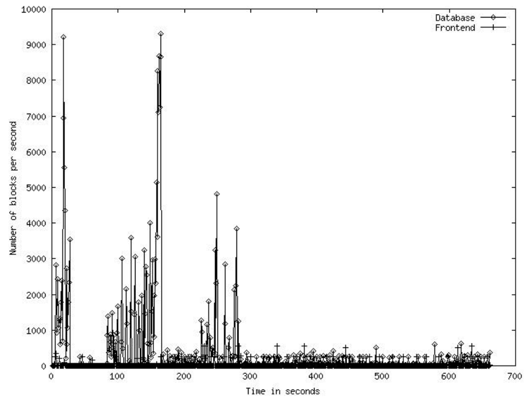

Figure 7. Online bookstore disk usage in number of blocks per seconds as a function of time at the peak throughput for the ordering mix.

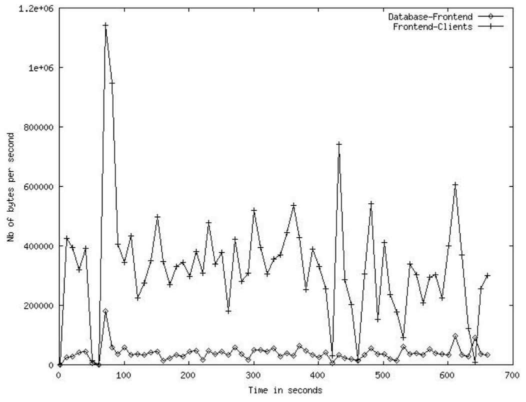

Figure 8. Online bookstore network usage in bytes/s as a function of time at the peak throughput for the ordering mix.

From these figures we conclude that for all workload mixes, the CPU on the database machine is the bottleneck resource at the peak throughput. The complex nature of many of the database queries makes the database the bottleneck. In comparison, the cost of handling and executing the

PHP scripts for these interactions on the Web server is small. The read-only queries are, on average, more complex than the read-write queries. Hence, for workload mixes with a larger number of read-only queries, overall throughput is lower and the database is more of a bottleneck.

We monitor the memory usage and disk access on the Web server and the database throughout all our experiments. None of these resources is the bottleneck. Figure 6 and figure 7 show the utilization of memory and disk for the ordering mix at its peak throughput, which is also the highest throughput of any of the three mixes. During a short initial transient period, the database reads information from the disk to warm up its cache. After this period, the working set fits in memory and hence disk access is low. Memory utilization in steady state is approximately 200MB on the Web server and 390MB on the database.

Figure 8 shows the network usage between the database and the Web server, and between the Web server and the clients. The latter is on average 3.2Mb/s, while the former is always lower than 1.6Mb/s. Clearly, neither of these forms a bottleneck.

As expected, when we add full transaction semantics, the throughput for all mixes is lower; in particular, the peak throughputs are 240, 395 and 191 interactions per minute for the browsing, shopping and ordering mix, respectively. The database CPU remains the bottleneck. When we use the larger database, the disk utilization on the database server becomes higher, but the database CPU remains the bottleneck. We obtain peak throughputs of 56, 120, and 494 interactions per minute, for the browsing, shopping, and ordering mix, respectively. Compared to the small database, the performance for the larger database drops much more significantly for the workloads that have a higher read-only component. Reads become a lot more expensive, because they go to disk much more often, while the cost of writes remains roughly the same. Due to the limitations of MySQL, we could not get reproducible results for the large database and full transaction semantics.

7. Bottleneck Analysis of Auction Site

Figure 9 shows the number of interactions per minute for each workload as a function of the number of clients. The peak throughput for the browsing mix is 8,520 interactions per minute with 800 clients, while the bidding mix achieves a peak of 9,780 interactions per minute with 1,100 clients. Figure 9 also shows the throughput using transactions on the database server. The browsing mix shows throughput comparable to the throughput obtained without transactions up to 600 clients, and then peaks at the slightly lower value of 7,740 interactions per minute with 800 clients. Due to limitations of MySQL, discussed in section 5, we were not able to obtain results for the bidding mix with transactions using a suitable number of clients.

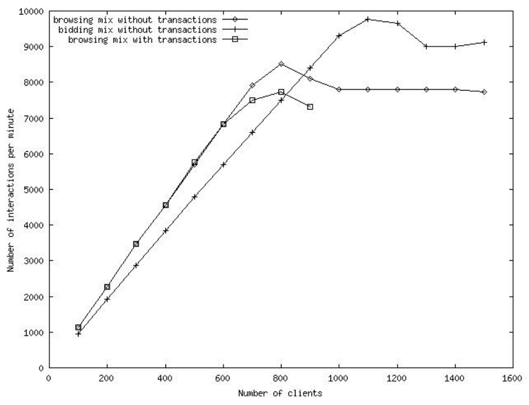

Figure 9. Auction site throughput in interactions per minute as a function of number of clients.

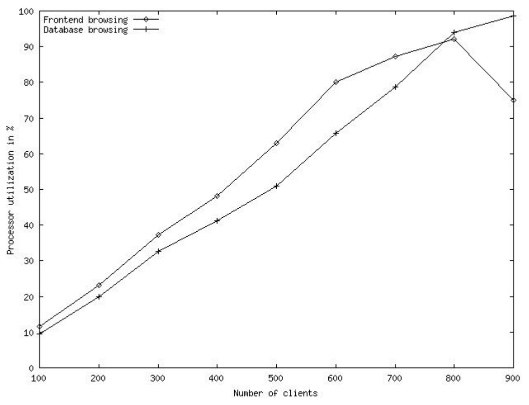

Figure 12. Auction site percentage CPU utilization as a function of number of clients for the browsing mix with transactions.

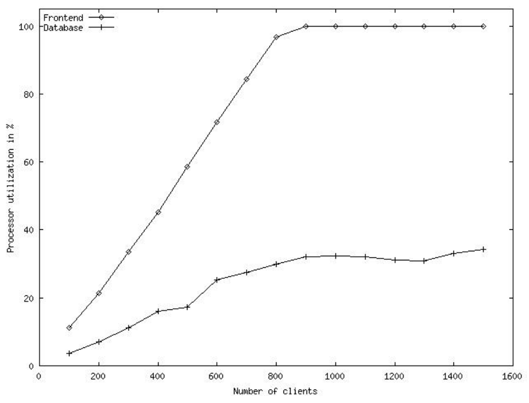

Figure 10. Auction site percentage CPU utilization as a function of number of clients for the browsing mix.

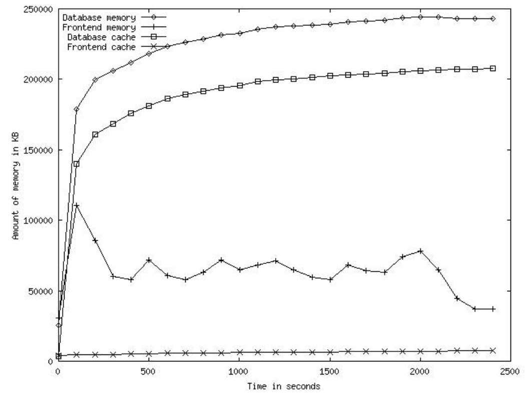

Figure 13. Auction site memory usage in KB as a function of time at the peak throughput for the browsing mix.

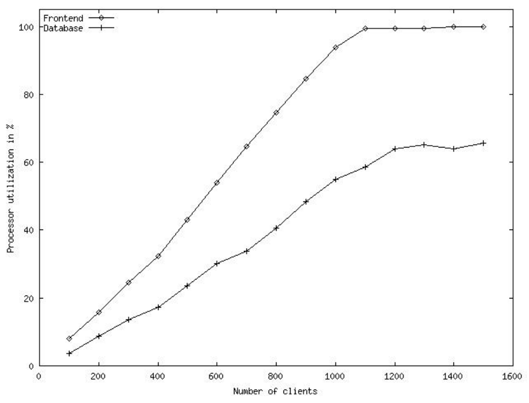

Figure 11. Auction site percentage CPU utilization as a function of number of clients for the bidding mix.

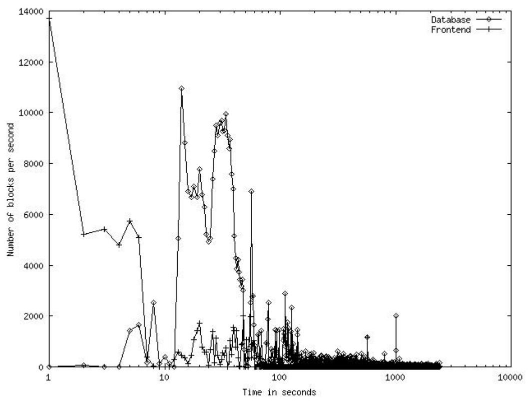

Figure 14. Auction site disk usage in number of blocks per second as a function of time (x-axis is log-scale) at the peak throughput for the browsing mix.

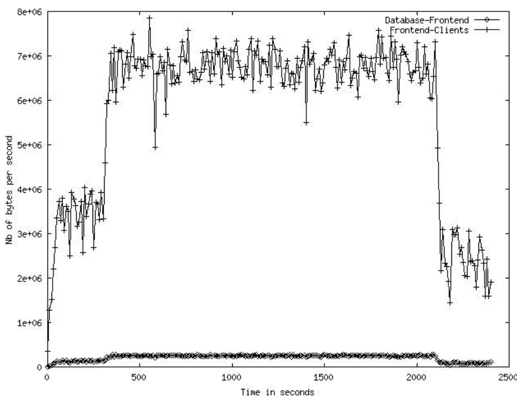

Figure 15. Auction site network usage in bytes/s as a function of time at the peak throughput for the browsing mix.

Figure 10 and figure 11 show the CPU utilization for the browsing and bidding mix, respectively. CPU utilization increases linearly with the number of clients on both the Web server and the database server, but it increases much more rapidly on the Web server. CPU utilization on the Web server at the peak throughput point is 100% for both workload mixes. On the database server, CPU utilization at the peak throughput is 58% for the bidding mix and 29% for the browsing mix. Figure 12 shows the CPU utilization for the browsing mix with transactions. CPU utilization grows linearly with the number of clients, again faster on the Web server than on the database, but only very slightly so. At the peak point, both database and Web server CPU are saturated. With even more clients, the situation reverses and the CPU on the database becomes the bottleneck. Given the poor overload behavior of MySQL (see section 5), the CPU utilization on the Web server drops.

Memory and disk usage on the Web server and the database server are reported in figure 13 and figure 14, respectively. On the Web server machine, at the beginning of the experiment, a lot of Web server processes are created and the scripts are read from the disk. This explains the initial disk activity and the sharp rise in memory use. After this startup phase, there is little disk activity and memory usage remains constant at a modest 70 MB. A similar phenomenon occurs on the database server. When the first requests start arriving at the database, there is a lot of disk activity, and memory use increases rapidly, until the point in time at which most of the working set (indices and frequently used tables) is in memory. After that, disk usage falls off, and memory usage remains stable at around 250MB, a relatively small value and certainly within reasonable bounds for a server machine. Although the database itself is relatively large, the temporal locality of the information in the database causes the working set to be relatively small.

Figure 15 shows the network utilization at the peak point between the clients and the Web server, and between the Web server and the database server, as a function of time. During steady state, the bandwidth between the clients and the Web server is about 55Mb/s, while the bandwidth between the Web server and the database server is about 2Mb/s. Therefore, network bandwidth is never the bottleneck for this application.

In summary, the Web server CPU is the bottleneck resource for the auction site. With transactions, however, both servers are saturated at the peak throughput for the browsing mix. We would expect that using a more complex business logic or adding features like user preferences to customize the look-and-feel of the pages sent to clients would further increase Web server CPU load. Possible solutions to this bottleneck include using an SMP or a cluster as the Web server.

We have experimented with a dual-processor node for the Web server, which was sufficient to make the database CPU the bottleneck for the bidding mix.

8. Bottleneck Analysis of Bulletin Board

Figure 16 presents the throughput in number of interactions per minute for the browsing and submission mixes as a function of the number of clients. The browsing mix peaks at 8,160 interactions per minute with 900 clients, the submission mix at 8,580 interactions per minute with 1,000 clients. Transactions are rarely used in connection with bulletin board sites, so we do not report results with full transactional semantics for this application.

Figure 18 and figure 19 show that for both the browsing and the submission mix, the Web server CPU is the bottleneck resource at the peak throughput point. For the submission mix, the situation reverses, however, just after the peak point. MySQL's scaling limitations cause a jump from 52% database CPU utilization for 900 clients to 100% with 1,100 clients. The comment table is the main bottleneck as this large table is involved in most of the browsing and update requests.

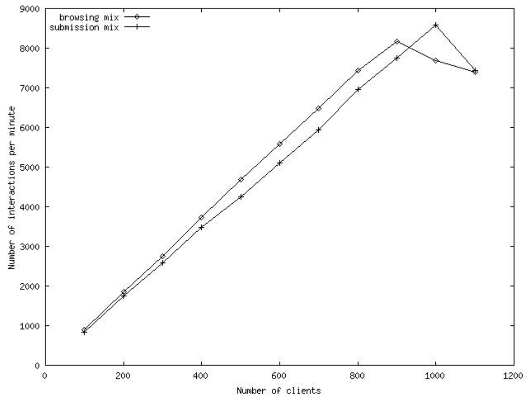

Figure 16. Bulletin board throughput in interactions per minute as a function of number of clients.

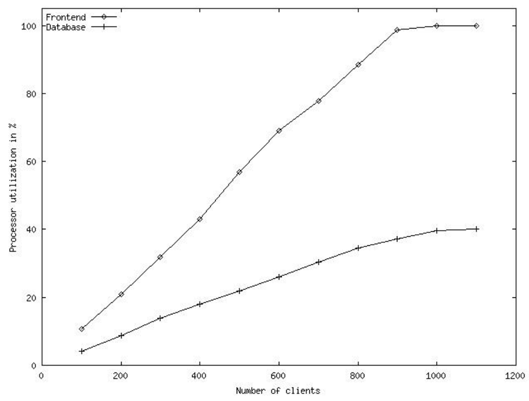

Figure 18. Bulletin board percentage CPU utilization as a function of number of clients at the peak throughput for the browsing mix.

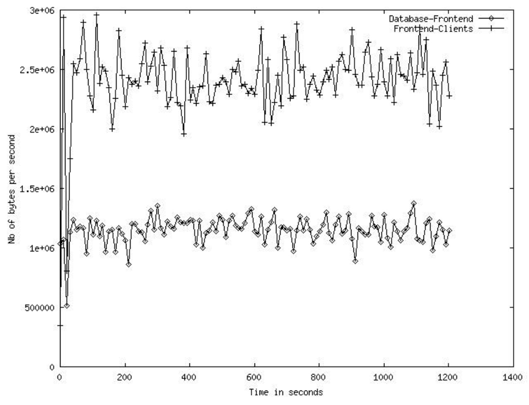

Figure 17. Bulletin board network usage in bytes/s as a function of time at the peak throughput for the browsing mix.

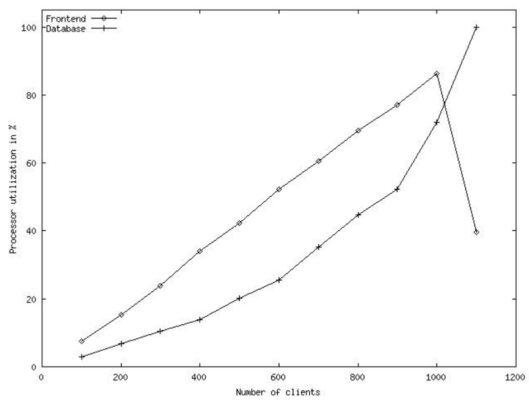

Figure 19. Bulletin board percentage CPU utilization as a function of number of clients at the peak throughput for the submission mix.

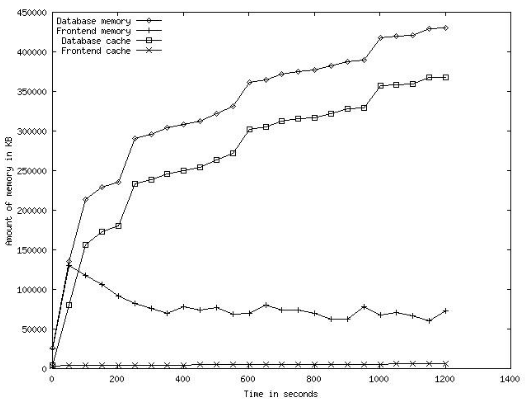

Figure 20. Bulletin board memory usage in KB as a function of time at the peak throughput for the browsing mix.

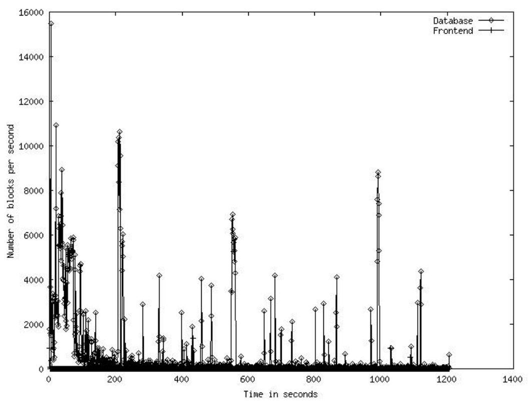

Figure 21. Bulletin board disk usage in number of blocks per second as a function of time at the peak throughput for the browsing mix.

Figure 20 and figure 21 report on memory and disk usage. As with the auction site, we observe a burst of reads at startup. With the auction site, clients only access new items, and therefore the working set is limited to the new items and fits in memory. With the bidding site, however, clients also continue to access old stories and comments. The disk reads after the initial startup are largely due to accesses to old stories. The memory utilization increases correspondingly. Due to the larger amount of data manipulated, the database server requires an average of 350MB, while Web server memory usage remains modest at 70MB.

Figure 17 shows that the network traffic between the database and the Web server is much higher than for the other sites (9Mb/s on average). The network traffic between the clients and the Web server is 20Mb/s. In any case, the network bandwidth is not a bottleneck.

To summarize, the Web server CPU is the bottleneck resource at the peak point for the bulletin board, for both workload mixes. The database CPU approaches saturation as well at peak throughput and becomes the bottleneck if even more clients are added. A possible approach to offloading the database is to generate static pages for the stories of the day or the most recent stories. We experimented with this technique, and found that it made the Web server the bottleneck under all circumstances.

9. Related Work

For static Web content, the presence of a number of agreed upon benchmarks, such as, e.g., the NLANR traces [11] and the Polygraph benchmark [19], have greatly fostered research in systems support for static Web content servers, including OS support, caching, and clustering. Similar studies for dynamic Web content sites have been far fewer, and their results much more difficult to compare, in our opinion, in part because of the lack of benchmarks. Zhang et al. [20] studied load balancing among machines in a dynamic content Web server cluster, but their study uses a read-only workload, avoiding any issues of consistency maintenance in the presence of writes. Ji et al. [5] used simulation of a white pages server and an auction site to study caching of database results. Our benchmarks allow measurement of system overheads on real systems. The Neptune project [13] studies scalability of clusters for dynamic content, but does not include a bottleneck analysis like the one presented in this paper.

Menascé et al. modeled client workloads for dynamic content sites [6]. Starting from access logs of an actual e-business site (an auction site that sells domain names), they have developed detailed models of customer behavior, and resource management methods to optimize site revenue [7]. For the online bookstore, we adopted the workload from the one specified by TPC-W. For the other applications, we have adopted similar workload models. It would be an interesting exercise to evaluate our benchmarks with their workloads.

Java servlets are an alternative to PHP for expressing the business logic of an application. Cain et al. [2] presented 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 identifying bottleneck resources for a variety of dynamic content applications, while their study is aimed at evaluating architectural features for a Java servlet implementation of TPC-W. An interesting avenue for further work would be to evaluate the impact of the choice of scripting language, as well as the choice of different Web servers and database engines, on the question of which resource is the bottleneck for a particular application.

10. Conclusions

We have presented three benchmarks for dynamic content sites with very different characteristics: an online bookstore, an auction site, and a bulletin board. For the online bookstore, we followed the specification provided by TPC-W. For the auction site and the bulletin board, we provided our own specifications. The implementations of the benchmarks use open-source software in common use on the Internet, thereby making our results representative for the many sites using these software packages.

We have used our implementations to carry out a bottleneck characterization of the benchmarks. Different benchmarks show different bottlenecks: the database CPU for the online bookstore, and the Web server CPU for the auction site and the bulletin board. Complex queries cause the database CPU to be the bottleneck for the online bookstore. In contrast, the queries for the other applications are simpler.

We are making the source code of our implementations freely available on our Web site. We hope other researchers will use them, making performance results of dynamic content Web sites more reproducible and easier to compare. In our own further work, we intend to use these benchmarks for studying caching, clustering and replication for dynamic content generation, and for comparing different software platforms for building dynamic content sites.

References

[1] The Apache Software Foundation – http://www.apache.org/
[2] 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.
[3] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker - Web Caching and Zipf-like Distributions: Evidence and Implications – *Proceedings of the IEEE Infocom Conference*, 1999.
[4] Jim Challenger and Arun Iyengar and Paul Dantzig – A Scalable System for Consistently Caching Dynamic Web Data - *Proceedings of IEEE INFOCOM'99*, 1999.
[5] Minwen Ji, Edward W. Felten, Jaswinder Pal Singh and Mao Chen – Query Affinity in Internet Applications – *Computer Science Technical Report*, Princeton University, 2001
[6] Daniel Menascé, Flavia Ribeiro, Virgilio Almeida, Rodrigo Fonseca, Rudolf Riedi and Wagner Meira Jr – In Search of Invariants for E-Businness Workloads – *Proceedings of EC'00*, 2000.

[7] Daniel Menascé, Rodrigo Fonseca, Virgilio Almeida and Marco Mendess – Resource Management Policies for E-commerce Servers – *Second Workshop on Internet Server Performance WISP'99,* 1999.
[8] eBay – http://www.ebay.com/.
[9] MySQL Reference Manual v3.23.36 – http://www.mysql.com/documentation/.
[10] Netcraft Web Server Survey, October 2001 – http://www.netcraft.com/survey/.
[11] NLANR – http://pma.nlanr.net/Traces/.
[12] PHP Hypertext Preprocessor – http://www.php.net/.
[13] Kai Shen, Tao Yang, Lingkun Chu, JoAnne L. Holliday, Doug Kuschner, Huican Zhu – Neptune: Scalable Replica Management and Programming Support for Cluster-based Network Services – *3rd USENIX Symposium on Internet Technologies and Systems (USITS),* 2001.
[14] Karthick Rajamani and Alan Cox – A Simple and Effective Caching Scheme for Dynamic Content - *Computer Science Technical Report 00-371, Rice University*, 2000.
[15] Slashdot – http://www.slashdot.org/.
[16] Slashcode – http://www.slashcode.org/.
[17] Sysstat package – http://freshmeat.net/projects/sysstat/.
[18] Transaction Processing Performance Council– http://www.tpc.org/.
[19] Web Polygraph – http://www.web-polygraph.org.
[20] Xiaolan Zhang, Michael Barrientos, J. Bradley Chen and Margo Seltzer – HACC: An Architecture for Cluster-Based Web Servers – *Proceedings of the 2000 Annual Usenix Technical Conference*, 2000.