# Electronic Commerce On the World-Wide Web: Performance and Availability

David Finkel, Robert E. Kinicki, Mikhail Mikhailov, Aditya Raghavendra
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609 USA
dfinkel@cs.wpi.edu

Sharon Cunningham, Yuriy Elkin, Arthur Lin, Mark Quinlivan,
Joel Sommers, Bob Wescott
Stratus Computer, Inc.
55 Fairbanks Boulevard
Marlboro, MA 01752 USA

## Abstract

This paper describes a continuation of an effort to study the relationship among performance, security, and availability in a Web-based electronic commerce system. In the first phase of the study, we examined the relationship between performance and security, and characterized the performance costs of various encryption approaches.

In the second phase of the project, we built a three-tier testbed electronic commerce system, using a cluster of servers at each tier. We characterize the performance of the system under different configurations and under different workloads. We demonstrate the performance advantages of a clustered design, and also discuss the improvements in availability provided by clustering.

The paper also discusses some implementation issues, and describes changes in implementation between the first and second phases of the project, and the effect of those changes on performance.

**Keywords:** World-Wide Web, Electronic Commerce, Clustering, Performance

## 1. Introduction

This paper describes the results of a the second phase of a joint research effort undertaken by Stratus Computer, Inc. [1] and researchers from the Computer Science Department of Worcester Polytechnic Institute. The first phase of this project was described in [2]. The goals of the first phase were to construct an Web electronic commerce site, and to measure the performance implications of using encryption to create a secure site. A testbed electronic commerce Web site was constructed, implementing a simulated stock brokerage application.

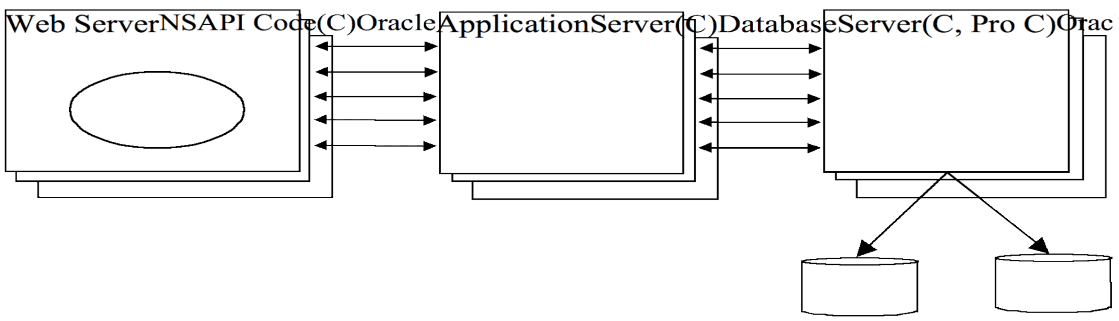
The goals of the second phase were to understand the relationship among the requirements of security, performance, and availability. Again, a stock brokerage application was implemented. As in the first phase, we built a three-tiered application: Web server, application server, and database server. In this phase, we added clusters of servers for each tier, as shown in Figure 1. The goal of the clustering was to improve performance and to improve availability [3]. Performance was improved by having the machines in a cluster share the workload. Availability was improved by having machines in one cluster monitor the machines in another cluster. When a server failure was detected, work that had been submitted to the failed server was re-submitted to a different server in the cluster.

We tested the performance of our system by measuring the number of transactions per second that could be sustained by the system for different numbers of clients and different cluster configurations.

## 2. Operation of the Testbed System

The testbed system operated as follows. A client driver program, running on Pentium PCs running the Linux operating system, generated transaction requests typical of transactions submitted by a user of a stock brokerage Web site. Different scripts of transactions were used to simulate different levels of user load.

The requests from the client driver were received by one of the Web servers. A Web server application, written using the Netscape Server API (NSAPI) [4], provided the necessary processing. The Web server forwarded the client request to one of the application servers; if the Web server did not receive a reply before a timeout expired, it would re-submit the client request to a different application server.



The application server applied business rules to the request, and submitted an appropriate database request to a database server. When the application server received a reply from the database server, it constructed a response in the form of an HTML page, and returned that response to the Web server , which then returned the response to the client.

The communications between the client machines and the Web server was encrypted by public key encryption, using SSL (Secure Socket Layer). The communications between the Web servers and the applications servers, and between the application servers and the database servers, used TCP/IP sockets, and was encrypted using private key encryption with the IDEA cipher [5].

## 3. Results

The principal goal of this phase of the project was to understand the scalability of our system architecture: that is, how performance would be affected by the use of multiple servers for each tier of our system. We performed extensive tests measuring the response time and the number of transactions per second for different levels of intensity of requests, and different server configurations.

Figure 2 gives an overview of the results of our testing. The figure shows the maximum throughput achieved in our testing under different testbed configurations. Throughput is measured in transactions per second. The default configuration had one Web server, one application server, and one database server. During the testing, the configuration was modified to include from one to four Web servers (1 WS to 4 WS), and then finally to four Web servers and two database servers (4WS, 2DBS). We see a nearly linear increase in throughput as the number of Web servers is increased, showing that our system design provides good scalability over the range of workloads tested. In addition, since we see an increase in performance when we change the configuration from four Web servers and one database server (4WS) to four Web servers and two database servers (4WS, 2DBS), we see that, in this configuration, the Web servers are not a bottleneck.

Figures 3 and 4 provide a more detailed look at our performance results. Both figures present graphs of the number of transactions per second vs. the offered load, for several system configurations. The configurations are as described for Figure 2. The offered load is the total number of threads executing on client machines that are sending requests to the Web servers; this is meant to represent varying the number of users connected to the Web site. For each configuration, we increased the offered load until the number of transactions per second leveled off, indicating that we had saturated the electronic commerce system.

The two graphs represent different transaction mixes; that is, the clients were programmed to submit different mixes of transaction requests to the Web server. Figure 3 is labeled “Simple Transaction Mix”, and this mix did not place a heavy load on the database server of the system. The transactions in this mix required that the database server consult at most a few records in the database. Figure 4 did place a heavy load on the database server. One of its transactions required retrieving the records for all the securities stored in the database.

We can see the difference between the two transaction mixes by examining Figures 3 and 4. In Figure 3, we see a significant increase in performance as we add Web servers to the configuration. In addition, we see very little increase in performance when we add a second database server to the configuration. These results indicate that the Web servers are the bottleneck in these configurations. As we add more Web servers, we relieve the bottleneck, and performance increases until the new configuration is saturated. Adding a database server does not improve performance significantly, because the database is not the bottleneck device. The Web servers are still limiting the performance of the system.

The situation is significantly different in Figure 4. There, the workload on the database server is more significant, and as a result the Web servers are no longer the bottleneck device. Although performance increases as we add Web servers, the performance increase is less than in Figure 3. This difference is due to the fact the Web server is no longer the bottleneck, and other components of the system are limiting performance. Finally, when we add a second database server, we see a more significant performance increase than we saw in Figure 3, indicating that the database server is now a more significant bottleneck than in Figure 3.

We also note that our system running the Complex Transaction Mix in Figure 4 achieves significantly higher throughput than with the Simple Transaction Mix of Figure 3. Both the transactions per second for a given offered load, and the maximum transactions per second achieved for a given configuration are higher for Figure 4 than for Figure 3. This result is again due to the different nature of the workloads in these two cases. Since the Simple Transaction Mix places most of its load on the Web servers, the performance in this case is limited by the performance of the Web servers. In the Complex Transaction Mix, the load is more evenly shared by the different tiers of the system (Web server, application server, database server), and so there is a greater degree of concurrency in the system, and a greater throughput of transactions.

We close this section with some observations about the implementation of our system, and some of the lessons we learned.

We needed to write our own programs to run on the Web servers. After a request was received by the Web server, our programs had to manage the communications with the application server. In the first phase of this project (see [2]), this programming was done using the Common Gateway Interface (CGI), with programs written in C. With CGI programs, a new process is started to handle each transaction. As a result, there is significant overhead for starting and stopping processes. In addition, since each CGI program was a new process, each one had to open a new socket to communicate with the application server, another significant source of overhead.

Based on what we learned in the first phase, in this second phase we changed the approach to programming on the Web server. We wrote our Web server programs using the Netscape Server API. We again wrote our programs in C, but used libraries provided by Netscape (see [4]). As a result, our programs ran in the Web server's process space, and so no starting and stopping of processes was necessary. In addition, we were able to create a pool of socket connections to the application servers when our Web server started running, and so we did not have to create connections for each transaction. A comparison of the Web server performance in phase one and phase two show a significant improvement in the throughput of the Web servers in phase two resulting from our change in programming approach.

The disadvantage of using the Web server API for programming is that the resulting programs are not portable. If we ever decided to change to a different Web server, we would have to completely re-write our Web server programs.

## 4. Conclusions and Future Work

Our testbed system demonstrates the value of the cluster approach to organizing the servers in our Web-based electronic system. We evaluated the performance of the systems under different configurations and different transaction workloads. We identified bottlenecks in our system, and showed that providing additional servers to the bottleneck component led to a significant improvement in performance.

We also demonstrated the usefulness of clustering in providing increased reliability. Through the use of timeouts and re-submission of timed-out transactions, our system can withstand failures in any of the components where there are multiple servers. Naturally, performance would degrade after such a failure. Further testing is required to quantify the effect on performance.

In addition, future work includes testing the availability of our system under a variety of failure situations, using an availability testing process described in [6].

## References

1. Stratus Computer, Inc., http://www.stratus.com/, 1998.

2. Bob Kinicki, David Finkel, Mikhail Mikhailov, Joel Sommers, Sharon Cunningham, Yuriy Elkin, Ranga Gopalan, Mark Quinlivan, "Electronic Commerce Performance Study", *Proceedings of Euromedia '98*, 1998, 26 - 34.

3. Gregory F. Pfister, *In Search of Clusters*, Second Edition, (Saddle River, NJ: Prentice-Hall, 1998).

4. Tony Beveridge and Paul McGlashan, *High Performance ISAPI/NSAPI Web Programming*, (Scottsdale, Arizona: The Coriolis Group, 1997).

5. Schneier, Bruce, *Applied Cryptography*, 2nd Ed., Wiley, New York, 1996.

6. Bob Wescott, "Availability", Stratus Technical Report, Stratus Computer, Inc., 1998.

**Figure 2: Maximum Throughput undeı Heavy Load**

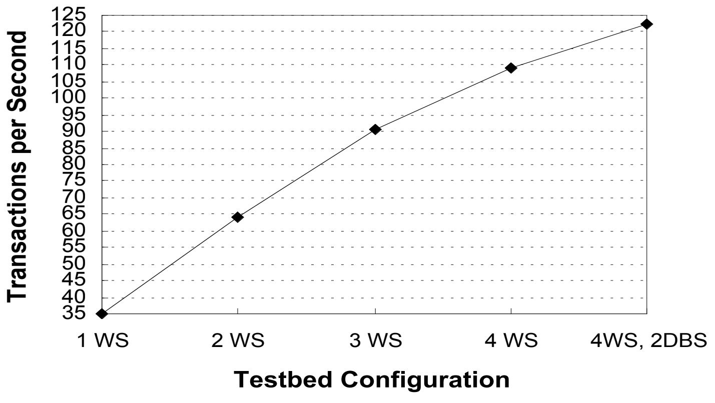


**Figure 3: Simple Transaction Mix**

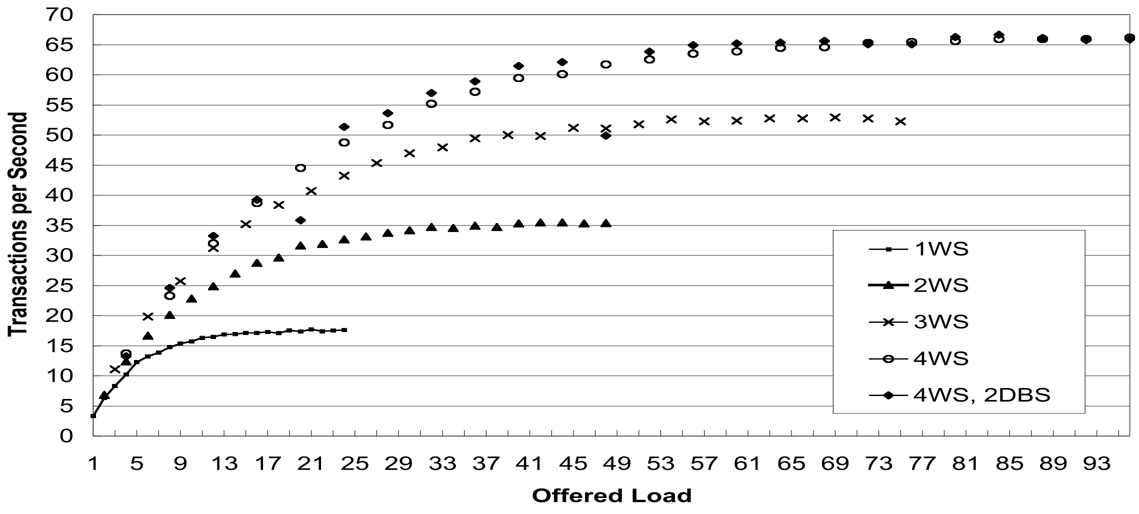
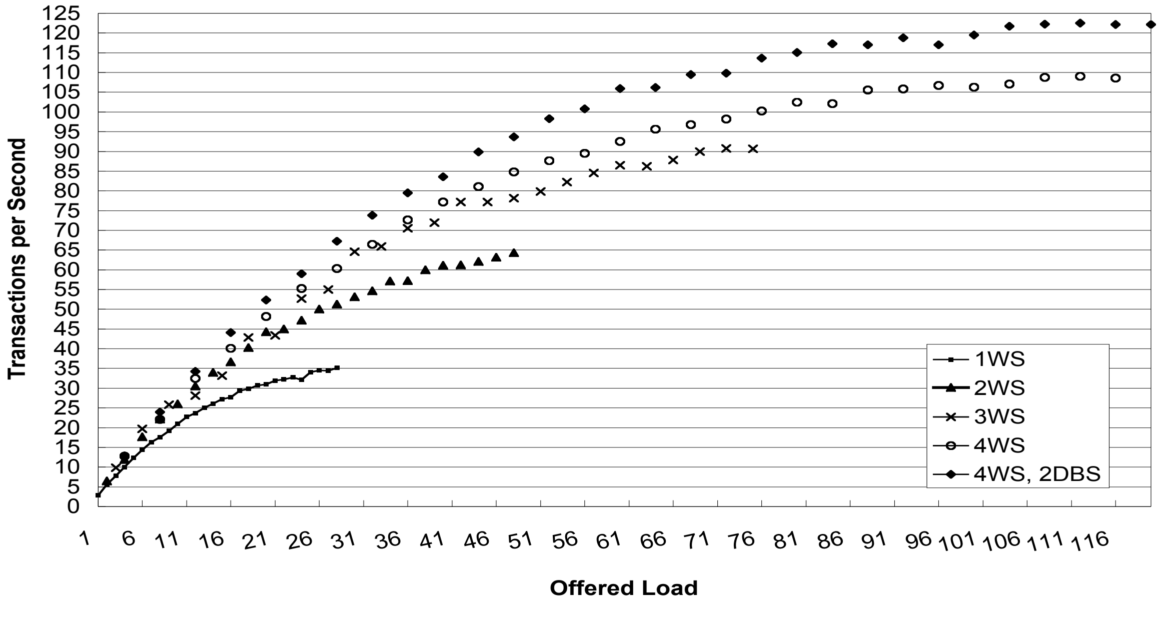


**Figure 4: Complex Transaction Mix**


Transactions per Second
1WS
2WS
3WS
4WS
4WS, 2DBS
Offered Load