Teaching Scalability Issues in Large Scale Database Application Development

Russell Anderson
randerson@mail.wtamu.edu

Musa Jafar
mjafar@mail.wtamu.edu

Amjad Abdullat
aabdullat@mail.wtamu.edu

CIS Department, West Texas A&M University
Canyon, TX 79018

Abstract

Many information systems degree programs include a course in database application development. The course typically requires students to design and build a complete database application. Instructors usually discuss database performance and scalability. However, giving students in-depth, hands-on performance and scalability experience is difficult. Problems often originate due to small size test databases and from testing a whole application with too few concurrent users. This can be partially overcome by requiring students to programmatically populate a large test database; it still does not address performance and scalability problems that arise when hundreds or even thousands of users concurrently execute transactions against the application. At the Computer Information Systems Department of West Texas A&M University, we implemented a hardware/software solution platform that allows students to assess most performance and scalability characteristics of a database application. The platform permits students to execute thousands of concurrent transactions against the application. Thus, students can monitor and gather performance statistics such as minimum, maximum, and mean transaction response time, and failure rates due to locking or configuration problems. This paper describes in detail the methodology and the solution platform we used, presents the results of our first round of use in a classroom setting, the students' learning experiences and issues encountered.

Keywords: database performance, scalability, web applications, testing

1. INTRODUCTION

At the CIS Department of West Texas A&M University, we have two courses in the database management series. The first introduces the students to database management. It covers relational database management vocabulary and theory, data modeling including E-R diagramming, data normalization, and a heavy dose of SQL. In the second course, the emphasis is on application development in a web-based enterprise environment where the browser is the front end, an application server in the middle and the database management system is employed as the primary mechanism for data storage and retrieval as the back end. (Chen, 2004) addresses the challenges in teaching a database course in general and (YAP, 2004) layout the challenges involved in deploying scalable Database-driven Web

Architecture and the various architectural components of such an application, although the authors did lay down the architectural components, they did not provide mechanisms for testing the performance and scalability of the applications. In this course we discuss application scalability and performance tuning. However until recently, we have not been able to give students a realistic, hands-on experience; allowing them to accurately assess the performance and scalability characteristics of their applications. This is was due to the effort involved in setting up a "neutral" test environment that is sustainable across semesters and the amount of scripting and configuration effort involved that is usually beyond the scope of the course (TPC BENCHMARK™ C).

Although many academicians did address the challenges and present strategies for teaching such a course sequence, we have not seen any where in the literature where the hands-on performance and scalability issue have been directly addresses (Yap, 20004; Abuhejleh, 20002; Lenox, 2004; Wagner, 2003; Chen, 2004).

This paper presents a methodology that we have developed at the CIS Department of WTAMU. It allows students to conduct, and then evaluate results of realistic, high-load application performance tests. Our specific objectives are:

- allow students to gain insight into how their database applications will perform under realistic load conditions;
- help students to better understand the factors affecting performance, scalability and DBMS configuration in developing database applications; and
- develop an environment that is sustainable and reusable across semesters with virtually zero scripting effort and minimal configuration and setup effort.

In the paper that follows, we first review the issues encountered in teaching performance and scalability assessment in a classroom setting. This is followed by a description of the methodology we have developed to enable such testing. The paper concludes with a presentation of the results achieved when the methodology was employed in our database applications course.

2. ASSESSMENT ISSUES

In the past, attempts by students to assess application performance and scalability issues were hindered by the following problems: First, the size of the database is small (small row count or small number of integrity constraints). It was typical to have students design a data model which in terms of the schema complexity was similar to that encountered in a real-world corporate environment. However, typical tables in corporate databases range in size from tens of thousands of rows to millions of rows; whereas students may manually enter up to one hundred rows total then begin testing. (Wagner 2003) provides a good resource and pointers to scientific databases that are large enough and are suitable for using in a database system course.

Second, in the corporate environment, the number of concurrent users of an application may range from the tens up to the tens of thousands. Large user populations are especially common in web based applications where the responsibility for data entry has been extended to the customer or business partner. In a classroom test environment, the student/developer himself may be the only user, or in the best case, a student project team of five to six members may perform some concurrency testing. In this latter case students may, using their knowledge of the code, test specific potential problems with a coordinated attack on a predetermined unit of data in the database, hitting a predetermined function point in the application. Such an approach may validate already identified potential concurrency problems or bottlenecks, but obviously fails to help students identify additional potential problem locations.

Before developing our current performance assessment methodology, the process we used for assessment was mostly manual. Students first developed the logic for each required transaction in their application. Each query/update against the database found in that logic was then evaluated. This manual assessment was based on the idea that the major determinant of query response time is the number of physical disk I/O's. If we can predict the number of physical I/O's that are required to resolve a query, then we can predict response time. To accomplish this we had the students develop a sample set of queries that retrieved and modified data in a set of tables of varying sizes – with and without indexes. As those queries executed, they recorded the disk I/O's and execution times, then used these results to develop simple functions to predict disk I/O's given the query type, and the size and structure of the tables involved. These functions could then be applied to the previously identified queries, yielding response time estimates.

As a second step, students were required to evaluate the logic of each transaction – asking the question: "Can the process be recoded or the database restructured in a way that will yield better response time?" Students used the "Explain Plan" feature of the Oracle DBMS that takes as input an SQL query then returns as output, the optimized steps that the DBMS will go through to execute the query. The output of "Explain Plan" can then be used to help identify inefficiencies in the transaction logic and indirectly point to possible changes in the logic and/or the database structure that would improve performance.

The above described processes have both advantages and shortcomings. The most important advantage is that it forces the student to think about what is going on inside the DBMS. It is no longer a magical black box. For example, students quickly realize that adding an index to improve data retrieval performance will adversely affect update performance on the indexed column. It also gives the student an appreciation for what the query optimizer is doing. For example, a student may look at the output of an "Explain Plan" and decides to change the query from a "nested select" to a "join" that retrieves the same result set. However, when they compare the first and second plans, they find that the optimizer has generated the exact same execution sequence. Changing the structure of the query has done nothing to improve expected response time.

The process has four major short-comings. First, it is difficult to factor in the effects of data caching. A query that takes 10 seconds to execute the first time, may take less than a second when immediately re-executed. In the real world, it is difficult to predict what data will likely be cached and what will not. Thus, in the I/O analysis, we take the pessimistic approach and have the students clear the cache before each query execution.

The second short-coming is that the process ignores the issues of concurrent access. It does not take into account the expected number of users and the performance effects that they will have on the DBMS. It also does not consider the potential problems that data locking may cause when concurrent users attempt to access and update the same data.

A third short-coming is that it does not take into consideration operating system and DBMS configuration options. Good database administrators earn their keep by successfully tweaking configuration settings. In our courses we certainly don't expect to make competent database administrators out of our students, but we would like to give them some exposure to the settings that are available and the potential effects that these settings can have on DBMS performance.

A final short-coming (in an academic environment) it is hard to deploy hardware, software architecture, terminals, network equipment and configuration that are "identical" which is a TPC requirement (TPC Benchmark C, 2005) for a realistic performance testing and scalability environment.

3. METHODOLOGY OVERVIEW

In this section we describe the methodology that we have developed to overcome the performance assessment problems described in the previous section. Students in our second database course (Database Applications) are assigned semester long application development projects that employ the methodology. The steps of the methodology are:

1. Create a data model for the proposed system by constructing an Entity-Relationship diagram, then converting the diagram to a relational database schema.
2. Implement the data model, and then populate the database.
3. Code the application.
4. Iterate until satisfactory results are achieved.
   a. Conduct performance and scalability tests of the application.
   b. Evaluate results.

c. Modify the application logic and the database configuration.

4. IMPLEMENTATION DETAILS

The application chosen for implementation in the course was a record keeping system for a small animal veterinary clinic. Specific sub-systems included: patient (animal) and client visit tracking, sales from inventory, payment processing, inventory management, and purchasing.

To get realistic results in the performance tests a database is needed that is comparable in size and structure to typical databases implemented in a corporate environment. Given that the project is of sufficient complexity, a well constructed Entity-Relationship diagram will guarantee that the resulting database schema will be useful for performance testing. A diagram of the clinic schema used is presented in Figure 1. The database schema was generated using IBM-Rational Architect from IBM. It consisted of twelve tables – five master tables (client, animal, service, vendor, and inventory) and seven containing transaction (detail) data. Also, five of the transaction tables had multi-attribute primary keys. These represented connections in many-to-many relationships between master and transaction entities. Although in terms of table count, this would not be considered representative of typical corporate databases, in terms of navigation complexity and size of the database, it is sufficient.

To populate a database of sufficient size, students were required to write a JAVA application that filled the database with randomly generated content. Issues in coding the application included:

- Even though the data was randomly generated, it still needed to conform to domain restrictions of the columns.
- All referential integrity constraints must be maintained. For example, detail rows for a sales order must only contain rows for items that are found in the products table and sold to a customer found in the customers table.
- All other data constraints must not be violated. For example, the date of a payment check for an invoice must not predate the invoice itself.

The data must be typical and reasonable. For example, orders may contain from one to ten line items; no total order amount will exceed $10,000; and a customer will not place multiple orders on the same day or maybe even in the same week.

The JAVA applications that students wrote to populate the database were required to handle all of the above. Once coded, databases of any desired size could be generated simply by changing a few constants inside the application.

The architecture chosen for application development was web-based (Forms, JAVA servlets and Java server pages (JSP)). JSP documents consist of a blend of html, defining the browser presentation, and JAVA code, executing the application logic on the server when an HTTP request is made. It is the JAVA code executing on the server that interacts with the DBMS enabling dynamic content and implementing the transaction processing requirements of the application. The interface between the web application server and the DBMS is accomplished in JAVA via the JDBC API.

Once the application code has been completed, the next step is to conduct the performance and scalability tests. This testing process and its components are the main focus of this paper.

The system used for testing is a combination of specialized hardware and software (see Figure 2). The main components of the system are:

- One Avalanche-220EE load testing appliance from Spirent Communication (Spirent, 2003). Its purpose is to generate large quantities of realistic network traffic simulating concurrent clients from multiple subnets in the hardware. The appliance is at the center of our test methodology, it simulates thousands of web clients from multiple sub networks through hardware and configuration.
- One Dell Power Edge-800 server running the Windows 2003 server operating system with one 1GB network card. This machine executed the Oracle 10g Database Management System implementing the student project database.
- One Dell Power Edge-800 server running the Windows 2003 server operating system with one 1GB network card. It executed the Tomcat application server

where the JSP applications were implemented.

In addition, two high speed Dell switches were included in the configuration. One was used for the test load traffic, and the second was used for server administration traffic. This second switch allowed the test load network traffic to be isolated from the test administration traffic.

Four transactions within the clinic application where chosen for the performance test. See Table 1 for purpose and the number of tables accessed by each transaction.

To conduct the actual performance tests, we used the Avalanche appliance to randomly select and submit transactions to the web application server which in turn made requests to the database server. The data submitted by the processes came from previously generated transaction files which were accessed independent of the performance analysis network by the Avalanche appliance.

In setting up a test run, Avalanche allows the investigator to specify load settings either in terms of transactions per second or number of concurrent users. (When specification is by number of concurrent users, once the maximum number of concurrent users is reached, Avalanche does not start the next transaction until a transaction currently in the mix completes.) For the performance tests, we chose to specify the number of transactions per second (tps). Each student's application was tested at 5, then 25, and finally 50 transactions per second.

The duration of each test was 80 seconds: a 15 second ramp-up time to reach the peak tps rate, a 60 second span at peak rate, and then a 5 second ramp-down. Thus each student received the results from three 80 second test runs: 5, 25 and 50 tps. After each test run, the database was restored to its original state.

5. TEST RESULTS

Avalanche generates two reports in spreadsheet format for each test run. The first is a summary of performance during the test run. A list of statistics included in this report is found in Table 2. In their analysis, students were told to focus on average, minimum, and maximum page response times, and transaction success rates at each of the specified transaction per second rates.

The second spreadsheet report contains measurements of the progress of the test run sampled at four second intervals. The measurements reported are listed in Table 3. As with the summary report, students were directed to focus on average, minimum, and maximum page response times and transaction success rates. When problems occurred, such as HTTP errors or significant increases in response time, the progress report allowed students to determine when during the run the problem occurred and which transactions were having the problem.

A third output dataset that students were given to inspect was the TCP log captured by Ethereal. Hereafter referred to as the pcap log. It contained packet contents for all network traffic to and from the Avalanche appliance (Avalanche and through configurations allow for the capture of the pcap file). Its usefulness is described later in the paper.

6. STUDENT EXPERIENCE

As previously stated, the first two objectives of our database performance assessment system were to help students gain an appreciation of how their applications would perform in a large "real world" environment and to help them identify possible design and software configuration changes that could be made to improve performance. In this section, we summarize the experiences of students relative to these objectives.

Finding 1: DBMS configuration problems: For many students, the first step in analysis of the data was to look at the average response times for each of the four transactions. A plot of typical results is shown in Figure 3.

Two conclusions may be drawn from the plot. First, at 5 tps, the response time is acceptable for all transactions (well under 1 second). At 25 tps, one of the transactions (animal visit history) jumps to 20 seconds. Second, it is obvious from the plot that the results are misleading. In going from 5 tps to 25 tps, the response time increases as expected; but in going to 50 tps, the response time decreases. Students recognizing this inconsistency were told to look at

more of the data. A plot of transaction success rates explains this (see Figure 4). The response time was actually decreasing, because many of the submitted transactions were actually failing and returning almost immediately. For example, at 50 tps, only 24% of the transactions to record an animal visit were succeeding. For more information on the failures, students were told to search the pcap logs. In doing so, they found that two different messages were frequently contained within the server response (Oracle Corporation, 2003) and (Berners-Lee, 1999) document the various **ORA-** and **HTTP** errors and their diagnosis:
ORA-00018*: maximum number of sessions exceeded*,
and
HTTP 500*: Internal Server error*.

In seeing the first, students immediately recognized that the default configuration for the DBMS was not adequate. It was set at one hundred maximum sessions, which was quickly reached. The second error message told students that the web server was not able to keep up with all requests. However, since this was a course in database management, not web server administration, they were told not to attempt to solve this problem, just to recognize its existence.

Finding 2: Coding considerations and the DBMS optimizer: As students compared average response times on transactions, they found large differences between student implementations. For example, in retrieving an animal history one student's average response time was about 500 ms, while another student's was about 7000 ms.

When they compared code, they found the reason for that difference. To retrieve an animal's history required reading from five different tables. The first student had retrieved the desired data with two different queries, each specifying multi-table joins. The second had programmatically retrieved the same data by reading one row from the Visit table then searching the ServicesRendered table for the current visit number, and finally reading service descriptions from the Services table. There were two lessons learned here. First, reduce as much as possible requests from the application server (JSP page) to the database server. Second, specify for the DBMS everything you want (via multi-table queries), then allow the optimizer to choose the best way to retrieve that data.

Finding 3: Deadlock Does Happen: In the search of the pcap log, students also encountered an occasional Oracle message see (Oracle Corporation, 2003) for more explanations of the error:
ORA-00060*: deadlock detected while waiting for* resource.
In class lectures, we had discussed the possibility of deadlock and the need to include in the code the ability to detect and properly respond to a deadlock situation. Most students considered the possibility of deadlock so remote that they did not take the time to properly handle it in their code. Seeing the above error in the log file brought them back to reality. Deadlocks do occur. It must be detected and handled.

Finding 4: The Value of Indexes: After analyzing results of their test runs, students were asked to modify their applications, trying to improve performance in processes that had slow response times. A common solution was to create an index on foreign key columns. In most cases performance did not improve. To help students understand why, students were taught how to use the Oracle "Explain Plan" feature, which reports the data access sequence and access methodology that the DBMS will follow in order to execute a given query. In doing so they found that the DBMS sequentially read through the detail table first, and then used foreign key values in that table to retrieve rows in the master tables using primary key indexes. Since, primary key columns were automatically indexed; the creation of indexes on foreign key columns provided no benefit.

Finding 5: Response Time is Very Much Affected by Lock Waits: After reconfiguring the DBMS to support sufficient session and open cursor requirements, students still observed dramatic response time deterioration when going from 5 to 25 then 50 tps. To help them understand what was happening, we had them open and watch the visual Oracle performance monitor where they could see a live graph at runtime showing the number of current locks and the number of sessions currently waiting on locks to be released. Although we did not provide output logs or other data that directly linked DBMS status to web server performance, students were able to use what they had visually seen on the monitor to go back and examine the spreadsheet output which reported transaction performance at four second intervals throughout the test run. In doing so they found a correlation between times of high transaction response time and the visually observed high lock wait counts.

7. ACKNOWLEDGEMENTS

All the tests were performed at the Software and Network Security Testing Lab (SoNSTLab) of the CIS department at WTAMU. Spirent Communication donated the appliance testing equipment (Avalanche and Reflector). Oracle 10g and IBM Rational are part of the Oracle and IBM Academic Initiative with the CIS Department at West Texas A&M University.

8. SUMMARY

By employing a network performance test appliance in conjunction with a web based java Servlet and JSP application, we were able to successfully provide students with a "real world" test environment of their database applications. They were able to evaluate their application performance under realistic user loads far better than would be possible by sitting down a group of students in a lab and have them concurrently execute transactions. The testing environment allowed students to:

- assess which transactions had potential response time problems,
- quickly locate problems in the DBMS configuration parameters,
- discover alternative approaches to application code that improved response time,
- recognize the need in their applications for better error detection and handling,
- see the affects of a locking scheme on response time, and
- more fully appreciate the fact that problems such as deadlock do occur and also need to be addressed in the code.

This was our first attempt using the network performance test appliance in a classroom setting. Changes we plan to make in the performance testing phase of future offerings of our database applications course include:

- generation of DBMS status and performance logs that will allow the students to directly link the performance measures of the DBMS with response time measures of the web application server rather than the visual link previously employed;
- having students experiment with different locking schemes to evaluate their affects on response time and deadlock frequency; and
- scheduling time in the course for more redesign/code iterations. In the original trial students only had time for two iterations.

9. REFERENCES

Abuhejleh, Ahmad "A Second Course in database Management Systems: A Rationale and a Proposed Course Outline" Proceedings of ISECON'2002.

Berners-Lee, T. et al (1999) "Hypertext Transfer Protocol HTTP/1.1" RFC2616, http://www.w3.org/.

Chen, Catherine and Charles Ray (2004) "The Systematic Approach in Teaching Database Applications: Is there Transfer When Solving Realistic Business Problems?" Information Technology, Learning and Performance Journal, Vol. 22 No. 1, Spring 2004.

Lenox, L. Terri and Charles R. Woratschek (2004) "The Pros and Cons of Using a Comprehensive Final Project in a Database Management Systems Course: Marvin's Magnificent Magazine Publishing House", Proceeding of ISECON'04.

Oracle Corporation (2003) "Oracle® Database Error Messages, 10g Release 1 (10.1), Part Number B10744-01".

Spirent Communication (2003) "Avalanche Analyzer User Guide" http://www.spirent.com/.

Spirent Communication (2003) "Avalanche-220EE User and Administrator guides" http://www.spirent.com.

TPC BENCHMARK™ C (2005) Standard Specification Revision 5.6" http://www.tpc.org.

Wagner, J. Paul, Elizabeth Shoop and John V. Carlis (2003) "Using Scientific Data to Teach a Database Systems Course" ACM SIGSE'2003 February 19-23, pp. 224-228.

Yap. Y. Alexander and Claudia, Loebbecke (2004) "A System for Teaching MIS and MBA Students to Deploy a Scalable Database-driven Web Architecture for B2C E-Commerce." Proceedings of ISECON'2004.

10. APPENDIX: FIGURES and TABLES

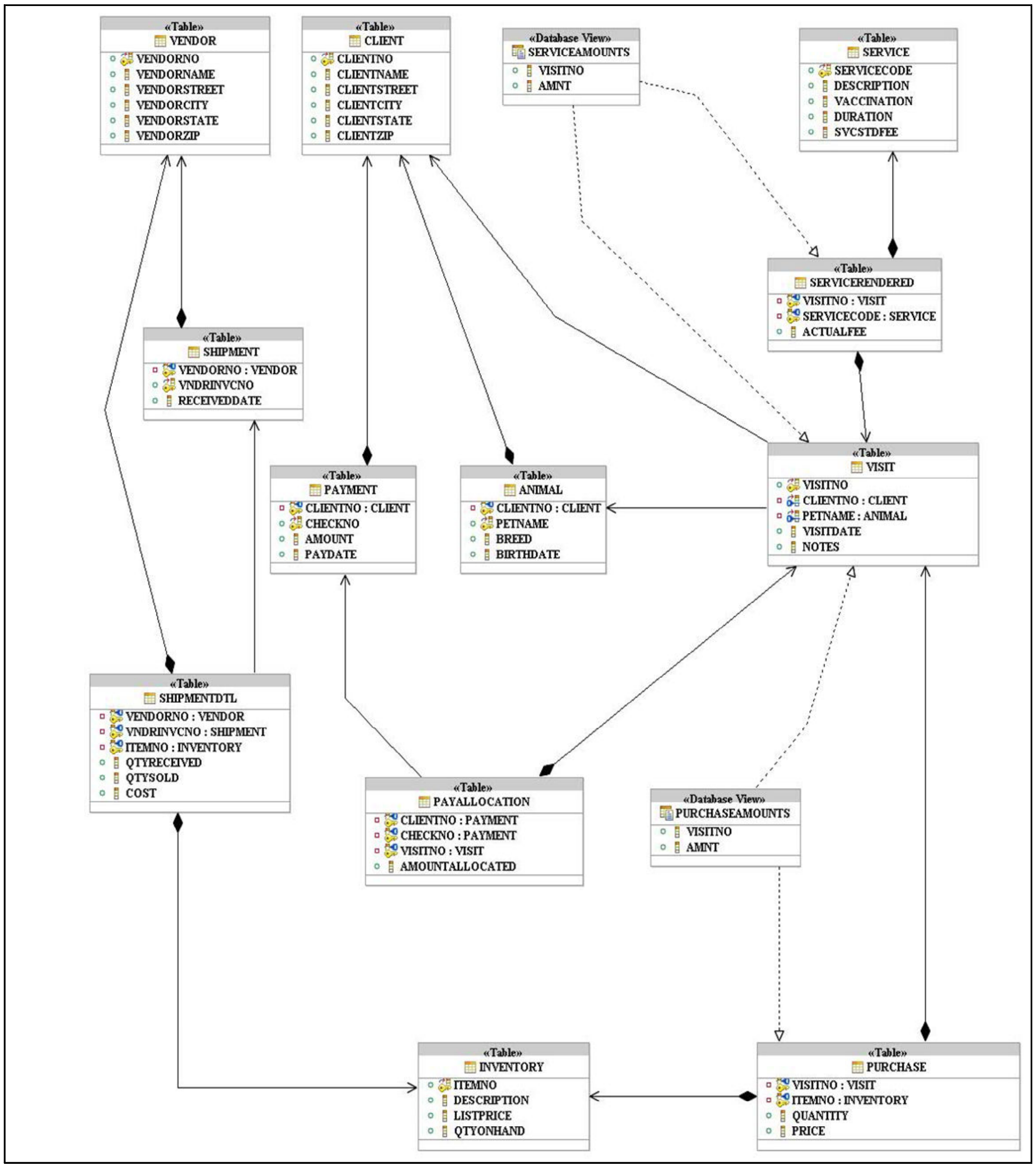

Figure 1. Vet Database Schema

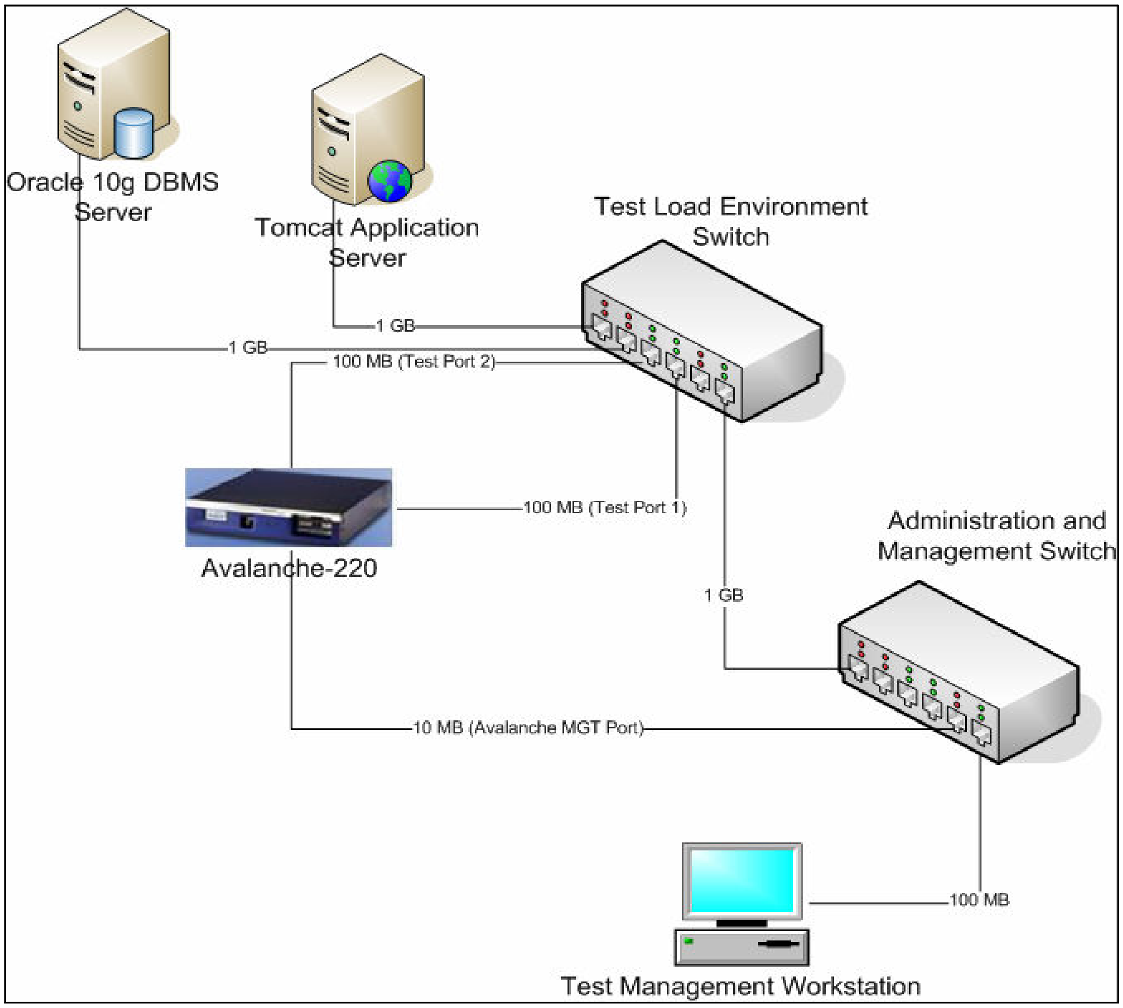
Oracle 10g DBMS Server
Tomcat Application Server
Test Load Environment Switch
1 GB
1 GB
100 MB (Test Port 2)
Avalanche-220
100 MB (Test Port 1)
Administration and Management Switch
1 GB
10 MB (Avalanche MGT Port)
100 MB
Test Management Workstation

Figure 2. Hardware Configuration of Test Environment

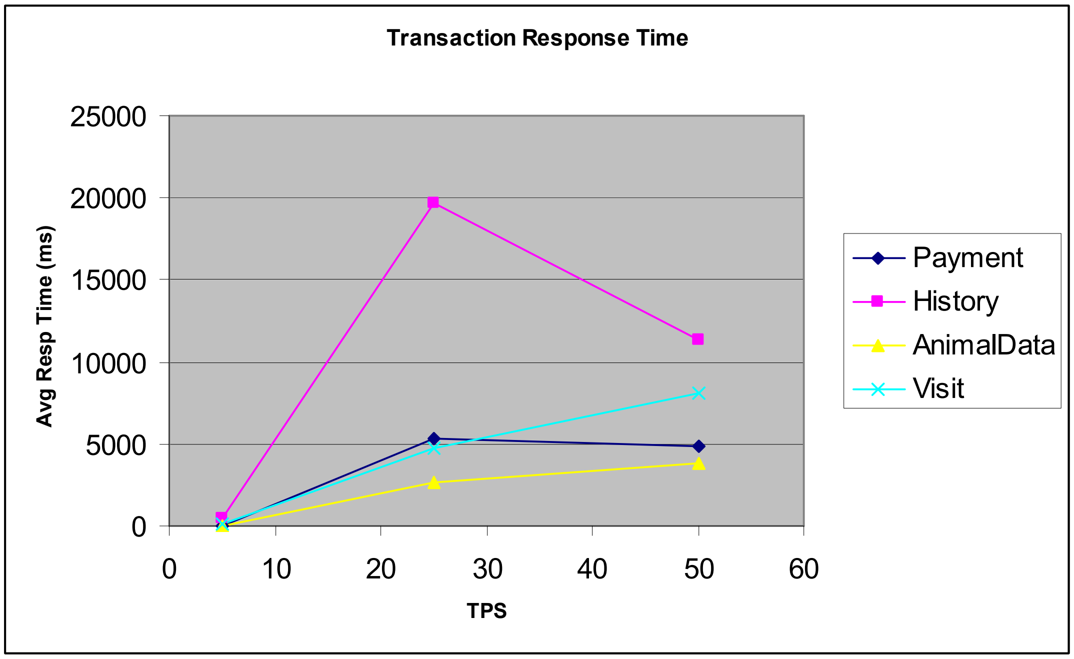

Figure 3. Transaction Response Time (ms)

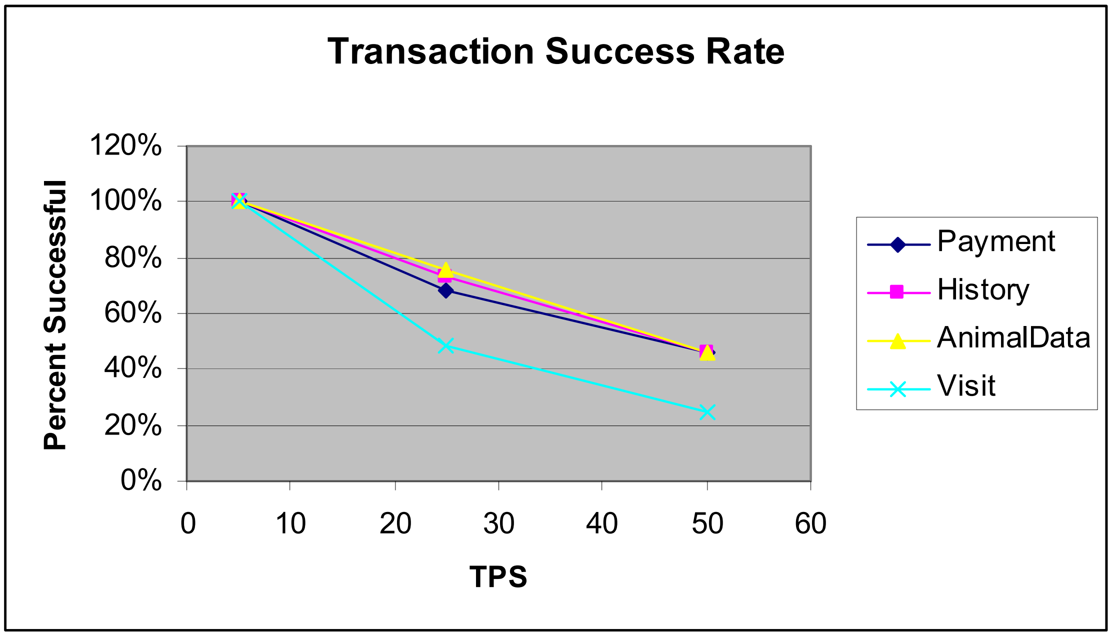

Figure 4. Transaction Success Rate (ms)

Purpose	# Tables Read	# Tables Modified
Retrieve client/animal data	2	0
Retrieve animal history	5	0
Record Visit	4	6
Record Payment	2	2

Table 1. Test Transaction

Total Attempted Transactions	Total Successful Transactions
Total Unsuccessful Transactions	Total Aborted Transactions
Attempted Transactions/Sec	Successful Transactions/Sec
Unsuccessful Transactions/Sec	Aborted Transactions/Sec
Total Attempted TCP Connections	Total Established TCP Connections
Min Time To TCP SYN/ACK	Max Time To TCP SYN/ACK
Avg. Time To TCP SYN/ACK	Min Round Trip Time
Max Round Trip Time	Avg. Round Trip Time
Avg. Retransmit Timeout	Min Time To First Data Byte
Max Time To First Data Byte	Avg. Time To First Data Byte
Min Est. Server Response Time	Max Est. Server Response Time
Avg. Est. Server Response Time	Min URL Response Time
Max URL Response Time	Avg. URL Response Time
Min Page Response Time	Max Page Response Time
Avg. Page Response Time	

Table 2. Available Test Summary Results (time in ms)

Total Attempted Transactions	Total Successful Transactions
Total Unsuccessful Transactions	Total Aborted Transactions
Attempted Transactions/Sec	Successful Transactions/Sec
Unsuccessful Transactions/Sec	Aborted Transactions/Sec
Total Attempted TCP Connections	Total Established TCP Connections
Min Time To TCP SYN/ACK	Max Time To TCP SYN/ACK
Avg. Time To TCP SYN/ACK	Min Round Trip Time
Max Round Trip Time	Avg. Round Trip Time
Avg. Retransmit Timeout	Min Time To First Data Byte
Max Time To First Data Byte	Avg. Time To First Data Byte
Min Est. Server Response Time	Max Est. Server Response Time
Avg. Est. Server Response Time	Min URL Response Time
Max URL Response Time	Avg. URL Response Time
Min Page Response Time	Max Page Response Time
Avg. Page Response Time	

Table 3. Available Test Summary Results (time in ms)

Seconds Elapsed	Desired Load (Transactions/sec)
Current Load (Transactions/sec)	Cumulative Attempted Transactions
Cumulative Successful Transactions	Cumulative Unsuccessful Transactions
Cumulative Aborted Transactions	Attempted Transactions/Second
Successful Transactions/Second	Unsuccessful Transactions/Second
Aborted Transactions/Second	Incoming Traffic (Kbps)
Outgoing Traffic (Kbps)	Incoming Packets (Packets/sec)
Outgoing Packets (Packets/sec)	Current Attempted TCP Connections
Attempted TCP Connection Rate	Current Established TCP Connections
Established TCP Connection Rate	Min Time to TCP SYN/ACK
Max Time to TCP SYN/ACK	Current Time to TCP SYN/ACK
Min Round Trip Time	Max Round Trip Time
Current Round Trip Time	Current Retransmit Time Out
Min Est. Server Process Time	Max Est Server Process Time
Current Est. Server Process Time	Min Time to TCP First Byte
Max Time to TCP First Byte	Current Time to TCP First Byte
Min Response Time Per URL	Max Response Time Per URL
Current Response Time Per URL	Min Response Time Per Page
Max Response Time Per Page	Current Response Time Per Page

Table 4. Available Test Run-Time Results (time in ms)