

Performance Analysis of Three Database Server Distribution Algorithms

By
Christopher G. Brown

Chapter 1

Introduction

Perhaps even more so than computer technology as a whole, technology related to the internet has evolved at an exponential rate in recent years. Within the internet, http has become the service protocol of choice, which means applications tend to be “web” based. This growth makes a wide variety of applications available to a huge pool of potential users.

Before applications became web applications the maximum number of potential users was limited to the size of the private network to which they were connected, generally a maximum value in the thousands of users. Whereas, with web based applications it is not unreasonable to expect a value in the millions of users.

To support an increase of this performance intensity it is important to optimize the manner in which stored data can be extracted, processed and forwarded to a web client. Hard drive technology, although improved in recent years, is still based on mechanical technology and therefore is often the slowest part of the computer system in regard to information retrieval. Even with state of the art disk drive technology adequate performance can not be obtained with some of the most intensive web applications involving the potential “millions of hits scenario.”

Therefore, given this bottleneck it seems reasonable to investigate storing the data on multiple disks, instead of on just one so hopefully the inquiries can be distributed across multiple devices instead of all being serviced by the same one. This methodology, if properly configured, offers potential to reduce the data access time. However, to what extent? Elnikety et al, 2004 [2] was able to improve throughput ten percent and decrease workstation response time by a factor of 14. It appears that there are a number of variables that influence the potential gain.

The first variable is workload intensity. It is expected as intensity increases the need to utilize some form of distributed database increases. For example, Kanitkar et al, 2002 [5] determined that distributed databases can offer significant performance advantages if the system is large enough in terms of users. They found that it takes about 40 users to reach this threshold.

The second factor is number of nodes upon which the database is distributed. One would expect that as the number of nodes increases, access time would be reduced. However, Guster et al, 2003 [3], states that at some point a point of diminishing returns will occur. This means the communication overhead among many nodes will negate the performance effect of adding additional nodes.

The third variable is the algorithm used to distribute the inquiries across multiple nodes. A symmetric algorithm, one that provides an equal chance of any given inquiry landing on any specific node, would be expected to offer the most promise.

Although the concept of the distributed database has been around for over 20 years, it has not dominated the computer landscape especially in business-related applications. The added complexity and cost of adding additional database nodes has greatly inhibited its development and use (Johnson, 2003 [4]). In fact many proponents of distributed processing, like Anthes, 2003 [1], admit that the deployed systems have barely moved beyond scientific, engineering and mathematical/statistical applications.

Although distributed database solutions are not widely deployed there is a real need for them. Applications such as the “millions of hits scenario” cannot be ignored and solutions need to be obtained if internet services are to continue to grow. Therefore research is needed that will delineate the performance advantages of distributed data base and suggest basic models of configuration. Smith et al, 2003 [7] agree and specifically state that there is a need for more performance evaluation research over more and larger databases.

Research Questions

This paper will explore the effectiveness of distributed database under a variety of conditions by conducting experiments using a number of different combinations of the variables listed above. Specifically, the following questions will be researched.

1. How does the workload intensity influence the need and performance of distributed database applications?
2. How does the number of nodes the database is stored upon affect the data access time?
3. How does the method used to assign a given query to a specific database node influence the access time?

In the course of answering these questions a simple and easily replicated method of measuring these factors will be described. It is hoped this method will have transferability to distributed data applications beyond the one used in the experiments contained in this paper.

Scope of the Study

In the interest of keeping the study feasible and focused, several parameters are defined.

Server Operating System Selection. The server operating system selected for this project is Linux. It was selected because of its openness and high degree of flexibility. It also offers high performance due to its overhead and optimized code. It was also felt that its Operating System (OS) script language would facilitate collection of performance data.

Database Software Selected. The database software selected was MySQL. This software is well tuned to the Linux operating system and uses the standard SQL language. Because MySQL is open source, it continues to grow in popularity.

Database Structure. The structure of the database will be limited to a single table. Although more complex structures would have a great influence on the performance potential, the goal of this study is to gain base line by varying the number of nodes, the workload, and the distribution algorithm. Although out of the scope of this study incorporating this variable might be well suited to subsequent research.

Workload Generator. Siege has been selected as the workload generator because it was designed to let web developers measure the performance of their code under duress, to see how it will stand up to load on the internet. It also allows for load variation by letting the user hit a webserver with a configurable number of concurrent simulated users. Those users place the webserver "under siege." The duration of the siege is measured in transactions, the sum of simulated users and the number of times each simulated user repeats the process of hitting the server.

Distribution Algorithms. Although there are multitude of possibilities, because this study is preliminary in nature it will focus on three of the most basic: sequential, random and load checking. The sequential method assigns requests in sequence among the allocated nodes without regard to their current load. The random method assigns requests randomly among the allocated nodes without regard to their current load. Whereas, the load checking methods checks the node in question to make sure its utilization is less than a certain load threshold.

Size of Cluster and Scaling Pattern. Although it would be interesting to test performance on some fairly large clusters, it is important to be practical in scope. Therefore, the maximum number of database nodes to be utilized will be limited to four. In terms of scaling, it is common to use the following pattern from which to access performance: 1, 2, 4, 8, and 16 processors. This "doubling" pattern has been widely used in other studies and from a consistency and transferability perspective, will be adopted in this study.

These limitations made the study manageable in scope and will hopefully make it easier for the reader to evaluate and use the results. Before presenting the details of the methodology and the experiments used to evaluate the research questions, a review of the literature in distributed databases is appropriate and will appear in the next chapter.

Chapter 2

Review of Literature

The review of literature will be broken into four parts. First, the advantages of using a distributed data base will be described. Second, design specification will be discussed. Third, performance issues will be delineated. Last, the degree to which distributed databases have been embraced by vendors will be discussed.

Advantages

Peddemors et al, 1998 [9] state there are numerous advantages to using the distributed database architecture, especially when the load becomes intense. They further state it is especially well suited for HTTP applications across the internet. Sobol et al, 1996 [10] state that the increase in client-server and other telecommunication based applications will spur dispersed and distributed processing, and hence the need for efficient access to organizational databases will increase. These increasing demands on databases make efficient storage space and access time important issues. Therefore, new and innovative database architectures including distributed databases will be required. Building distributed databases using the client/server architecture has been successful for quite some time. For example, Roussopoulos et al, 1993 [6] developed an advanced data management system at the University of Maryland in 1993. However, it appears that the explosion of internet applications and the resulting “millions of hits scenario” has brought the need for employing distributed data bases to the foreground.

Design Considerations

Amiri, 2003 [11] states that there are numerous inherent advantages for a multimedia retailer to select a distributed database architecture. However, the design of the system must be well thought out. The problem consists of planning the design/expansion of the distributed database system by introducing new database servers and possibly retiring some existing ones. The goal will be to reduce telecommunication costs for processing user queries and server acquisition, operations and maintenance in a multi-period environment where user processing demand varies over time.

Li et al, 2004 [12] also emphasized the importance of good design. They state, with the availability of content delivery networks (CDN), many database-driven web applications rely on data centers that host applications and database contents for better performance and higher reliability. However, it raises additional issues associated with database/data center synchronization, query/transaction routing, load balancing, and application result correctness/precision. Therefore, they feel that these design issues must be addressed if critical web applications in a distributed data center infrastructure are to be successful.

Welsh, 2002 [13] agrees that good design is important and further states that existing programming/data models and operating system structures do not adequately meet the

needs of complex, dynamic internet servers, which must support extreme concurrency (on the order of tens of thousands of client connections) and experience load spikes that are orders of magnitude greater than the average. Therefore, the manner in which the load is balanced among distributed database nodes becomes crucial to obtaining adequate retrieval performance.

Simha et al, 1997 [14] have described two of the major concerns of distributed database design. One is the problem of characterizing the number of distinct sites accessed by transactions in a distributed database, and the other is the problem of determining the number of block accesses in a relation. The first problem is directly related to this study because it deals with the number of nodes and the access pattern. The second problem deals with how the data will be subdivided within a given node.

All the literature reviewed reveals concerns about maintaining reliability given the added complexity of distributed databases. Xiong et al, 2001 [15] addressed that concern. *Data replication* can help database systems meet the stringent temporal constraints of current real-time applications, especially web-based directory and electronic commerce services. A prerequisite for realizing the benefits of replication, however, is the development of high-performance *concurrency control* mechanisms. Simply stated, this means all nodes containing the data must be synchronized and up to date.

Wu et al, 1996 [16] agree that reliability is important and devised a protocol to address the problem. Their paper presented a novel scheme for implementing a flexible replica control protocol in distributed database systems. The scheme required fewer nodes to be locked to perform the read/write operations. This not only provided better performance, but also gave the system designer extra flexibility to implement the protocol.

In terms of practicality for smaller organizations, there has been some concern about implementing distributed databases on cheaper less specialized hardware as opposed to high end clusters. Soleimany et al, 2002 [17] proved that a distributed database can be successfully implemented on standard PC architecture. Specifically they state, a network of workstations (NOWs) is an attractive alternative to parallel database systems due to the cost advantage. In a typical database, client workstations (nodes) submit queries/transactions and receive responses from the database server. With even recent PC-based client nodes providing traditional workstation-class performance, performance improvements can be obtained by offloading some of the processing typically done on the traditional server node to these powerful client nodes. Parallel query processing takes advantage of the idle cycles on the client nodes to process the query.

Performance Issues

Cannataro et al, 2002 [18] are proponents of distributed processing. They state that the integration of parallel and distributed computational environments will produce major improvements in performance for both computing and data intensive applications in the future. In fact their introductory article provides an overview of the main issues in

parallel data intensive computing in scientific and commercial applications and encourages the reader to go into the more in-depth articles contained later in the special issue journal in which their work was published.

Jutla et al, 1999 [19] feel that it is important for end users to be able to evaluate the performance potential of distributed databases. Their paper focuses on the design issues in developing benchmarks for e-commerce. They state that because of the multidisciplinary aspects of e-commerce and the various emerging and distinct e-commerce business models, creating a single benchmark for the e-commerce application is not feasible. Furthermore, they add, the diverse needs of small to medium enterprises (SMEs) and big business motivate the need for a benchmark suite for e-commerce.

Rajamani, 2002 [20] states that the key to providing adequate performance in today's internet applications is attacking the data request time problem. Specifically, web sites have gradually shifted from delivering just static html pages and images to customized, user-specific content and a plethora of online services. Multi-tiered database-driven web sites form the predominant infrastructure for most structured and scalable approaches to dynamic content delivery. However, even with these scalable approaches, the request-time computation and high resource demands for web sites with dynamic content generate results in significantly higher latency times and lower throughput than for sites with just static content and hence require well thought out designs.

Kanitkar, 2000 [21] states that the method for distributing the queries across the nodes has a major impact on data request time. To attack that distribution problem he also proposed a new policy for scheduling transactions that assigns higher priorities to transactions that have more of their required data available locally. Then, in order to further improve the efficiency of the distributed database, he proposed a load-sharing mechanism that coordinates the movement of data and transactions so as to process each transaction at the site that offers the highest probability of successful completion.

This concern for load balancing within database nodes is shared by Huaa et al, 1999 [22]. Specifically, they feel that although a symmetric distribution might be a good starting point for the inter-arrival distribution of requests, sampling the inter-arrival distribution of the application in question and tuning the load balance algorithm appropriately could lead to improved performance.

Fricksa et al, 1999 [23] also concur with the need for load balancing and have studied this distribution question. Specifically, they proposed an analytic approach to compute the response-time distribution of operator consoles in a distributed data environment. The technique developed is based on Markov regenerative processes (MRGPs) and described with the assistance of deterministic and stochastic Petri nets. For database applications with non-symmetric distributions this methodology offers promising results.

Vendor Acceptance

Keyes, 1998 [24] states that vendors see network distributed data bases as important to the future growth and development of web-based applications. Her analysis is based on the following question. What do you get if you combine the Internet, or an Intranet, and a relational database management system (DBMS) or even an object oriented DBMS? Almost heaven, according to several database and web software companies. That is why the leading database vendors, Netscape, and others are engaged in a frantic rush to release products, stake out territory, or just map strategy to make it happen.

Keyes delineates the vendor's long term goal. In the past year, all the major relational DBMS companies—including Informix, Oracle, Sybase, IBM, and Microsoft—spelled out how they will let their customers combine the benefits of web technology with databases. Ultimately, everyone wants to support heavy-duty transaction processing. The immediate goal is to tie databases more tightly to the web through new products that can do things like accept a query from a web browser, extract the data from a database, and format it in HTML for return to the web. The long-range goal is nothing short of robust, secure transaction processing.

In terms of the Open Systems Interconnect (OSI) model, Keyes describes the vendor's interpretation. Database vendors, already secure in the art of three-tier database processing, see the web as the ultimate in middleware—widely distributed, platform independent and easy to use.

Furthermore, the potential of distributed databases has already been embraced and implemented by vendors although aimed at high-end users. In fact, Townsend et al, 2003 [8], in a white paper for Oracle report that with distributed processing their database product now scales to support millions of transactions per minute.

With the review of literature now complete, Chapter 3 will focus on the research methodology and results. Conclusions and recommendations will appear in Chapter 4.

Chapter 3

Methodology and Results

As stated earlier there are three research questions designed to guide this study:

1. How does the workload intensity influence the need and performance of distributed database applications?
2. How does the number of nodes the database is stored upon affect the data access time?
3. How does the method used to assign a given query to a specific database node influence the access time?

These questions can be modified to provide three null hypotheses which can be tested through experimentation.

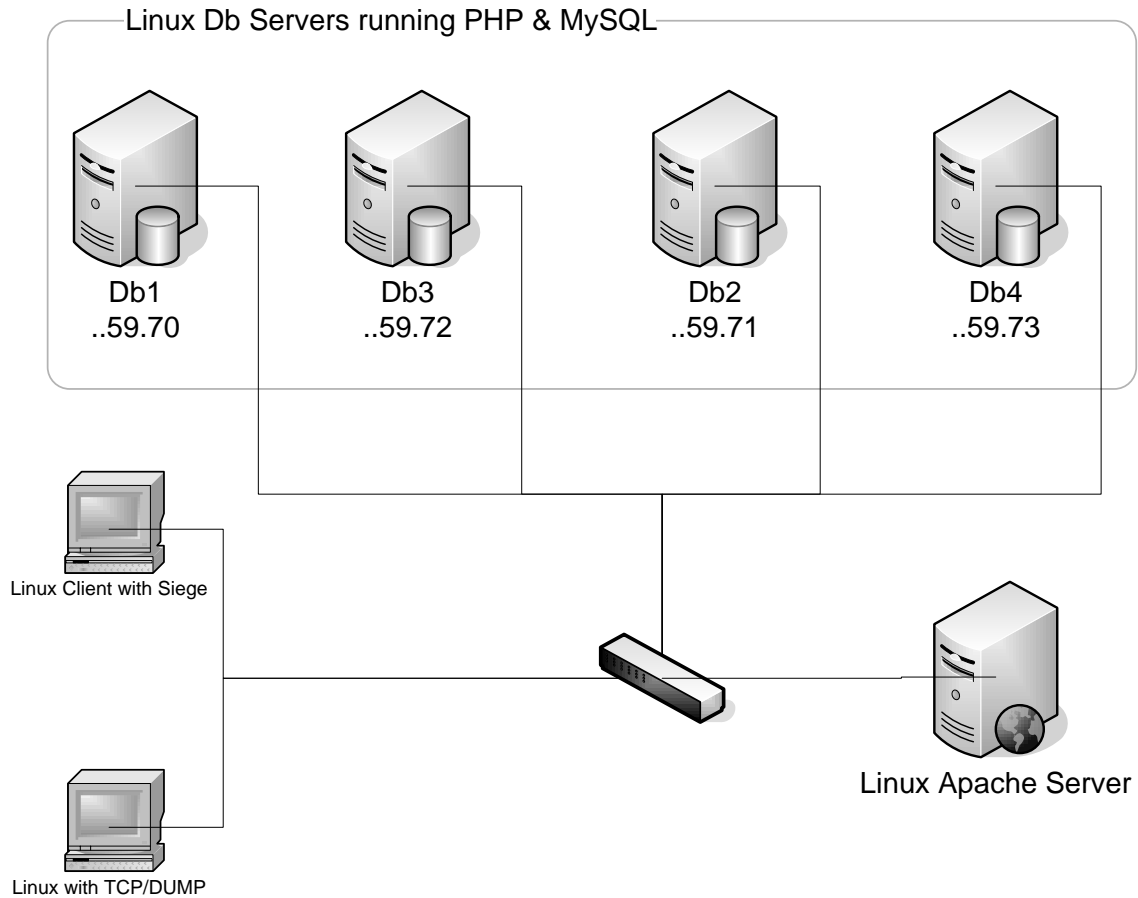
H1. Workload intensity has no affect on the retrieval time of records from a distributed database and hence on the delay back to the originating client.

H2. The number of nodes a database is stored upon has no affect on response time back to the originating client.

H3. The algorithm used to distribute requests to a given distributed database node has no affect on the delay back to the originating client.

In order to collect data to test these hypotheses a database test bed will be devised in which the workload can simulated for any number of concurrent client browser sessions. The distribution algorithm can be varied and the number of nodes on which the database is distributed can be varied from one to four. A drawing of this test bed appears below as Figure 1.

Figure 1.



The actual collection agent within this environment will be a packet sniffer process generated by TCPDUMP. This collection agent will trap data from each packet generated by the experimental tests. The URL's used to test the three methods were sequential (<http://199.17.59.65/page/?function=sequential>), random (<http://199.17.59.65/page/?function=random>) and load balanced (<http://199.17.59.65/page/?function=load>). The apache server would then redirect the output based on the predefined algorithm set up for each method. The following variables will appear in each packet record: time stamp, source Media Access Control (MAC) address, destination MAC address, size of the packet, source network.node.port address, and destination network.node.port address. This data, once processed can provide metrics in the following categories: delay to the client, data throughput, and data intensity. The workload was generated by a high-end processor running Linux. The software used Siege (see Appendix A), which is able to generate web traffic streams of varying intensity. For the experiments run herein the traffic of eight consecutive groups of 50, 100, 200, and 400 clients was generated in four separate tests. The client requests were forwarded to a Linux webserver via a 100 mbps Ethernet network. That webserver in turn made the disk Input/Output (I/O) requests to either one, two or four database servers running a MYSQL database consisting of a single indexed table having 29 fields containing 11,552 records. In the case where multiple database servers are used the same database was replicated to each database node. Therefore, the data request could be filled

by any one of the four potential databases and get the same results. Different methods were used to determine which of the data base servers (if multiple db servers were used) would receive any given request. In the sequential method, the requests followed a set sequence such as server one, then two, then three, then four, then back to one. The random method used a random number generator to select a dserver randomly from the pool of servers. It was hoped that if the number generator was truly random that the work load would get evenly distributed. The load balancing method monitored the operating system on each potential database node to ascertain its current load in real time. Dbservers under heavy loads, which were unable to report in a timely interval, were assumed to be at 100% utilization. Selection was based on the lowest utilization currently reported. For sample data and the PHP code used on the distribution Apache web server and each of the dbservers see Appendix B. The data collected is reported in a series of Tables. A separate table is provided for each of the four different client loads tested. The data sample column is used, intermittently, to separate out multiple tests using the same client variables.

Table 1
8 Consecutive Iterations of 50 Concurrent Sessions.

Data Sample	Client (A) client 1 (B) zeus	Query Distribution Type	Sequential Iterations	Server Nodes	Clients	Average Delay (ms)	Throughput (bytes/s)	Packet Intensity (packets/s)
1	A	N/A	8	1	50	2.07193316	92758.467	241.321
1	A	Sequential	8	2	50	0.74688173	191275.131	669.450
1	A	Sequential	8	4	50	0.39466387	312233.329	1266.901
1	A	Random	8	2	50	0.71621023	167432.264	698.119
1	A	Random	8	4	50	0.47683332	275472.700	1048.584
1	A	Load Balanced	8	2	50	0.17195826	252105.315	2907.682
1	A	Load Balanced	8	4	50	0.08090560	522526.965	6180.042

The data collected at the 50 client level is displayed in Table 1. At the 50 client level, each test was performed once per method and dserver node configuration. As the session load increases in Tables 1 – 4, the performance difference is amplified and suggests a higher performance return per additional dserver node.

- The first column designates which data sample the results were computed from.
- The second column indicates that the client workload stream was generated by a single Intel machine.
- The third column describes the database node allocation method. This concept is not applicable when only one dserver is used.
- The fourth column describes the number of times that the simulated 50 clients generated a request stream.

- The fifth column depicts the number of database servers used.
- The sixth column reports the number of clients generating the workload.
- The seventh column reports the average delay back to the client in filling the request. It is clear that all three distribution types improve the performance beyond the single database server. At the two database server level the sequential and random methods improve it by about 1.3 ms and the load balancing method shows even greater improvement by reducing the delay 1.9 ms. At the four dbserver level the results are even more dramatic. The sequential and random methods reduce the delay by about 1.6 ms whereas the load balancing method reduces the delay by almost two ms.
- The eighth column depicts the throughput in bytes per second. As would be expected when delay is reduced, the same amount data is delivered more quickly which results in higher per capita delivery rate. Throughput was improved from about 92,000 bytes/sec on the single database model to about 1/2 million bytes/sec on the load balanced model using four dbservers.
- The last column reports the intensity of packet traffic. As would be expected these values follow a pattern similar to the previous column in that as delay decreases packet intensity increases. In this case the packet intensity at the largest delay value was about 240 packets/sec. While at the smallest observed packet intensity delay value was about 6,000 packets/sec.

Table 2
8 Consecutive Iterations of 100 Concurrent Sessions.

Data Sample	Client (A) client 1 (B) zeus	Query Distribution Type	Sequential Iterations	Server Nodes	Clients	Average Delay (ms)	Throughput (bytes/s)	Packet Intensity (packets/s)
1	A	N/A	8	1	100	3.88490715	44360.966	128.703
1	A	Sequential	8	2	100	0.71031160	197973.048	703.916
1	A	Sequential	8	4	100	0.37551237	361269.535	1331.514
1	A	Random	8	2	100	0.63998721	202004.413	781.266
1	A	Random	8	4	100	0.44903326	312168.375	1113.503
1	A	Load Balanced	8	2	100	0.13790886	318776.122	3625.583
1	A	Load Balanced	8	4	100	0.08812921	484603.770	5673.488
2	A	Load Balanced	8	4	100	0.07656717	556347.453	6530.214

The data collected at the 100 client level is displayed in Table 2. At the 100 client level, multiple tests were conducted using the same configuration of dbserver nodes and selected query distribution method, to allow for analysis of variance for further research on this topic.

- The first column designates which data sample the results were computed from, as occasionally multiple tests were performed using the same configuration.
- The second column indicates that the client workload stream was generated by a single Intel machine.
- The third column describes the database node allocation method. This concept is not applicable when only one dbserver is used.
- The fourth column describes the number of times that the simulated 100 clients generated a request stream.
- The fifth column depicts the number of database servers used.
- The sixth column reports the number of clients generating the workload.
- The seventh column reports the average delay back to the client in filling the request. It remains clear that all three distribution types improve the performance beyond the single database server. At the two database server level the sequential and random methods improve it by about 3.2 ms and the load balancing method shows even greater improvement by reducing the delay 3.7 ms. At the four dbserver level the results are slightly more dramatic. The sequential and random methods reduce the delay by about 3.5 ms whereas the load balancing method reduces the delay by almost 3.8 ms.
- The eighth column depicts the throughput in bytes per second. As would be expected when delay is reduced the same amount of data is delivered more quickly which results in higher per capita delivery rate. Throughput improved from about 44,000 bytes/sec on the single database model to about 1/2 million bytes/sec on the load balanced model using four dbservers.
- The last column reports the intensity of packet traffic. As would be expected these values follow a pattern similar to the previous column in that as delay decreases packet intensity increases. In this case the packet intensity at the largest delay value is about 130 packets/sec. While at the smallest delay, observed packet intensity is about 6,500 packets/sec.

Table 3
8 Consecutive Iterations of 200 Concurrent Sessions.

Data Sample	Client (A) client 1 (B) zeus	Query Distribution Type	Sequential Iterations	Server Nodes	Clients	Average Delay (ms)	Throughput (bytes/s)	Packet Intensity (packets/s)
1	A	N/A	8	1	200	4.80601741	17878.602	104.036
1	A	Sequential	8	2	200	5.19456850	21524.983	96.254
1	A	Sequential	8	4	200	0.34005095	330987.386	1470.368
1	A	Random	8	2	200	13.61430900	8529.467	36.726
1	A	Random	8	4	200	0.89513973	157894.511	558.572
1	A	Load Balanced	8	2	200	0.10743538	424712.763	4653.961
1	A	Load Balanced	8	4	200	0.05969465	724683.596	8375.961

The data collected at the 200 client level is displayed in Table 3. At the 200 client level, each test was performed once per method and dbserver node configuration.

- The first column designates which data sample the results were computed from.
- The second column indicates that the client workload stream was generated by a single Intel machine.
- The third column describes the database node allocation method. This concept is not applicable when only one dbserver is used.
- The fourth column describes the number of times that the simulated 200 clients generated a request stream.
- The fifth column depicts the number of database servers used.
- The sixth column reports the number of clients generating the workload.
- The seventh column reports the average delay back to the client in filling the request. Aside from the elevated delay in the two dbserver level it remains clear that all three distribution types at the four dbserver level improve the performance beyond the single database server. At the two database server level the sequential method remains relatively consistent whereas the random method shows an increase in delay by as much as 8.8 ms and the load balancing method shows a dramatic improvement by reducing the delay 4.7 ms. At the four dbserver level the results are completely different. The sequential method shows a reduction of about 4.5 ms and the random method shows a reduction in the delay by about four ms whereas the load balancing method reduces the delay by 4.7 ms.
- The eighth column depicts the throughput in bytes per second. As would be expected when delay is reduced the same amount of data is delivered more quickly which results in higher per capita delivery rate. Throughput improved

from about 18,000 bytes/sec on the single database model to about 3/4 million bytes/sec on the load balanced model using four dbservers.

- The last column reports the intensity of packet traffic. As would be expected these values follow a pattern similar to the previous column in that as delay decreases packet intensity increases. In this case the packet intensity at the largest delay value is about 100 packets/sec. While at the smallest delay value observed packet intensity is about 8,400 packets/sec.

Table 4
8 Consecutive Iterations of 400 Concurrent Sessions.

Data Sample	Client (A) client 1 (B) zeus	Query Distribution Type	Sequential Iterations	Server Nodes	Clients	Average Delay (ms)	Throughput (bytes/s)	Packet Intensity (packets/s)
1	A	N/A	8	1	400	4.20020566	23200.616	119.042
1	B	N/A	8	1	400	21.56025828	5609.872	23.191
1	A	Sequential	8	2	400	0.62360851	216110.392	801.785
1	B	Sequential	8	2	400	11.54540789	8978.663	43.307
1	A	Sequential	8	4	400	0.31362455	385330.204	1594.263
1	B	Sequential	8	4	400	0.71562455	166563.402	698.690
1	B	Random	8	2	400	7.02728106	14320.478	71.151
1	B	Random	8	4	400	5.76863794	19649.033	86.676
1	A	Load Balanced	8	2	400	0.26592366	170338.792	1880.239
2	A	Load Balanced	8	2	400	0.24486395	211604.990	2041.950
3	A	Load Balanced	8	2	400	0.13944143	358677.363	3585.735
4	A	Load Balanced	8	2	400	0.43814796	99091.412	1141.167
1	A	Load Balanced	8	4	400	0.09072802	474901.359	5510.977
2	A	Load Balanced	8	4	400	0.12549306	346656.873	3984.284
3	A	Load Balanced	8	4	400	0.60850789	87940.314	821.682
4	A	Load Balanced	8	4	400	0.12308939	350594.854	4062.089

The data collected at the 400 client level is displayed in Table 4. At the 400 client level, multiple tests were conducted using the same configuration of nodes and selected query distribution method, to allow for analysis of variance for subsequent research on this topic. For the purposes of this paper, preliminary disciplinary analysis will be the focus. Of the four Tables, Table 4 demonstrates the highest performance gain when moving from a single dbserver to a four node distributed dbserver array.

- The first column designates which data sample the results were computed from as occasionally multiple tests were performed using the same configuration.

- The second column indicates that the client workload stream was generated by a single Intel machine and in some instances by two clients.
- The third column describes the database node allocation method. this concept is not applicable when only one dbserver is used.
- The fourth column describes the number of times that the simulated 400 clients generated a request stream.
- The fifth column depicts the number of database servers used.
- The sixth column reports the number of clients generating the workload.
- The seventh column reports the average delay back to the client in filling the request. It remains clear that all three distribution types improve the performance beyond the single database server. At the two database server level the sequential method demonstrated improvement by about 3.6 ms whereas the random method indicates an increase in delay by about 2.8 ms. The load balancing method shows the greatest improvement by reducing the delay four ms. At the four dbserver level the results are slightly more dramatic. The sequential method demonstrates a reduction of almost 3.9 ms whereas the random methods increased the delay by about 1.5 ms. The load balancing method reduces the delay by almost 4.1 ms.
- The eighth column depicts the throughput in bytes per second. As would be expected when delay is reduced the same amount of data is delivered more quickly which results in higher per capita delivery rate. Throughput improved from about 23,000 bytes/sec on the single database model to about 1/2 million bytes/sec on the load balanced model using four dbservers.
- The last column reports the intensity of packet traffic. As would be expected these values follow a pattern similar to the previous column in that as delay decreases packet intensity increases. In this case the packet intensity at the largest delay value is about 120 packets/sec. While at the smallest delay value observed packet intensity is about 5,500 packets/sec.

A comparison of values at the various client levels is best depicted graphically and Figure 2-10 will depict the values observed on average delay, throughput, and packet intensity. Average delay is depicted by Figures 2-4, with Figure 2 showing the results with the sequential method. The results from the random method are reported in Figure 3 and the results for the load balancing method are in Figure 4. Detailed plots of session times and packet payloads for the sequential, random and load balanced models by loads of 50, 100, 200 and 400 concurrent sessions can be found in Appendix D.

Figure 2

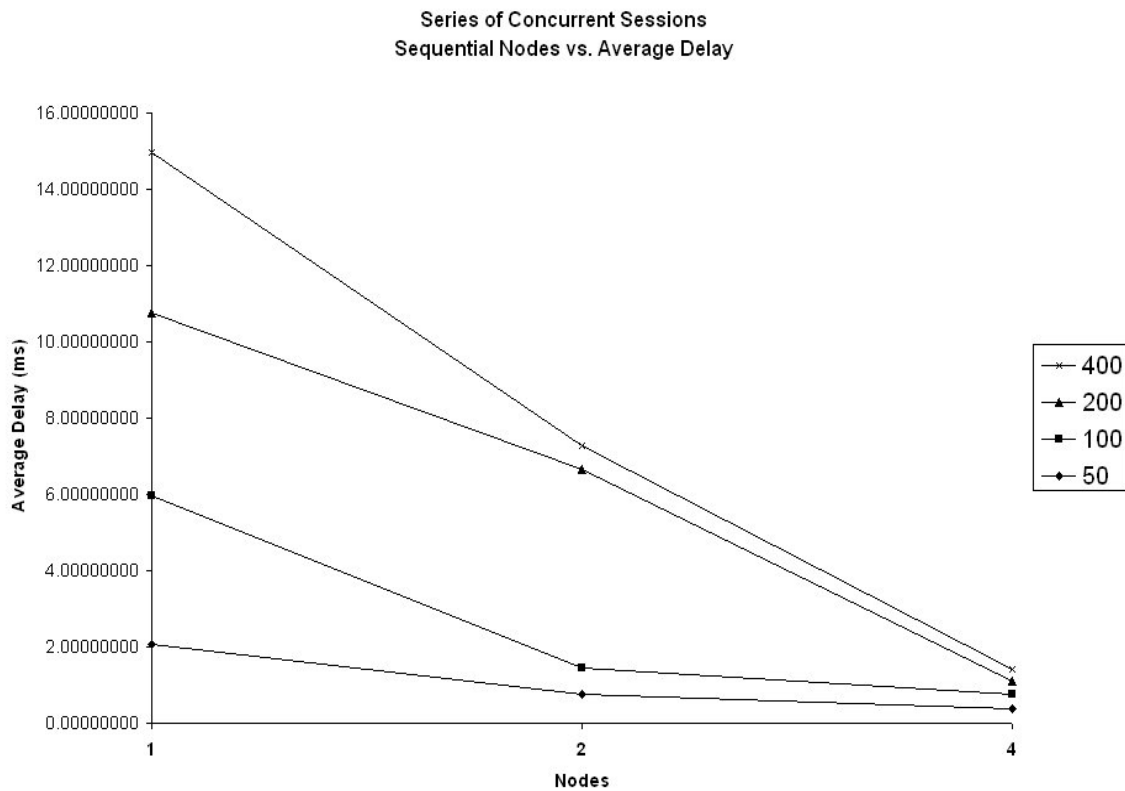


Figure 3

Series of Concurrent Sessions
Random Nodes vs. Average Delay

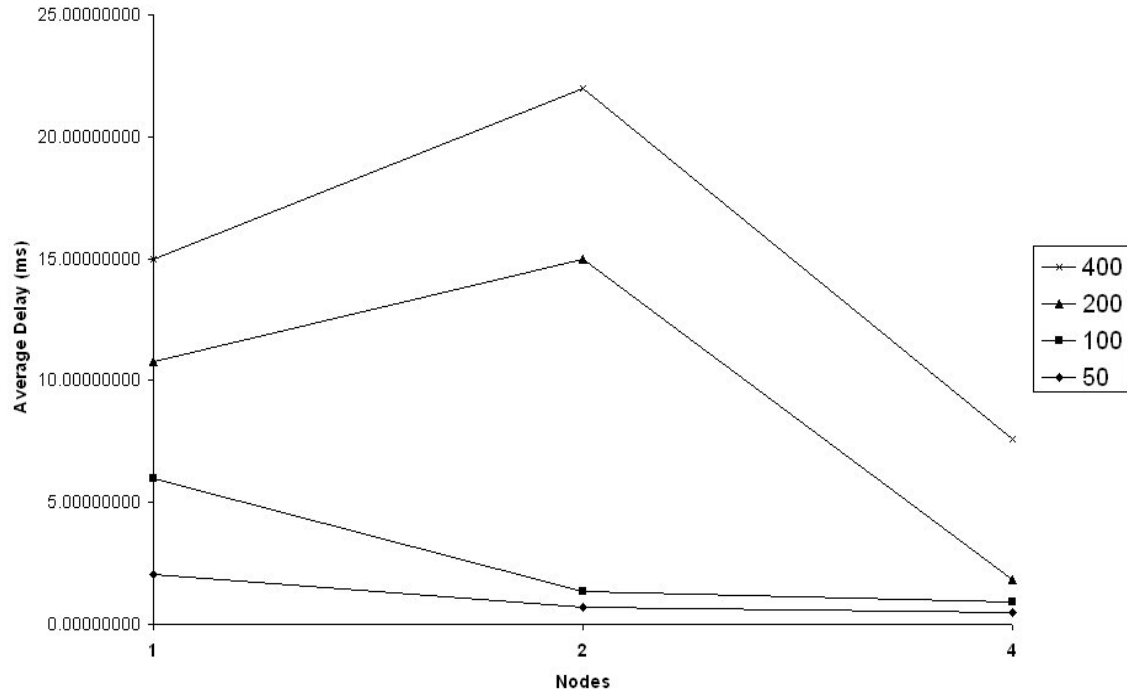
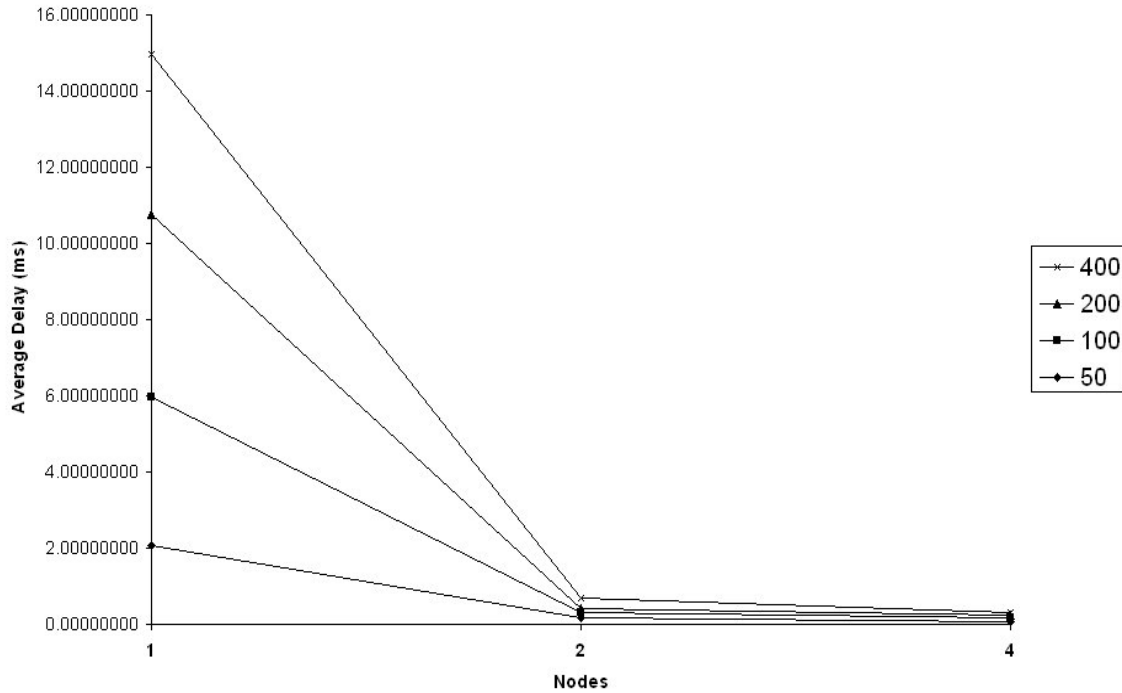


Figure 4

Series of Concurrent Sessions
Load Balanced Nodes vs. Average Delay



In all methods delay generally decreases as the number of dbrowsers is increased. However, in the case of the random method delay actually increased when moving from one to two servers, and also showed some improvement (decrease) when using four dbrowsers. It is clear that of the three methods used the load balancing was the most efficient. Although the sequential method resulted in the desired decreasing linear pattern, it was not as pronounced as with the load balancing method. The random method actually demonstrated more efficiency loss due to calculation overhead at the 2 dbrowser level and didn't obtain the efficiency that either of the other two models had at higher load levels. The load balancing method showed the most dramatic improvement at all levels when compared to the other two models.

Figure 5

Series of Concurrent Sessions
Sequential Nodes vs. Throughput

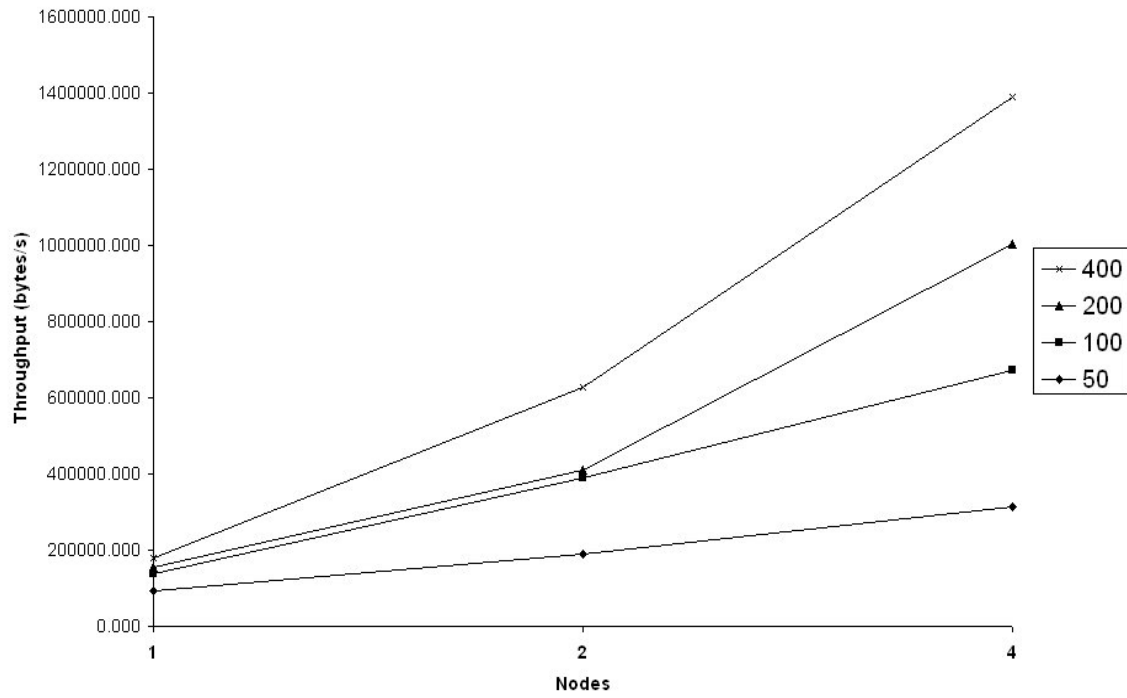


Figure 6

Series of Concurrent Sessions
Random Nodes vs. Throughput

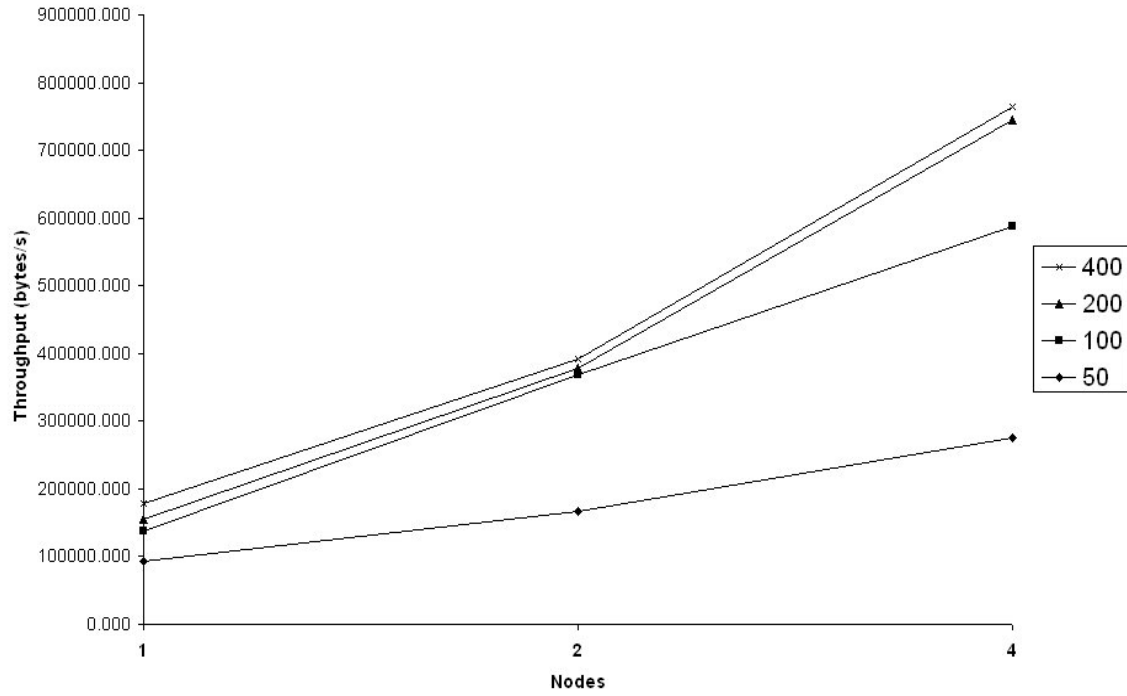
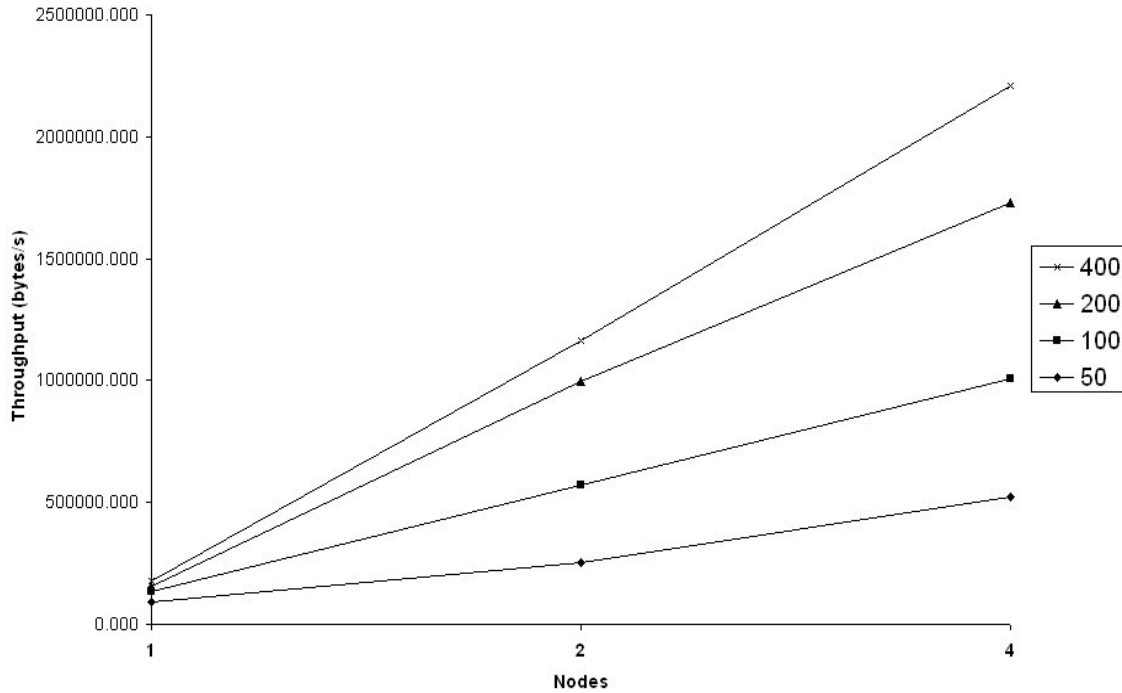


Figure 7

Series of Concurrent Sessions
Load Balanced Nodes vs. Throughput



With the decrease of delay by adding additional observers, an increase in throughput is expected. The results for throughput are not as dramatic as delay. By adding additional observers there is a somewhat linear trend with the throughput increase as we move from a sequential model to a load balanced model. As shown in Figure 7, using the random model the data with two and four observers are closely related and nearly congruent. This congruency can be largely attributed to the calculation overhead effect of the random algorithm as seen in figures 2 – 4. Further testing would be required to predict when the throughput thresholds would be reached by adding more observers and contrasting the sequential results with the load balanced results. The sequential method appears to deliver a nonlinear trend which depicts a higher return for each additional observer. However, it should be noted that the load balanced throughput at four observers approaches 1/2 million bytes per second whereas the sequential model at four nodes demonstrates a throughput of just over 1/3 million bytes per second.

Figure 8

Series of Concurrent Sessions
Sequential Nodes vs. Packet Intensity

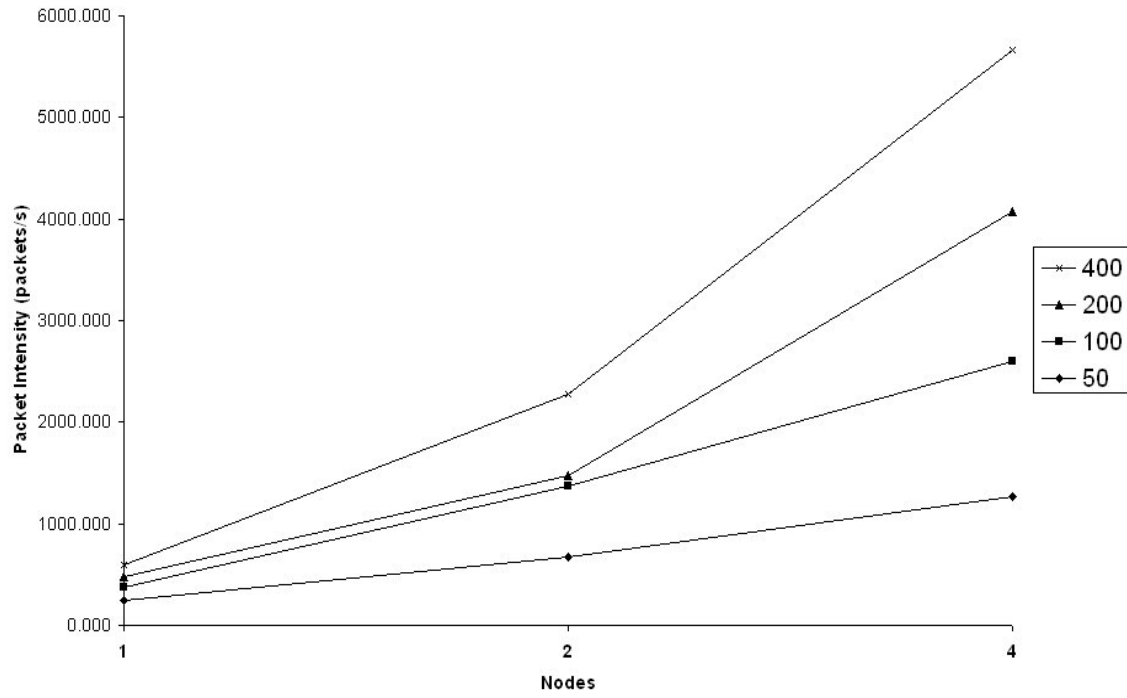


Figure 9

Series of Concurrent Sessions
Random Nodes vs. Packet Intensity

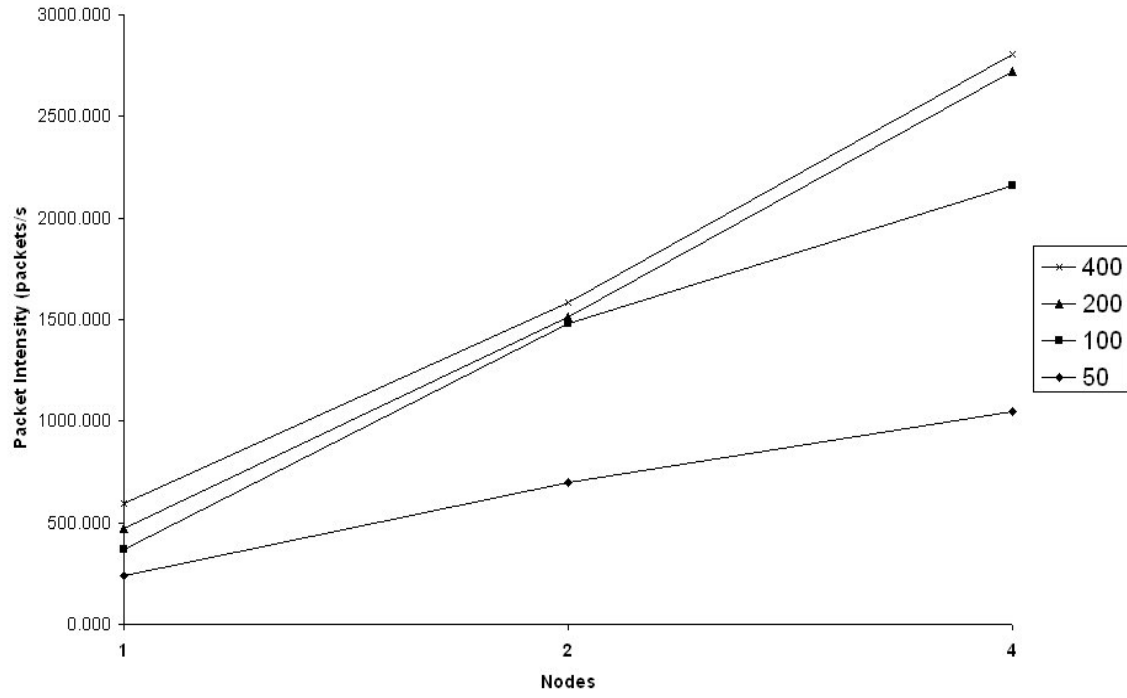
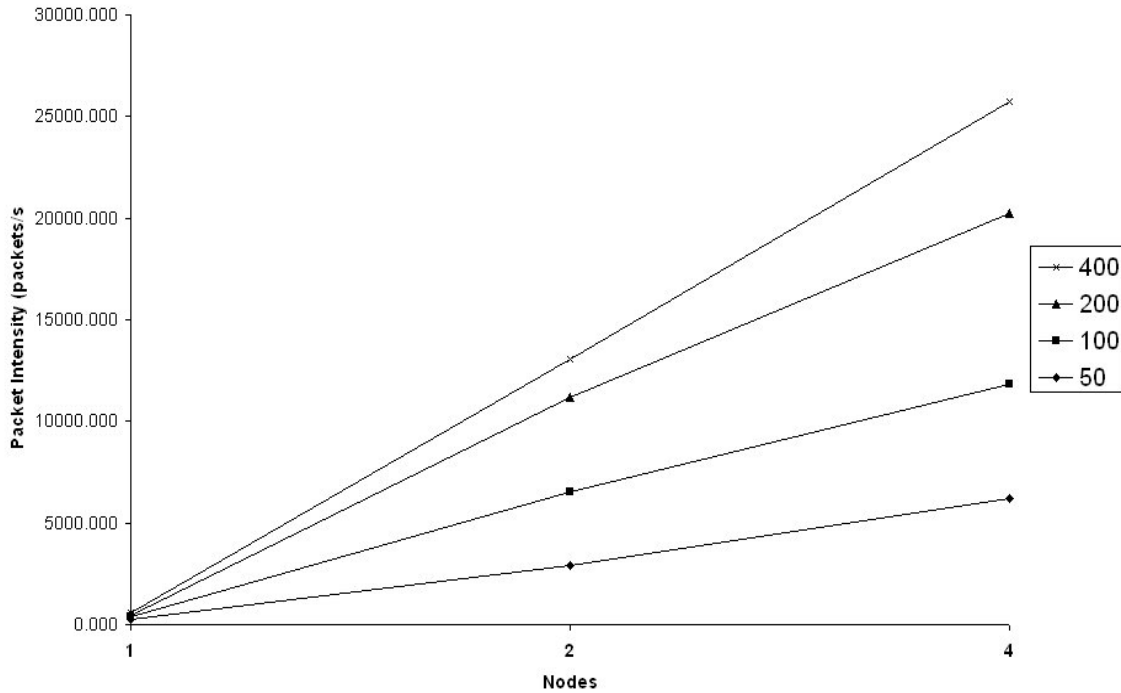


Figure 10

Series of Concurrent Sessions
Load Balanced Nodes vs. Packet Intensity



In all methods packet intensity generally increases as the number of observers is increased. However, in the case of the random method there is a clear indication that overhead is costly until a higher connection load is sustained. The load balanced model is even more efficient than the sequential model. The load balanced model peaks with four observers undergoing a load of 400 connections at just above 5,500 packets per second whereas the sequential model delivers at a bit under 1,600 packets per second. The random model results, with two and four observers, are closely related and nearly congruent. This congruency can be largely attributed to the calculation overhead effect of the random algorithm as seen in figures 2 – 4. Further testing would be required to predict when the packet intensity thresholds would be reached by adding more observers and contrasting the sequential results with the load balanced results.

Chapter 4

Conclusions

Rejection of the Three Null Hypotheses

H1. Workload intensity has no affect on the retrieval time of records from a distributed database and hence on the delay back to the originating client.

When moving from 50 concurrent sessions to 400 concurrent sessions on a single dbserver node, the delay increases by a factor of two. Adding additional dbserver nodes, distributing the workload among four nodes and increasing the concurrent sessions from 50 to 400 will increase the delay by almost 330%. Therefore hypothesis H1 must be rejected.

H2. The number of nodes a database is stored upon has no affect on response time back to the originating client.

Using the load balanced method and moving from one dbserver to four dbservers under a workload of 50 concurrent sessions there is a decrease in average delay by 96%. Setting the workload to 400 concurrent sessions, using the load balanced method and moving from one to four dbserver nodes decreases the average delay by 98%. Therefore hypothesis H2 must be rejected.

H3. The algorithm used to distribute requests to a given distributed database node has no affect on the delay back to the originating client.

Setting the workload to 50 concurrent sessions, using four dbservers, and switching from the load balanced to the sequential method, the average delay increases by almost 490% and by almost 590% when switching to the random method. When increasing the workload to 400 concurrent sessions, using four dbservers, and switching from the load balanced method to the sequential method, the average delay increases by almost 350% and by over 6,000% when switching to the random method. Therefore hypothesis H3 must be rejected.

Performance Gain as Attributed to Adding Dbserver

Average Delay

The Sequential model demonstrates a decrease in delay when moving from a single dbserver under a load of 50 concurrent sessions to a four dbserver model under the same load by 81%. This effect is amplified when the load increases to 400 concurrent sessions, reducing the delay by 98%.

It is difficult to measure the scalability with the load balanced model as it offers an immediate delay reduction of 96% even at the 50 session level when moving to four

dbservers. The effect is relatively consistent when we increase the load to 400 concurrent sessions resulting in a reduction in delay from the single dbserver model under 400 concurrent sessions by 98%.

The Random Model offers the least promising results when addressing packet delay. Compared to an 81% decrease with the sequential model and a 96 % decrease with the load balanced model, the random model offers a mere 77% decrease in average delay under a load of 50 concurrent sessions when moving from a single dbserver to four dbservers. Adding the same number of dbservers under a higher load of 400 concurrent sessions actually increases the average delay under the random model by 37%. This delay increases by 67% when going to two db servers.

Throughput

The sequential model offers a consistent increase in performance when moving from a single dbserver to four dbservers. Under a load of 50 concurrent sessions the increase to four dbservers results in a gain in throughput of almost 240%. When moving from a single dbserver under a load of 400 concurrent session to 4 dbservers, throughput is increased by 1,560%.

The load balanced model demonstrated the largest increase in throughput: 3,953% at a load of 200 concurrent sessions when moving from a single dbserver to four dbservers. A load of 400 concurrent sessions moving from one dbserver to four dbservers results in increased throughput for the load balanced model of just under 1,950%, a much lower return than the 200 session load.

The random model offers a decrease in throughput when moving from a single dbserver to four dbservers under a load of 400 concurrent sessions of 15%. Moving from one dbserver to two dbservers under the same load results in a decrease in throughput of 38%.

Packet Intensity

The sequential model peaks with an increase of packet intensity at the 200 concurrent sessions level, when moving from one dbserver to four dbservers, by 1,310%. Under a load of 200 concurrent sessions, there is a decrease in packet intensity when moving from one dbserver to two dbservers by 7% under the sequential model. When the load is increased to 400 concurrent sessions and moving from a single dbserver to four dbservers, the sequential model drops back to an increase in packet intensity of just below 1,240%. When the load is increased to 200 concurrent sessions and moving from a single dbserver to two dbservers, the sequential model demonstrates an increase in packet intensity of just below 570%.

The random model packet intensity improvement peaks with an increase of packet intensity at the 100 concurrent sessions level when moving from one dbserver to four dbservers by 760%. Under a load of 200 concurrent sessions, there is a decrease in packet intensity, when moving from one dbserver to two dbservers, of 65% under the random model. However, when moving from one dbserver to four dbservers under the same load, an increase of almost 440% is observed. The packet intensity decreases in the

random model when moving from one dbserver to both two and four dbservers by 40% and almost 30% respectively.

The load balanced model packet intensity increase peaks at the 200 concurrent session load by moving from one dbserver to four dbservers, noting an increase in packet intensity of almost 8,000% which is the highest recorded increase of any method by over six times. The load balanced method shows a depreciated increase in packet intensity by about 1/2 under a load of 400 concurrent sessions, when moving from one dbserver to two and four dbservers of 2,900% and 4,500%.

Clearly there is an increase in performance as we add more dbservers in both the random and load balanced models. With higher session load, the performance increase is more dramatic in the sequential and substantially notable in the load balanced model.

Performance Gain Among Different Allocation Methods

The highest average delay reduction reported occurs under the load balanced method with a load of 200 concurrent sessions when moving from one dbserver to four dbservers, delivering a reduction of over 99%. When testing the decrease in average delay, the load balanced method never drops below 92% when moving from one dbserver to two dbservers and then to four dbservers under any load from 50 concurrent connections to 400 concurrent connections.

The random method, when moving from one dbserver to two dbservers under a load of 200 concurrent sessions, is attributed with the lowest record delay increase of 180%. The random method demonstrates a promising decrease in average delay under a load of 100 concurrent sessions by peaking with a reduction of more than 85% when moving from one dbserver to four dbservers.

The sequential method initially demonstrates a decrease in average delay of 64% and 81% for a 50 concurrent connection model moving from one dbserver to two dbservers and then to four dbservers respectively. The sequential model demonstrates consistent decrease in average delay when moving from one dbserver to two dbservers and to four dbservers under any load from 50 concurrent connections to 400 concurrent connections. The only exception occurs with 200 concurrent sessions when moving from one dbserver to two dbservers resulting in an increase in average delay of 8%.

Impact of Client Intensity on Design Methodology

Higher loads result in inconclusive over saturation of server utilization. Noticeable difficulty was observed when sustaining 800 concurrent sessions of network requests originating from a single Siege client. Additional Siege clients were utilized by distributing the number of concurrent sessions evenly among the two Siege clients. When adding additional Siege clients it was clear that the four dbserver model was not sufficient to handle that number of requests. Often servers would cease functioning when their active process count rose above 285 processes. Siege would also pause for

indefinite periods of time when not enough query requests were acknowledged. This had detrimental effects on the sequential and random models as the Siege client could not issue new requests to available servers when it was waiting for acknowledgment of prior requests sent for processing by saturated servers. There seemed to be no immediate saturation concerns with the main query distribution server. It was purposefully appropriated as a higher end system to alleviate any bottleneck in the overhead needed to execute the PHP server distribution calls. Server recovery time was also a noteworthy concern. In most instances it was not necessary to restart the db servers between test intervals. However, there appeared to be a two to five minute blackout time when it was advisable not to initiate additional siege queries upon completion of a previous test. The server had to reclaim resources until it could resume a steady state. There were a few tests where Siege would throw errors rather than persisting through each session for results. Occasionally tests were completed before the 100,000 packet goal was reached. This would often indicate that one of the servers had engaged a security policy and disabled the HTTP process.

Recommended Combination of Servers and Query Distribution Method

Clearly the load balanced method has outperformed the random and even the sequential model. Possible enhancements to the apparatus might include the following two methods: (1) Doubling the db servers from four to eight and running two web servers, each serving different applications, and dynamically allocating db servers to web server applications as needed, and then releasing the db server to the other allocation server when load increases as web client demand increases, (2) increasing the number of db servers to 32 running the load balanced method and testing each power of two using 2, 4, 8, 16 and 32 db servers under a load of 400, 800, 1600 concurrent connections using four to eight siege clients distributing the concurrent sessions among the siege clients evenly.

Recommendations for Further Research

Modify to the Apparatus and or Methodology

Pretest each of the servers to determine if they are performing within a tolerable level prior to each test. This can be a 50 concurrent session test executed directly against each db server concurrently or successively. Determine statistical variance among each of the loads. Determine the cause of peak performance for the load balanced model to be at 200 concurrent sessions and then diminishing with 400 concurrent sessions.

Demonstrate Scalability by Increasing Db servers

It is clear as the number of dbrowsers increases there is a corresponding increase in the performance. Determine the required load to maximize justification for adding each additional dbrowser. Ask: At what point would it be advisable to add additional web servers with segmented or dynamically allocated dbrowser arrays?

Increase Client Intensity

Currently one Siege client can generate enough concurrent sessions to model 1600 clients distributed as eight sets of 200 concurrent sessions within a five minute interval.

Additional client load would require adding an additional Siege client and distributing the load evenly among the two clients. Data can be collected on each Siege client using TCPDUMP. The Data can then be interpreted and a unique port address can be assigned to each session to enable session time and packet throughput analysis.

It certainly appears that additional database nodes can result in increased performance. This is especially true when a load balanced algorithm is used. However, it would be expected that at some level a point of diminishing returns would be reached. The data collected herein does not address that point. Additional research is needed to address that question. Therefore, because only a small number of nodes were used, this study is more significant in prototyping the process than in obtaining scaling data.

References

- [1] Anthes, G. (2003). "Grids Extend Reach", Computerworld, 37(41), pp. 29-30.
- [2] Elnikety, S. et al. (2004). "A method for transparent admission control and request scheduling in e-commerce web sites", Proceedings of the 13th international conference on World Wide Web, pp. 276-286.
- [3] Guster, D., Safonov P., Hall C., Sundheim R. (2003). "Using Simulation to Predict Performance Characteristics of Mirrored Hosts Used to Support WWW Applications", Issues in Information Systems. 4 (2), 2003.
- [4] Johnson, M. (2003). "Gridlock Reality", Computerworld, 37(41), p 24.
- [5] Kanitkar, V. & Delis, A. (2002). "Distributed Query Processing on the Grid", IEEE Transactions on Computers, 51(3), pp.269-278.
- [6] Roussopoulos, N., Economou, N. & Stamenasm, A. (1993). "A Testbed for Incremental Access Methods", IEEE Transactions on Knowledge and Data Engineering, 5(5) pp.762-774.
- [7] Smith, J. (2003). "Distributed Query Processing on the Grid", International Journal of High Performance Computing Applications, 17(4), pp.353-367.
- [8] Townsend, M. & Tsai, J. (2003). "Oracle 9i New Features", Oracle Corporation, Redwood City, CA.
- [9] Peddemors, A. J. H. & Hertzberg, L. O. (1998). "A high performance distributed database system for enhanced Internet services", Future Generation Computer Systems. Volume 15, Issue 3 , 1 April 1999, Pages 407-415.
- [10] Sobol, Marion G., Kagan, Albert & Shimura, Hirohisa (1996). "Performance criteria for relational databases in different normal forms", Journal of Systems and Software, Volume 34, Issue 1 , July 1996, Pages 31-42.
- [11] Amiri, Ali (2003). "A coordinated planning model for the design of a distributed database system", Information Sciences, Volume 164, Issues 1-4 , 2 August 2004, Pages 229-245.
- [12] Li, Wen-Syan, Altintas, Kemal & Kantarciolu, Murat (2004). "On demand synchronization and load distribution for database grid-based Web applications", Data & Knowledge Engineering, Volume 51, Issue 3 , December 2004, Pages 295-323.
- [13] Welsh, Matthew David (2002). "An architecture for highly concurrent, well-conditioned Internet services", University of California, Berkeley; 0028, DAI, 64, no. 02B (2002): p. 819.
- [14] Simhaa, Rahul & Majumdarb, Amitava (1997). "An urn model with applications to database performance evaluation", Computers & Operations Research, Volume 24, Issue 4, April 1997, Pages 289-300.
- [15] Xiong, Ming, Ramamritham, Krithi, Haritsa, Jayant R. & Stankovic, John A. (2001). "MIRROR: a state-conscious concurrency control protocol for replicated real-time databases", Information Systems, Volume 27, Issue 4, June 2002, Pages 277-297.

- [16] Wu, Chienwen & Befford, Geneva G. (1996). "Improving the flexibility for replicated data management in distributed database systems", *Computers & Industrial Engineering*, Volume 31, Issues 3-4, December 1996, Pages 901-905. 18th International Conference on Computers and Industrial Engineering.
- [17] Soleimany, Cyrus & Dandamudi, Sivarama P. (2002). "Performance of a distributed architecture for query processing on workstation clusters", *Future Generation Computer Systems*, Volume 19, Issue 4, May 2003, Pages 463-478. Selected papers from the IEEE/ACM International Symposium on Cluster Computing and the Grid, Berlin-Brandenburg Academy of Sciences and Humanities, Berlin, Germany, 21-24 May 2002.
- [18] Cannataro, Mario, Talia, Domenico & Srimani, Pradip K. (2002). "Parallel data intensive computing in scientific and commercial applications", *Parallel Computing*, Volume 28, Issue 5, May 2002, Pages 673-704.
- [19] Jutla, Dawn, Bodorik, Peter and Wang, Yie (1999). "Developing internet e-commerce benchmarks", *Information Systems*, Volume 24, Issue 6, September 1999, Pages 475-493. *Information Systems Support for Electronic Commerce*.
- [20] Rajamani, Karthick (2002). "Multi-tier caching of dynamic content for database-driven Web sites", *Rice University: 0187, DAI*, 63, no. 03B (2002): p. 1433.
- [21] Kanitkar, Vinay Vasant (2000). "Collaborative and real-time transaction processing techniques in client-server database architectures", *Polytechnic University: 0179, DAI*, 61, no. 04B (2000): p. 2036.
- [22] Huaa, Kien A., Tavanaponga, Wallapak and Lob, Yu-Lung (1999). "Performance of Load Balancing Techniques for Join Operations in Shared-Noting Database Management Systems", *Journal of Parallel and Distributed Computing*, Volume 56, Issue 1, January 1999, Pages 17-46.
- [23] Fricksa, Ricardo M., Puliafito, Antonio & Trivedic, Kishor S. (1999). "Performance analysis of distributed real-time databases", *Performance Evaluation*, Volume 35, Issues 3-4, May 1999, Pages 145-169.
- [24] Keyes, Jessica (1998). "Datacasting How to Stream Databases over the Internet", McGraw-Hill, 1998.

Appendix A

Siege

<http://www.joedog.org/siege/>

Siege is an http utility designed to benchmark web server code under loads common or otherwise to internet loads. Siege supports basic forms of authentication, cookies, standard HTTP and SSL (HTTPS) protocols. The main feature utilized by this model is Siege's ability to hit a web server with a set number of concurrent simulated users.

TCPDUMP

<http://www.tcpdump.org/>

TCPDUMP is a utility for capturing packet headers over a designated network interface. For the purposes of this experiment, the output was saved to a file using the `-w` flag. TCPDUMP also offers the ability to designate which parts and much of the packet header is to be saved. The number of packets to capture parameter was set to 100,000 packets, and in some cases resulted in less than this threshold.

Appendix B

PHP Code

PHP:

Sample of the PHP code used on the Apache server:

```
<?php

#####
// Settings
#####

$servers = 4; //Number of Servers
/-- $test = true for testing output

#####
//Function app
#####
define("APP_DATA_FILE",
      "/tmp/application.data");

define("LOAD_DATA_FILE",
      "/tmp/mp.txt");

function application_start ()
{
    global $_APP;

    // if data file exists, load application
    // variables
    if (file_exists(APP_DATA_FILE))
    {
        // read data file
        $file = fopen(APP_DATA_FILE, "r");
        if ($file)
        {
            $data = fread($file,
                filesize(APP_DATA_FILE));
            fclose($file);
        }

        // build application variables from
        // data file
        $_APP = unserialize($data);
    }
}

function application_end ()
{
    global $_APP;

    // write application data to file
    $data = serialize($_APP);
    $file = fopen(APP_DATA_FILE, "w");
    if ($file)
    {
        fwrite($file, $data);
    }
}
```

```

        fclose($file);
    }
}

function load_start ()
{
    global $_LOAD;

    // if data file exists, load application
    // variables
    if (file_exists(LOAD_DATA_FILE))
    {
        // read data file
        $file = fopen(LOAD_DATA_FILE, "r");
        if ($file)
        {
            $data = fread($file,
                filesize(LOAD_DATA_FILE));
            fclose($file);
        }

        // build application variables from
        // data file
        $_LOAD = $data;
    }
}

function load_end ()
{
    // write application data to file
    $data = "";
    $file = fopen(LOAD_DATA_FILE, "w");

    if ($file)
    {
        fwrite($file, $data);
        fclose($file);
    }
}

function reverse(&$inarray ) {
    for( $i = 0; $i < sizeof( $inarray , 1); $i++ )
        $outarray[ $i ] = $inarray[ sizeof( $inarray ) - $i - 1 ];
    $inarray = $outarray;
}

if ($function == "sequential") {
    #####
    // Sequential server selection
    #####
    //echo "sequential";
    application_start();

    if ($_APP["serverID"]++ >= $servers+69) {
        $_APP["serverID"] = 70;
    }
    elseif ($_APP["serverID"] < 70) {
        $_APP["serverID"] = 70;
    }
    $URL = "http://199.17.59.".$_APP["serverID"] . "?id=parameter";
    application_end();
}

```

```

        //echo $URL;
        header ("location: $URL");
    }
    elseif ($function == "load") {
    //#####
    // Server selection by load
    //#####

    if ($test=="true")
    {
        echo "load (TeSt MoDe)<BR>=====<br>";
    }
    $lowestUtilization = "100";
    $lowestUtilizationSvr = 1;
    load_start();
    //-----
    // Build Array
    //-----
    $delim = "%\n\n";
    $loadArray = explode($delim,$_LOAD);
    reverse( $loadArray);
    $i = 0;
    //-- Initilization of server utilization Array
    for ($count=1; $count <= $servers+1; $count++)
    {
        $serverUtilization[] = "100";
    }

    //-- iterating through array
    //--      • Finding most recent utilization values
    $ellipsis = "...<br>";
    while (list($IndexValue, $ElementContents) = each($loadArray))
    {
        $i++;
        // -- Get Server
        $serverID =
str_replace("db","",str_replace(".", "", strrev(strchr(strrev($loadArray[$i]), ".
"))));
        if ($serverID > 0)
        {
            // -- Get utilization percent
            $utilization = abs(strchr($loadArray[$i],"\t")/100);
            if ($test=="true" && $i < 20)
            {
                echo "Server($serverID) = $utilization<br>";
            }
            elseif ($test == "true")
            {
                echo "$ellipsis";
                $ellipsis = "";
            }
            if ($serverUtilization[$serverID] == "100")
            {
                $serverUtilization[$serverID] = $utilization;
            }
        }
    }
}

//-- Saving utilization in Application Session Variables and finding lowest
utilization
application_start();
for ($count=1; $count < $servers+1; $count++)

```

```

{
  if ($serverUtilization[$count] == "100" && $_APP[$count] > 0)
  {
    $serverUtilization[$count] = $_APP[$count];
  }
  $_APP[$count] = $serverUtilization[$count];
  if ($test=="true")
  {
    echo "Application Session-->SERVER($count) = $_APP[$count]<BR>";
  }
  // -- Determining lowest utilization
  if ($lowestUtilization > $serverUtilization[$count])
  {
    $lowestUtilization = $serverUtilization[$count];
    $lowestUtilizationSvr = $count;
  }
}

if ($test=="true")
{
  echo "LOWEST UTILIZATION is SERVER $lowestUtilizationSvr @
$lowestUtilization Utilization<br>";
}
application_end();
$lowestUtilizationSvr += 69;
$URL = "http://199.17.59.".$lowestUtilizationSvr . "/?id=parameter";

/-- Check value of $i > 100 then flush load file
if ($i > 100)
{
  load_end();
}
if ($test=="true")
{
  echo "Redirect to ---> $URL";
}
else
{
  header ("location: $URL");
}

}
else {
  /#####
  // Random server selection
  /#####
  //echo "random";
  $r = rand(0,$servers-1);
  $URL = "http://199.17.59.7$r" . "/?id=parameter";
  //echo $URL;
  header ("location: $URL");
}

}

application_end();

exit;

?>

```

Output test screen on browser under no load (4 servers):

```
load (TeSt MoDe)
=====
Server(1) = 0.04
Server(2) = 0.02
Server(3) = 0.02
Server(4) = 0
Server(1) = 0
Server(2) = 0
Server(3) = 0.03
Server(4) = 0.01
Server(1) = 0.01
Server(2) = 0.01
Server(3) = 0.01
Server(4) = 0.01
Server(1) = 0
Server(2) = 0.01
Server(3) = 0.01
Server(4) = 0.04
Server(1) = 0.01
Server(2) = 0.04
Server(3) = 0.01
...
Application Session-->SERVER(1) = 0.04
Application Session-->SERVER(2) = 0.02
Application Session-->SERVER(3) = 0.02
Application Session-->SERVER(4) = 0
LOWEST UTILIZATION is SERVER 4 @ 0 Utilization
Redirect to ---> http://199.17.59.73/?id=parameter
```

Output test screen on browser under no load (2 servers):

```
load (TeSt MoDe)
=====
Server(3) = 0.01
Server(4) = 0.01
Server(1) = 0.04
Server(2) = 0
Server(3) = 0.03
Server(4) = 0.01
Server(1) = 0
Server(2) = 0.01
Server(3) = 0.01
Server(4) = 0
Server(1) = 0.01
Server(2) = 0.01
Server(3) = 0.01
Server(4) = 0.04
Server(1) = 0.01
Server(2) = 0.01
Server(3) = 0.01
Server(4) = 0.02
Server(1) = 0.04
...
Application Session-->SERVER(1) = 0.04
Application Session-->SERVER(2) = 0
```

```
LOWEST UTILIZATION is SERVER 2 @ 0 Utilization
Redirect to ---> http://199.17.59.71/?id=parameter
```

Output test screen on browser under full load (8 iterations of 400 concurrent sessions using 4 servers):

```
load (TeSt MoDe)
=====
Server(3) = 0
Server(4) = 0.01
Server(3) = 0.05
Server(2) = 0.69
Server(1) = 0.97
Server(4) = 0.01
Server(3) = 0.01
Server(2) = 0.49
Server(4) = 0.04
Server(1) = 0.01
Server(3) = 0.01
Server(2) = 0.01
Server(4) = 0.01
Server(1) = 0.04
Server(3) = 0
Server(2) = 0.04
Server(4) = 0
Server(1) = 0
Server(3) = 0.02
...
Application Session-->SERVER(1) = 0.97
Application Session-->SERVER(2) = 0.69
LOWEST UTILIZATION is SERVER 2 @ 0.69 Utilization
Redirect to ---> http://199.17.59.71/?id=parameter
```

Output test screen on browser under full load (8 iterations of 400 concurrent sessions using 2 servers):

```
load (TeSt MoDe)
=====
Server(3) = 0
Server(2) = 0.01
Server(4) = 0.01
Server(3) = 0.02
Server(2) = 0.01
Server(1) = 0.99
Server(4) = 0
Server(3) = 0
Server(2) = 0.01
Application Session-->SERVER(1) = 0.99
Application Session-->SERVER(2) = 0.01
Application Session-->SERVER(3) = 0
Application Session-->SERVER(4) = 0.01
LOWEST UTILIZATION is SERVER 3 @ 0 Utilization
Redirect to ---> http://199.17.59.72/?id=parameter
```

PHP:

Sample of the PHP code used on each of the dbservers:

```
<html>
<?php

    $DBhost = $_SERVER['SERVER_ADDR'];
    $DBuser = "sa";
    $DBpass = "sa";

    # Connect to the DataBase
    $link = mysql_connect($DBhost, $DBuser, $DBpass)
    or die("Unable to connect to database");

print "<head><title>Connect Server $DBhost</title></head>\r\n";
print "<body>\r\n";

print "$DBhost<br>";
echo "id= $id <br>";

#####
// 1) Continent
#####
$sql[] = "SELECT cont AS Continent \r\n ".
        "FROM hamlog \r\n " .
        "GROUP BY Continent \r\n ";
#####
// 2) State
#####
$sql[] = "SELECT state AS State \r\n ".
        "FROM hamlog \r\n " .
        "GROUP BY State \r\n ";
#####
// 3) Band
#####
$sql[] = "SELECT DISTINCT band AS Band \r\n ".
        "FROM hamlog \r\n " .
        "GROUP BY Band \r\n ";
#####
// 4) TOP 50 Counties & States
#####
$sql[] = "SELECT \r\n ".
        "    count(*) as c, \r\n ".
        "    cnty AS County, \r\n ".
        "    state AS State \r\n ".
        "FROM hamlog \r\n ".
        "GROUP BY \r\n ".
        "    State, \r\n ".
        "    County \r\n ".
        "HAVING \r\n ".
        "    LENGTH(County) > 0 AND \r\n ".
        "    LENGTH(State) > 0 \r\n ".
        "ORDER BY \r\n ".
        "    c DESC, \r\n ".
        "    State, \r\n ";
```

```

        " County \r\n ".
        "LIMIT 50 \r\n ";
#####
// 5) Power
#####
$sql[] = "SELECT pwr AS Power \r\n ".
        "FROM hamlog \r\n " .
        "GROUP BY Power \r\n ";
#####
// 6) TOP 25 Grids
#####
$sql[] = "SELECT \r\n " .
        " count(grid) AS c, \r\n " .
        " grid AS Grid \r\n " .
        "FROM hamlog \r\n " .
        "GROUP BY Grid \r\n " .
        "ORDER BY c DESC \r\n " .
        "LIMIT 25 \r\n ";
#####
// 7) Top 25 Names
#####
$sql[] = "SELECT \r\n " .
        " COUNT(name) AS c, \r\n " .
        " name AS Name \r\n " .
        "FROM hamlog \r\n " .
        "GROUP BY \r\n " .
        " Name \r\n " .
        "ORDER BY \r\n " .
        " c DESC, \r\n " .
        " Name \r\n " .
        "LIMIT 25 \r\n ";
#####
// 8) Select 1 random record
#####
$sql[] = "SELECT * \r\n " .
        "FROM hamlog \r\n " .
        "ORDER BY rand() \r\n " .
        "LIMIT 1 \r\n ";
#####
// 9) Select 10 random records
#####
$sql[] = "SELECT * \r\n " .
        "FROM hamlog \r\n " .
        "ORDER BY rand() \r\n " .
        "LIMIT 10 \r\n ";
#####
// 10) Select 50 random records
#####
$sql[] = "SELECT * \r\n " .
        "FROM hamlog \r\n " .
        "ORDER BY rand() \r\n " .
        "LIMIT 50 \r\n ";

$r = rand(0,count($sql)-1);
//$r = 5;
echo "=====  
";

```



```

echo "SQL #\$r<br>";
echo "=====  
<br>";
echo "<pre>\$sql[\$r]</pre>";
//exit;

$db="db1";
if (! $link)
die("Couldn't connect to MySQL");
mysql_select_db($db , $link)
or die("Couldn't open $db: ".mysql_error());
$result = mysql_query( $sql[\$r] )
or die("SELECT Error: ".mysql_error());
$num_rows = mysql_num_rows($result);
print "<br>";
print "$num_rows record(s) found.<P>";
print "<table width=200 border=1>\n";
while ($get_info = mysql_fetch_row($result)){
print "<tr>\n";
foreach ($get_info as $field)
print "\t<td><font face=arial size=1/>$field</font></td>\n";
print "</tr>\n";
}
print "</table>\n";
mysql_free_result($result);
mysql_close($link);
?>
</body>
</html>

```

Output dbserver query results (i.e. sample data of packet payload contents) :

The data was drawn (with permission) from the amateur radio station log of Lee Lorentz, licensed as WBØTRA.

Sample #1

199.17.59.70

id=

=====

SQL #7

=====

```
SELECT *
FROM hamlog
ORDER BY rand()
LIMIT 1
```

1 record(s) found.

888	N5RL	2002	21553	5	5	ESTHER	SAN	US	14.28047	0	20	T	K	X	X	WBØTR	SSSBØ	10	ELØ9R	4	7	BEXA	N	0
0	Q	-11-	1	9	9	GOLLIHA	ANTONI	B	7		M	X			A	2	0	H			R	5		
		17				R	O																	

Sample #2

199.17.59.70

id=

=====

SQL #9

=====

```
SELECT *
FROM hamlog
ORDER BY rand()
LIMIT 50
```

50 record(s) found.

759	L75F	200	004	5	5			USB	21.34	0	1	LU	F	X	WBO	WPXSB	1		S	1			L	0
1	M	2-	022	9	9				788	5				TRA	02	0			A	3			7	

		03-30							M									5	
5023	W7BAS	2001-10-27	003822		59	59	BRUCE	KENMORE	USB	28.709	00M	WAKXX	WBO TRA	CQWWS B01	100	CN87VS	NA3		W70
2683	AC7DU	2000-08-05	042500	042500	59	57	BRUCE	BOISE	SSB	50.125	06M	IDKFF	WBO TRA		100	DN13UN	NA36		AC70
6565	WJOM	2002-02-02	145031		59	59	DICK	DULUTH	LSB	7.25	00M	MNKXX	WBO TRA	MNQPO2	100		NA47		WJ00
6108	WA3JMV	2002-01-19	195530		59	59	STAN	BROOMALL	USB	28.427	00M	PAXXX	WBO TRA	NAQPS B01	100	FM29HX	58	DELAWARE	WA30
4475	NA3DX	2001-06-23	194457		59	59	EXPLORERS	LOTHIAN	CW	14.01462	00M	MDKFF	WBO TRA	FD2001	100	FM18QT			NA30
7974	OL5Q	2002-03-31	172417		59	59	CONTEST STN	QSL VIA OK1HRA	USB	28.3941	00M	OLFF	WBO TRA	WPXSB02	100	JO60VI	EU128		OL50
144	HA3MN	1996-08-02	220000	220100	57	57	EGON	HUNGARY	SSB	14.182	00M	HAFX	WBO TRA		80		EU128		HA30
8106	PS5S	2002-03-31	221140		59	59			USB	28.48358	00M	PYXX	WBO TRA	WPXSB02	100		SA11		PS50

7467	JH7A FR	200 2-03-03	224 355		59	59	TETSUO BABA	JAPAN	USB	28.37 5	10 0M		JA	X	X	WBO TRA	ARDXS B02	100 0		A S	25	45				J H 7	0
8498	W0SD	200 2-11-16	221 134		59	59	EDWARD GRAY	SALEM	USB	21.36 652	15 0M		S D	K	F	F	WBO TRA	SSSB0 2	100 0	EN1 3GP		47	MCCOOK			W 0	0
3707	KE6R D	200 1-03-24	194 821		59	59	KATSUM I	TORRANCE	USB	28.33 4	10 0M		C A	K	X	X	WBO TRA	WPXSB 01	100 0	DM0 3UT	N A	3				K E 6	0
7773	SN2B	200 2-03-30	165 818		59	59			USB	21.21 34	15 0M		SP	X	X	WBO TRA	WPXSB 02	100 0		E U	15					S N 2	0
3846	KC7C XR	200 1-03-25	033 606		59	59	ERIC	PULLMAN	USB	14.28 4	20 0M		W A	K	F	F	WBO TRA	WPXSB 01	100 0	DN1 6JR	N A	36	WHITMA N			K C 7	0
6125	K1GL J	200 2-01-19	201 119		59	59	GREG	FLORENCE	USB	28.42 7	10 0M		V T	K	X	X	WBO TRA	NAQPS B01	100 0	FN3 3LR		58	RUTLAN D			K 1	0
9470	PA0I JM	200 2-12-15	162 134		59	59	J KIKKER T	NETHERLA NDS	USB	28.35 6048	10 0M		PA	F	F	WBO TRA	02ARL 10M	100 0		E U	14	27				P A 0	0
8222	OD5N H	200 2-05-30	023 817	023 900	59	55	PUZANT AZIRIA N	BEIRUT	USB	18.13	17 0M		OD	F	F	WBO TRA		100 0	KM7 3SV	A S	20	39				O D 5	0
3007	W4AN	200 0-	064 600		59	59	BILL	ALPHARET TA	LSB	3.78	80 0		G A	K	F	F	K0B LR	SSSB2 000	100 0	EM7 4VC		58	LUMPKI N			W 4	0

4380	KB8SCR	2001-06-23	180403		59	59	DAVID	FORT RECOVERY	USB	14.312	020M		O	K	X	X	WBO TRA	FD 2001	100	EN700K									KB8	0					
2448	RZ1A WO	2000-03-05	013700	013700	59	59		RUSSIA	SSB	14.3314	020M				U	A	F	X	WBO TRA	00ARL DXP	100			E	16					RZ1	0				
7760	AK3Z	2002-03-30	150322		59	59	JOHN	WESTMINSTER	USB	14.206	020M		M	D	K	X	X	WBO TRA	WPXSB 02	100	FM19MN		N	5					AK3	0					
3514	RD4M	2001-03-18	030327	030400	59	59		RUSSIA	USB	14.16035	020M				U	A	F	F	WBO TRA		100	LO44TF		E	16	19				RD4	0				
7613	VA7D P	2002-03-30	011648		59	59	G DOUGLAS PICHET	PENTICTON	USB	21.2051	015M				V	E	X	X	WBO TRA	WPXSB 02	100			N	3			B	C	VA7	0				
1973	W2GG	1999-11-21	063504	063504	59	59	ROBERT	SPARKS	SSB	3.819	080M		M	D	K	F	F	WBO TRA	SWPS99	100	FM19PN		N	58		BALTIMORE			W2	0					
8451	K6GT	2002-11-16	211431		59	59	GEORGE	SUNNYVALE	USB	28.398795	010M		C	A	K	X	X	WBO TRA	SSSB02	100	CM87XI								3	6	SANTA CLARA			K6	0
10315	KG4N OZ	2003-02-01	210704		59	59	RON	LAWRENCE BURG	USB	14.236	020M		T	N	K	F	F	WBO TRA	MNQP2003	100	EM65IF		N	48		LAWRENCE				KG4	0				
1205	N4PN	1998-	173203	173203	59	59	PAUL	ST GEORGE	SSB	21.426	015		F	L	K	F	F	WBO TRA	98CQW WDX	10			N	58		FRANKLIN		N	4	N	4	0			

		08-27														0												
2349	V47K P	200 0-03- 04	191 300	191 300	5 9	5 9		ST KITTS / NEVIS	SSB	21.30 4	0 5 M		V4	F	F	WB0 TRA	00ARL DXP	1 0 0		N A	8	1 1		V 4 7	0			
663	N9AF	199 7-03- 09	214 900	214 900	5 9	5 9	FREDE RICK	PLAINFIE LD	SSB	3.868	0 0 M	I N	K	F	F	WB0 TRA	WI QSO	1 5 0	EM6 9TR	N A	5	8	HENDRI CKS		N 9	0		
3779	KD5F KY	200 1-03- 24	222 204		5 9	5 9	JAMES	WACO	USB	21.25	0 5 M	T X	K	X	X	WB0 TRA	WPXSB 01	1 0 0	EM1 1JN	N A		4			K D 5	0		
4167	N2MR	200 1-04- 21	024 221		5 9 9	5 9 9	MARK SMITH	FRANKLIN VILLE	PSK 31	14.06 95	0 0 M	N J	K	X	X	WB0 TRA	PSK20 01	2 5	FM2 9LO	N A	5			GLOUCE STER		N 2	0	
9167	N2BJ	200 2-12- 14	182 616		5 9	5 9	BARRY	NEW LENOX	USB	28.68 5083	0 0 M	I L	K	X	X	WB0 TRA	02ARL 10M	1 0 0	EN6 1AM						N 2	0		
10355	KB0L XE	200 3-02- 01	150 700		5 9	5 9	GARY	BIG LAKE	USB	50.13	0 6 M	M N	K	X	X	WB0 TRA	MNQP2 003	1 0 0		N A	4	7			SHERBU RNE		K B 0	0
1267	VP2E	199 8-10- 25	222 550	222 550	5 9	5 9	DX CONTES T GROUP	LEEWARD ISLANDS	SSB	21.27	0 5 M		VP 2- E	F	X	WB0 TRA	98CQW WDX	1 0 0		N A	8	1 1			V P 2	0		

Sample #3

199.17.59.70

id=

=====

SQL #3

=====

```
SELECT
    count(*) as c,
    cnty AS County,
    state AS State
FROM hamlog
GROUP BY
    State,
    County
HAVING
    LENGTH(County) > 0 AND
    LENGTH(State) > 0
ORDER BY
    c DESC,
    State,
    County
LIMIT 50
```

50 record(s) found.

78	STEARNS	MN
73	KING	WA
61	HENNEPIN	MN
57	SANTA CLARA	CA
55	WRIGHT	MN
54	WAKE	NC
49	LOS ANGELES	CA
48	MEEKER	MN
38	HARRIS	TX
36	DALLAS	TX
34	SAN DIEGO	CA
32	MONTGOMERY	MD
29	MIDDLESEX	MA
26	SAN MATEO	CA
26	HONOLULU	HI
25	ORANGE	CA
25	OAKLAND	MI
24	MARICOPA	AZ
23	ALAMEDA	CA
23	COOK	IL
23	WESTCHESTER	NY
22	ALLEGHENY	PA

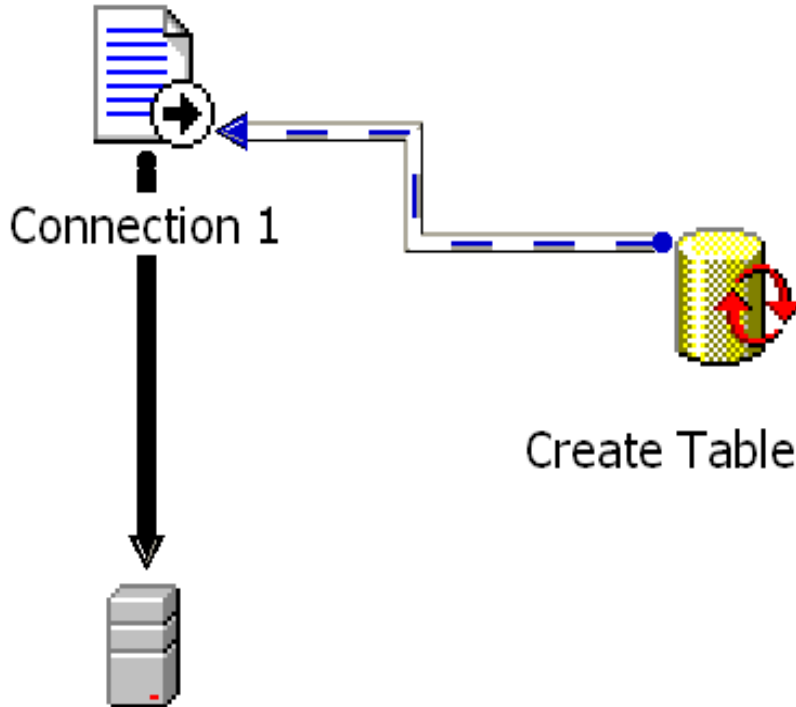
20	HILLSBOROUGH	NH
20	MULTNOMAH	OR
19	YUMA	AZ
19	MESA	CO
19	KANE	IL
19	PRINCE GEORGES	MD
18	NEW HAVEN	CT
18	ADA	ID
18	DUTCHESS	NY
18	PROVIDENCE	RI
18	PIERCE	WA
17	CONTRA COSTA	CA
17	JACKSON	MO
17	SUFFOLK	NY
17	WASHINGTON	OR
17	DAVIDSON	TN
17	CHITTENDEN	VT
16	SACRAMENTO	CA
16	GWINNETT	GA
16	ERIE	NY
15	BOULDER	CO
15	SAINT CLAIR	IL
15	RAMSEY	MN
15	CLACKAMAS	OR
14	SANTA CRUZ	CA
14	FAIRFIELD	CT
14	MARION	IN
14	DAKOTA	MN

Appendix C

Process of Importing TCPDUMP output into Microsoft SQL Server and Microsoft Excel


Import TCPDUMP into MS SQL Server.

DTS Package



DTS: Create Table

General

 You can run SQL code on the selected connection. You must select a connection and then provide the SQL code to execute.

Description:

Existing connection:

Command time-out:

SQL statement:

```
CREATE TABLE [Thesis] [dbo] [logfile_s2_1_load_0] (  
[tStamp] varchar (255) NULL,  
[macSrc] varchar (255) NULL,  
[macDes] varchar (255) NULL,  
[payload] varchar (255) NULL,  
[addressSrc] varchar (255) NULL,  
[dir] varchar (255) NULL,  
[addressDes] varchar (255) NULL,
```

DTS: Connection 1 (Text data file)


General

Specify a new connection or an existing connection to the data source.

New connection:

Existing connection:


Data source:

 Text files can be delimited or fixed field. To connect, you must select a file and then provide the properties that define the file.

File name:


DTS: Transformation

Source | Destination | Transformations | Lookups | Options

 Enter a table name or the results of a query as a data source.


Description:

Connection: Connection 1

Table / View:  ▾

SQL query:

Source Destination Transformations Lookups Options

 Store the results for this transformation.

Connection: Connection 2

Table name: [Thesis].[dbo].[logfile_z_s]

Name	Type	Nullability	Size	Precision	Scale
tStamp	varchar	<input checked="" type="checkbox"/>	255		
macSrc	varchar	<input checked="" type="checkbox"/>	255		
macDes	varchar	<input checked="" type="checkbox"/>	255		
payload	varchar	<input checked="" type="checkbox"/>	255		
addressSrc	varchar	<input checked="" type="checkbox"/>	255		
dir	varchar	<input checked="" type="checkbox"/>	255		
addressDes	varchar	<input checked="" type="checkbox"/>	255		
protocol	varchar	<input checked="" type="checkbox"/>	255		
fragment	varchar	<input checked="" type="checkbox"/>	255		

Source Destination Transformations Lookups Options

Define the transformations between the source and destination.

Name: DTSTransformation_1

Type: Copy Column

New Edit Delete Test

Source	Destination
Col001	tStamp
Col002	macSrc
Col003	macDes
Col004	payload
Col005	addressSrc
Col006	dir
Col007	addressDes
Col008	protocol
Col009	fragment
Col010	

Select All Delete All

OK Cancel Help

DTS: Destination


General

Specify a new connection or an existing connection to the data source.

New connection:

Existing connection:

Data source:

 To connect to Microsoft SQL Server, you must specify the server, username, and password.

Server:

Use Windows Authentication

Use SQL Server Authentication

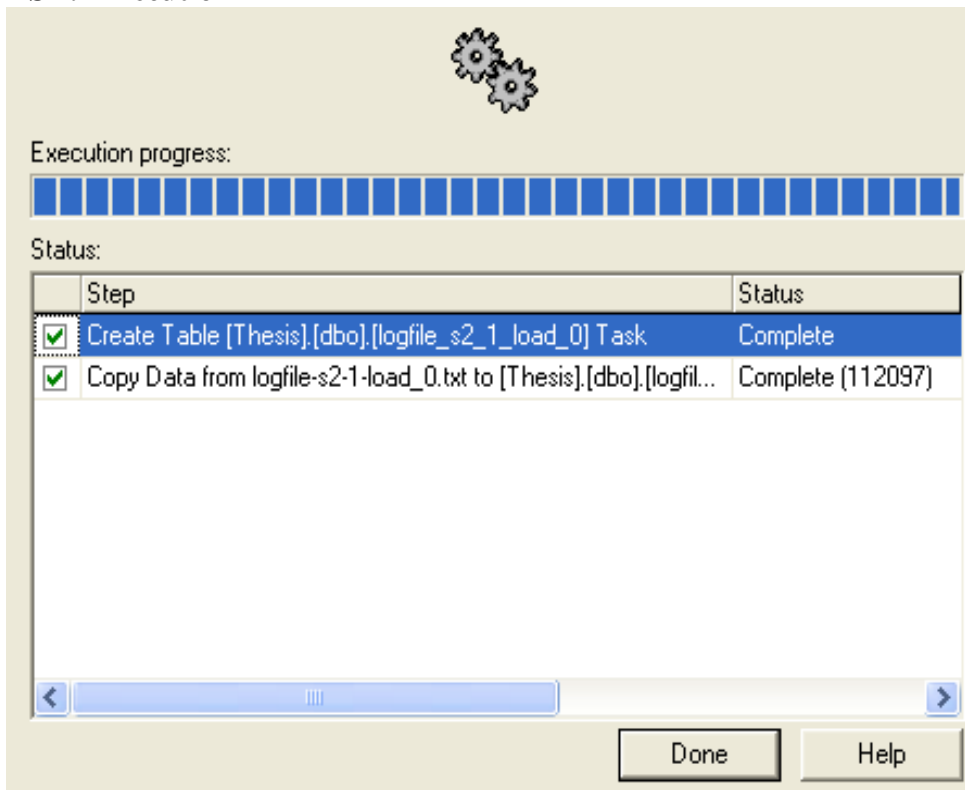
Username:

Password:

Database:

```
-- SQL used to create a data table for each test (NOTE Table name
changes to match log file naming convention.)
CREATE TABLE [Thesis].[dbo].[logfile_s2_1_load_0] (
[tStamp] varchar (255) NULL,
[macSrc] varchar (255) NULL,
[macDes] varchar (255) NULL,
[payload] varchar (255) NULL,
[addressSrc] varchar (255) NULL,
[dir] varchar (255) NULL,
[addressDes] varchar (255) NULL,
[protocal] varchar (255) NULL,
[fragment] varchar (255) NULL
)
```

DST: Execution



Step 3: Create Stored procedures and views used in determining Average Delay, Throughput and Packet Intensity. One set of views, and stored procedure is created for each data set.

i. Drop standard view if already created

```
if exists (select * from dbo.sysobjects where id =  
object_id(N'[dbo].[logfile_s2_1_loadView]') and OBJECTPROPERTY(id,  
N'IsView') = 1)  
drop view [dbo].[logfile_s2_1_loadView]  
GO
```

ii. Create standard view

```
CREATE VIEW dbo.logfile_s2_1_loadView  
AS  
SELECT TOP 100 PERCENT  
tStamp,  
CAST(REPLACE(payload, ':', '') AS int) AS payload,  
addressSrc,  
addressDes,  
  
REPLACE (RIGHT (addressDes, CHARINDEX ('.', REVERSE (addressDes)) - 1  
, ':', '')) as temp,  
CASE RIGHT (addressSrc, 3)
```

```

        WHEN '.80' THEN
REPLACE (RIGHT (addressDes, CHARINDEX('.', REVERSE (addressDes)) - 1
), ':', '')
        ELSE
RIGHT (addressSrc, CHARINDEX('.', REVERSE (addressSrc)) - 1 )
        END
        AS portSrc,
        CAST (LEFT (tStamp, 2) AS decimal (10, 6)) * 3600 +
        CAST (SUBSTRING (tStamp, 4, 2) AS decimal (10, 6)) * 60
+
        CAST (SUBSTRING (tStamp, 7, 9) AS decimal (10, 6)) AS t
FROM      dbo.logfile_s2_1_load
ORDER BY portSrc, tStamp
GO

```

iii. Drop throughput view if already created

```

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[logfile_s2_1_loadView_tp]') and
OBJECTPROPERTY(id, N'IsView') = 1)
drop view [dbo].[logfile_s2_1_loadView_tp]
GO

```

iv. Create throughput view

```

CREATE VIEW dbo.logfile_s2_1_loadView_tp
AS
SELECT      TOP 100 PERCENT portSrc, MAX(t) - MIN(t) AS t,
SUM(payload) AS payload
FROM      dbo.logfile_s2_1_loadView
GROUP BY portSrc
ORDER BY portSrc
GO

```

v. Drop stored procedure if already created

```

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[lf_s2_1_load_stat]') and OBJECTPROPERTY(id,
N'IsProcedure') = 1)
drop procedure [dbo].[lf_s2_1_load_stat]
GO

```

vi. Create statistics stored procedure

```
CREATE PROCEDURE [dbo].[lf_s2_1_load_stat] AS
/*
Convention
=====

                                logfile s2 1 load
                                ----- - - - - -
                                |   |   |   |   |
logfile - initial filename -----/   |   |   |
                                |   |   |   |
z - data from zeus client -----/   |   |   |
                                |   |   |   |
s# - 1 = using 1 server              |   |   |
    2 = using 2 servers              |   |   |
    4 = using 4 servers -----/   |   |
                                |   |   |
# - 1 = 50 concurrent sessions      |   |   |
    2 = 100 concurrent sessions     |   |   |
    3 = 200 concurrent sessions     |   |   |
    4 = 400 concurrent sessions -----/   |
                                |   |   |
Seq - seq = sequence                 |   |   |
    ran = random -----/
*/

--#####
--logfile_s2_1_load
--#####

-- AVERAGE DELAY
SELECT
    (MAX(t) - MIN(t))/count(portSrc) / 2 AS Average_Delay
FROM
    dbo.logfile_s2_1_loadView

-- THROUGHPUT
SELECT
    SUM(payload)/sum(t) AS Throughput
FROM
    dbo.logfile_s2_1_loadView_tp

-- PACKET INTENSITY
SELECT
    count(portSrc)/(MAX(t) - MIN(t)) AS Packet_Intensity
FROM
    dbo.logfile_s2_1_loadView
GO
```

vii. Execute stored procedure

```
lf_s2_1_load_stat
```

viii. Results

```
Average_Delay
-----
.0001662230122126350

(1 row(s) affected)

Throughput
-----
298088.1711530463984497266

(1 row(s) affected)

Packet_Intensity
-----
3008.0070944712852680828

(1 row(s) affected)
```

Step 5: Load data into Excel

- i. Sample Excel column of load balanced data from 4 servers using 8 iterations of 50 concurrent sessions.**

NOTE: Excel data is derived based on the throughput view in step 3 above.

<u>portSeq</u>	<u>time</u>	<u>payload</u>
57964	0.010289	1049
57965	0.147174	31238
57966	0.00319	696
57967	0.002593	1049
57968	0.092772	9705
57969	0.002683	1049
57970	0.217409	13198
57971	0.00261	1049
57972	0.468031	13664
57973	0.002636	1049
57974	0.830308	197832
57975	0.002736	1049
57976	0.895952	18450
57977	0.002433	1049
57978	0.002519	1049
57979	1.011154	31815
.	.	.
.	.	.
.	.	.