CAN DISTRIBUTED DATABASES PROVIDE AN EFFECTIVE MEANS OF SPEEDING UP WEB ACCESS TIMES

CHRISTOPHER G. BROWN
SAINT CLOUD STATE UNIVERSITY
chrisb@stcloudstate.edu

DENNIS GUSTER
SAINT CLOUD STATE UNIVERSITY
dcguster@stcloudstate.edu

SARA KRZENSKI
SAINT CLOUD STATE UNIVERSITY
krsa0601@stcloudstate.edu

ABSTRACT

To support web application performance scalability, it is important to optimize stored data, which can be extracted, processed and forwarded to a web client. Hard drive technology, based on mechanical technology, is the slowest part of the information retrieval. Given the "millions-to-one" mechanical bottleneck, it is reasonable to investigate optimization by storing data on multiple disks, distributed across multiple devices. This methodology suggests a reduction of data access time. With the "millions of hits scenario" and as Internet services continue to grow, research is needed to delineate the performance advantages of distributed databases and the basic models of configuration. This paper used a series of experiments with three different distribution algorithms to determine the potential advantage of a distributed database. It was found that a load balancing algorithm run on four data base nodes could significantly improve performance.

Keywords: distributed databases, load balancing, IO performance, Parallel algorithms

INTRODUCTION

Before applications became web applications (pre-http), the maximum number of potential users was limited to the size of the connected private network, generally a maximum value in the thousands of users. With Internet web applications, it is not unreasonable to expect a value (of potential users) in the millions of users.

To support web application performance scalability, it is important to optimize stored data, which can be extracted, processed and forwarded to a web client. Hard drive technology, based on mechanical technology, is the slowest part of the information retrieval. With state of the art disk drive technology, adequate performance cannot be obtained with the most intensive web applications involving the "millions of hits scenario."

Given this "millions-to-one" mechanical bottleneck, it is reasonable to investigate optimization by stor-

ing data on multiple disks, distributing across multiple devices. This methodology suggests a reduction of data access time; and as Elnikety et al. [4] shows, there was an improvement of throughput by ten percent and a decrease workstation response time by a factor of 14 when distributed databases were employed. Further it appears that there are three variables to consider in trying to optimize a distributed database within a WWW application.

The first variable acknowledged was workload intensity. Intensity increases the need to utilize a form of a distributed database. Kanitkar et al. [9] determined that distributed databases offer significant performance advantages, if the system was large enough, in terms of users. Kanitkar [8] found it takes about 40 users to reach a performance threshold.

The second factor was the number of distributed database nodes. As expected, adding additional nodes reduces access time. However, Guster et al. [5] states a point of diminishing returns occur when the communication overhead among the many nodes negates the performance effect of adding additional nodes.

The third variable acknowledged was 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, was expected to offer the most promise.

Although the concept of the distributed database has been around for over 20 years, it has not become dominate in business-related applications. The complexity and cost of adding database nodes has inhibited development and use [6, pp 24]. Specifically Anthes [2] states that deployed distributed database systems have barely moved beyond scientific, engineering and mathematical/statistical applications.

With the "millions of hits scenario" and as Internet services continue to grow, research is needed to delineate the performance advantages of distributed databases and the basic models of configuration. Smith et al. [15] agrees and specifically states the need for more performance evaluation research with larger databases.

Advantages

Peddemors et al. [11] 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. [16] state that the increase in client-server and other telecommunication-based applications will spur dispersed and distributed processing, and, as a result, 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. [13] 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 databases to the foreground.

Design Considerations

Amiri [1] 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 as well as operations and maintenance in a multi-period environment where user-processing demand varies over time.

Li et al. [10] 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.

Simha et al. [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.

The literature reviewed reveals concerns about maintaining reliability given the added complexity of distributed databases. Xiong et al. [19] 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. [18] agree that reliability is important and, therefore, 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.

Performance Issues

Cannataro et al. [3] 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. The article also encourages the reader to go into the more in-depth articles that appeared later in the special issue journal in which their work was published.

Jutla et al. [7] 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 [12] 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 compared to sites with just static content. As a result, these sites require well thought out designs [17].

Kanitkar [8] 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 assigned 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 coordinated the movement of data and transactions so as to process each transaction at the site that offered the highest probability of successful completion.

Scope of the Study

In the interest of keeping the study feasible and narrowly focused, several parameters were defined to help clarify the test environment used:

Server Operating System Selection. The server operating system selected for this project was Linux because of its openness and high degree of flexibility. Linux offered high performance due to its low overhead and optimized code.

Database Software Selected. The database software selected was MySQL. This software was tuned to the Linux operating system and uses the standard SQL language.

Database Structure. The structure of the database was limited to a single table. The goal of the study was to gain a baseline by varying the number of nodes, the workload, and the distribution algorithm.

Workload Generator. Siege was selected as the workload generator because it was designed to let developers measure performance of their code under "siege" or duress. Siege allows load variation with a configurable number of transactions, the sum of simulated users, and the number of times each simulated user repeats the process of accessing the server.

Distribution Algorithms. Although there is multitude of possibilities, this study focused on three of the most basic: sequential, random and load checking. The sequential method assigns requests in sequence among the allocated nodes verses the random method which assigns requests randomly among the allocated nodes. As for the load checking method, it checks the node to make sure its utilization is less than a certain load threshold.

Size of Cluster and Scaling Pattern. The maximum number of database nodes utilized was limited to four. In terms of scaling, it has been common to use the following pattern to access performance: 1, 2, 4, 8, and 16 processors. This scaling "doubling" pattern has been widely used in other studies; and from a consistency and transferability perspective, it was adopted in this study. These limitations were presented to make the study more manageable in scope and to make it easier for the reader to evaluate/use the results.

METHODOLOGY AND RESULTS

Research Questions

This paper explored the effectiveness of a distributed database under a variety of conditions by conducting experiments using different combinations of variables listed above. Specifically, the following questions were researched:

1. How does the workload intensity influence the need and performance of distributed database applications?

2. How does the number of database nodes affect the data access time?
3. How does the algorithm used to assign a given query to a specific database node influence the access time?

These questions were 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 to the originating client.
H2. The number of nodes a database is stored upon has no affect on response time to the originating client.
H3. The algorithm used to distribute requests to a given distributed database node has no affect on the delay to the originating client.

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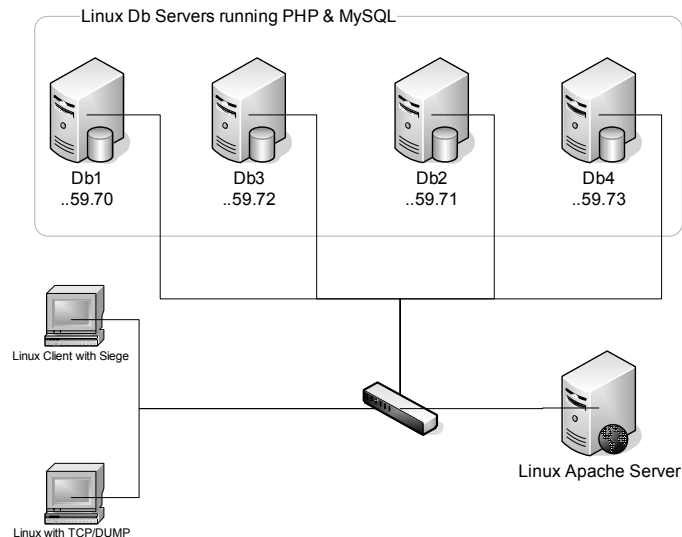

Figure 1: Test-bed Used

The actual collection agent within this environment was a packet sniffer process generated by TCPDUMP. This collection agent trapped data from each packet generated by the experimental tests. The URL's used to test the three methods were sequential, random and load balanced. More detail about these methods is available at http://web.stcloudstate.edu/chrisb/thesis_20050501.doc.

The apache server would then redirect the output based on the predefined algorithm set up for each method. The following variables appeared 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, provided metrics in the following categories: delay to the client, data throughput, and data intensity. A high-end processor running Linux generated the workload. The software used was Siege, which was able to generate web traffic streams of varying intensity. For the experiments run herein, the traffic of eight consecutive groups of 50, 100, and 200 clients was generated in three separate tests. The client requests were forwarded to a Linux web server via a 100 Mbps Ethernet network. The web server, 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 return 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: server one, then two, then three, then four, then back to one. The random method used a random number generator to select a dbserver randomly from the pool of servers. It was expected that if the number generator were truly random, the workload would be 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.

The data collected was reported in a series of Tables. Tables one through three appear below:

Table 1: 8 Consecutive Iterations of 50 Concurrent Sessions.

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

The data that was collected at the 50 client level is displayed in Table 1. At the 50-client level, each test was performed once per method and dbserver node configuration. As the session load increased, the performance difference was amplified and, as a result, suggested a higher performance return per additional dbserver node.

- The first column describes the database node allocation method. This concept is not applicable when only one dbserver is used.
- The second column describes the number of times that the simulated 50 clients generated a request stream.
- The third column depicts the number of database servers used.
- The fourth column reports the number of simulated clients generating the workload.
- The fifth column reports the average delay back to the client in filling the request
- The sixth column depicts the throughput in bytes per second.
- The last column reports the intensity of packet traffic.

Table 2: 8 Consecutive Iterations of 100 Concurrent Sessions.

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

Table 3: 8 Consecutive Iterations of 200 Concurrent Sessions.

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

A comparison of values at the various client levels is best demonstrated graphically and Figures 2-10 will depict the values observed on average delay, throughput, and packet intensity. Figures 2-4 depict average delay in respect to the sequential, random, and load-balanced methods respectively. Detailed plots of session times and packet payloads for the sequential, random, and load-balanced models by loads of 50, 100, and 200 concurrent sessions are available on request.
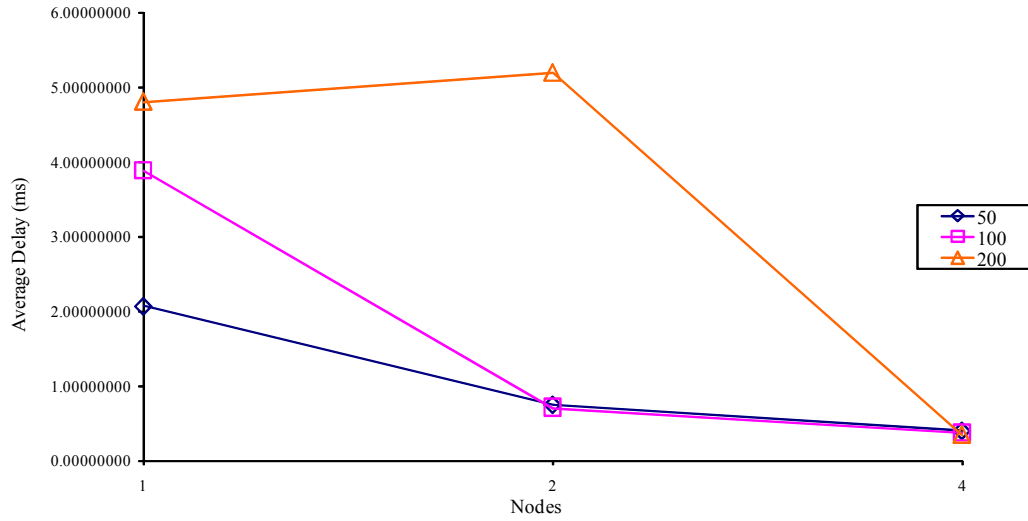
Figure 2: Series of Concurrent Sessions. Sequential Nodes vs. Average Delay
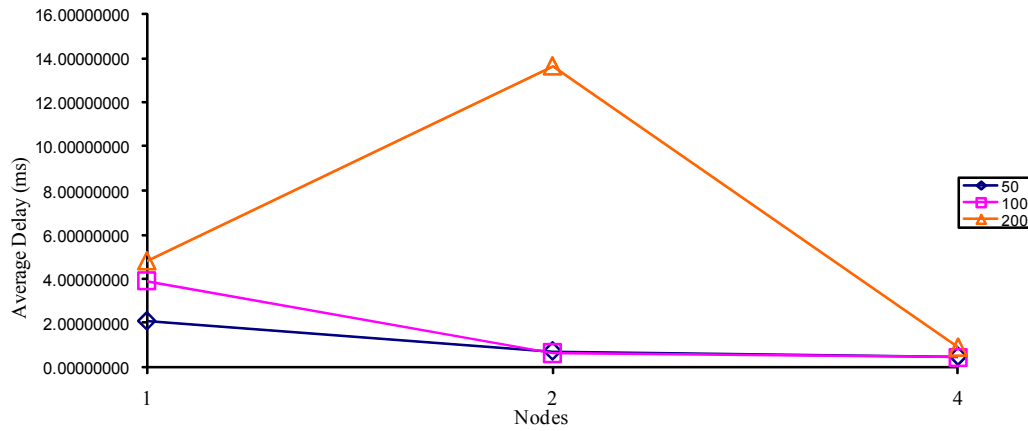

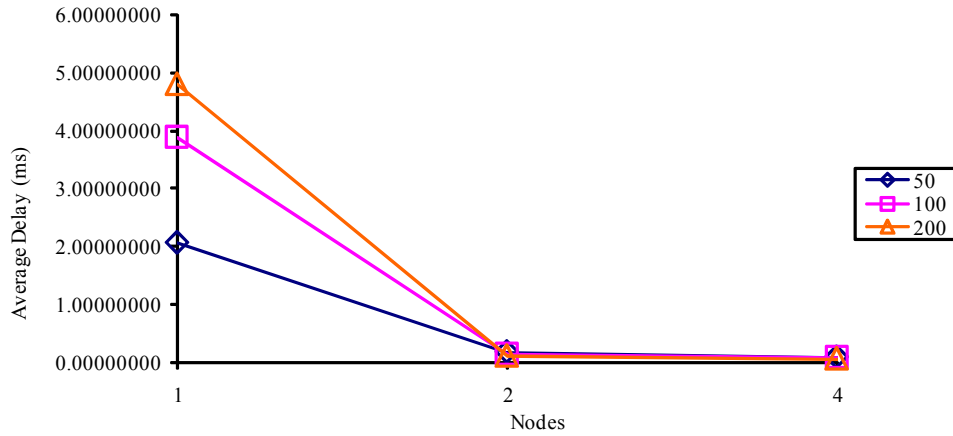Figure 3: Series of Concurrent Sessions. Random nodes vs. Average Delay

Figure 4: Series of Concurrent Sessions. Load Balanced vs. Average Delay

In all methods, delay decreased as the number of dbservers was increased. However, in the case of the sequential and largely the random method, delay actually increased when moving from one to two servers and showed improvement (measured decrease in average delay) when using four dbservers. It is clear that 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 demonstrated more efficiency loss due to calculation overhead at the 2 dbserver level and did not obtain the efficiency that either of the other two models had at higher load levels. The load balancing method produced the most dramatic improvement at all levels when compared to the other two models.
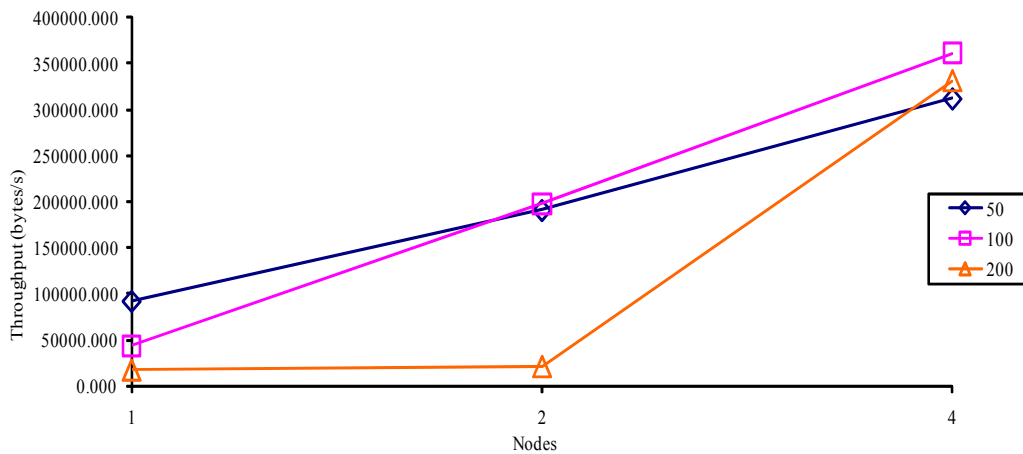

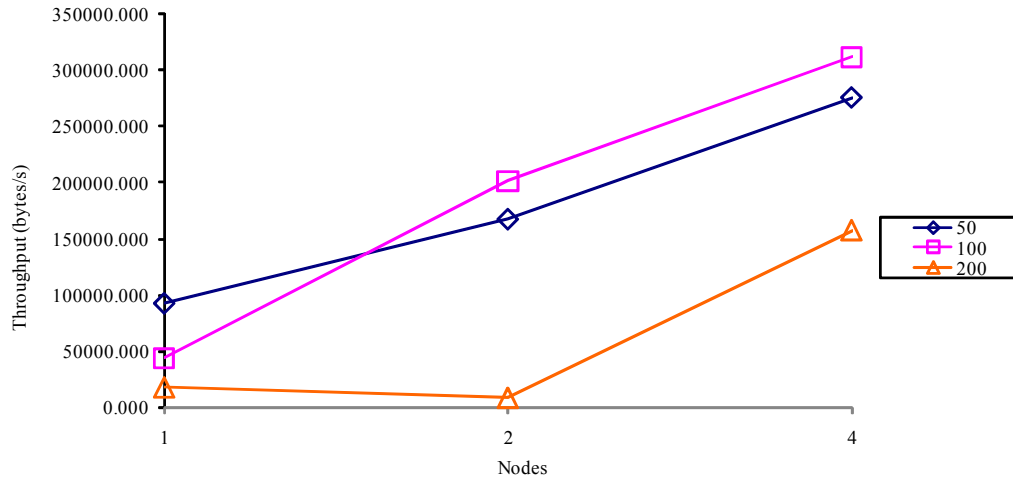Figure 5: Series of Concurrent Sessions. Sequential Nodes vs. Throughput

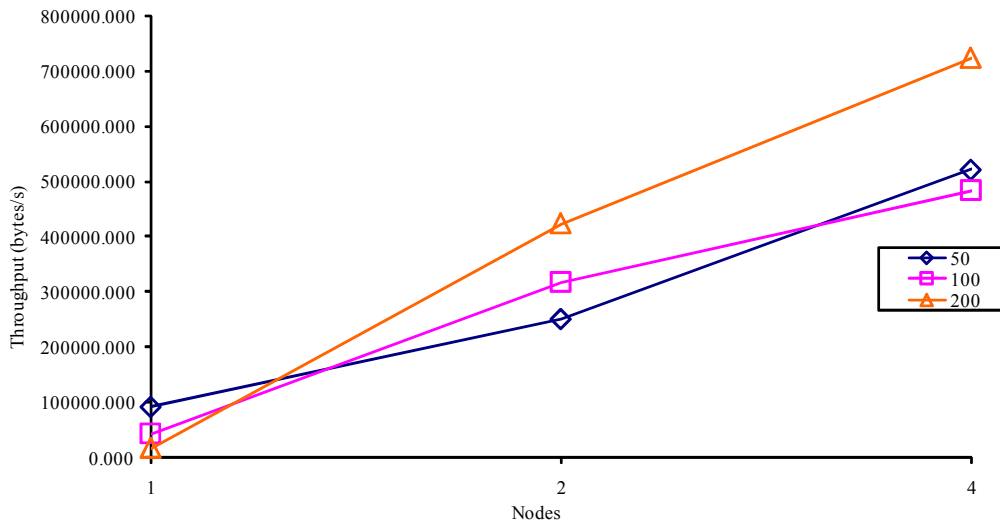Figure 6: Series of Concurrent Sessions. Random vs. Throughput

Figure 7: Series of Concurrent Sessions. Load Balanced Nodes vs. Throughput

With the decrease of delay (by adding additional dbservers), an increase in throughput was expected. The results of the throughput were not as dramatic as delay. By adding additional dbservers, there was a liner trend demonstrated by an increase in throughput as we moved from a sequential model to a load-balanced model. Using the random model, the data with two and four dbservers are closely related and nearly congruent. This congruency can be largely attributed to the calculation overhead effect of the random algorithm. Further testing would be required to predict when the throughput thresholds would be reached by adding more dbservers and contrasting the

sequential results with the load balanced results. The sequential method delivered a nonlinear trend, which depicts a higher return for each additional dbserver. However, it should be noted that the load-balanced throughput at four dbservers is 724,683 bytes per second whereas the sequential model at four nodes demonstrated a throughput of 330,987 bytes per second.
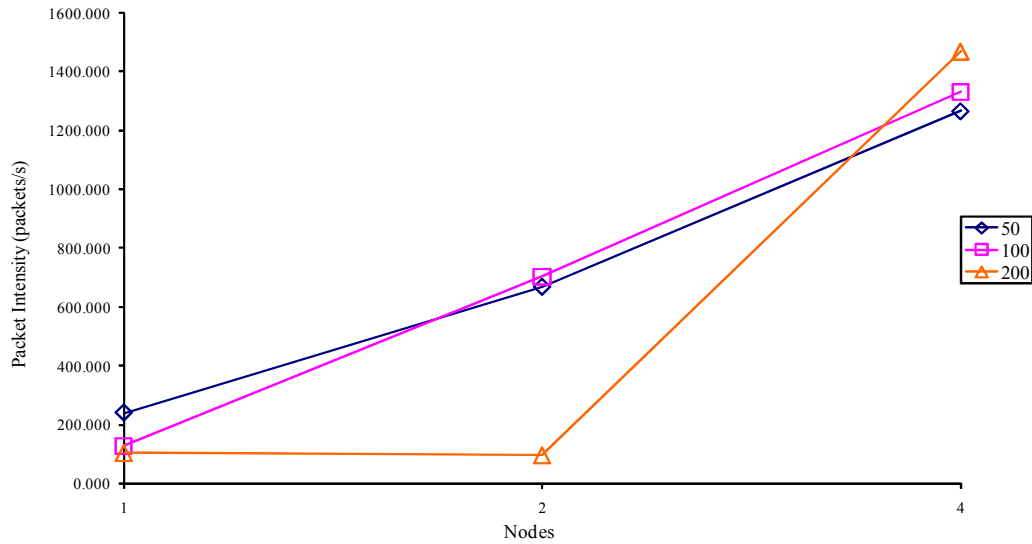

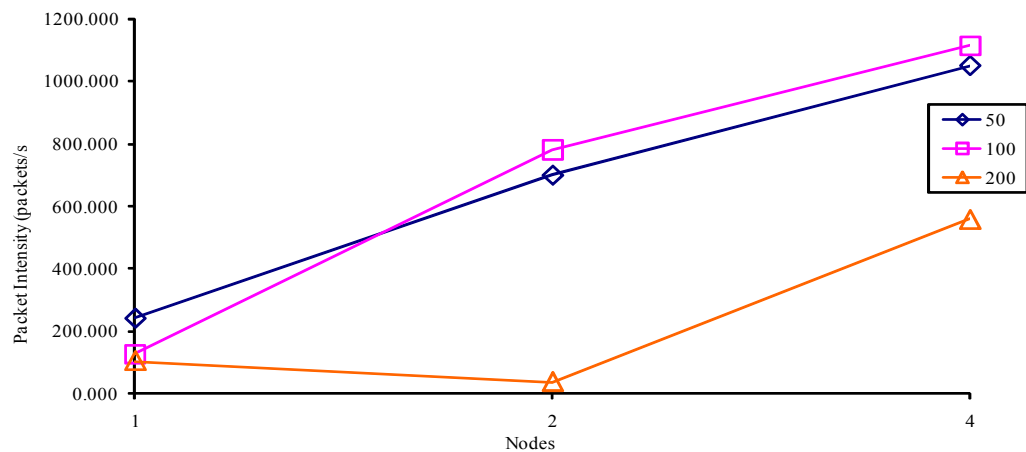Figure 8: Series of Concurrent Sessions. Sequential Nodes vs. Packet Intensity

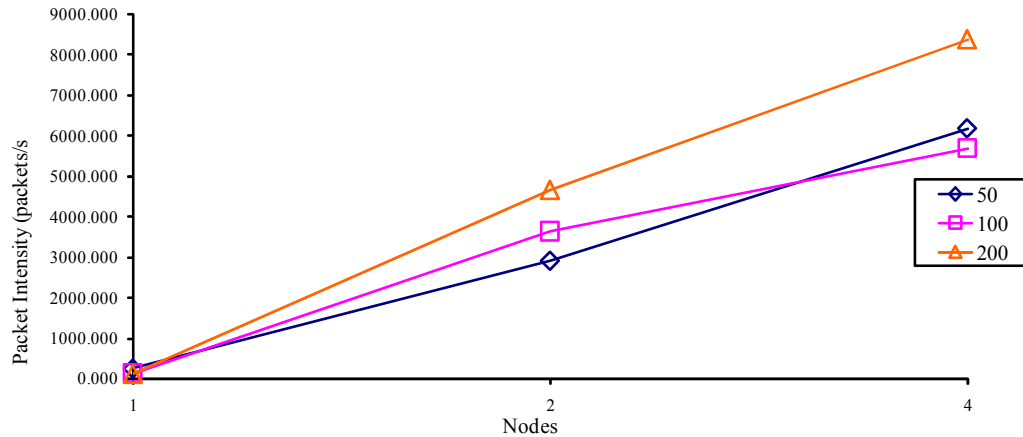Figure 9: Series of Concurrent Sessions. Random Nodes vs. Packet Intensity

Figure 10: Series of Concurrent Sessions. Load Balanced Nodes vs. Packet intensity

In all methods, packet intensity generally increased as the number of dbservers was increased. However, in the case of the random method, there was a clear indication that overhead is costly until a higher connection load is sustained. The load-balanced model was more efficient than the sequential model. The load balanced model peaked with four dbservers undergoing a load of 200 connections at 4,654 packets per second whereas the sequential model delivered at 1,470 packets per second. The random model results, with two and four dbservers, are closely related and nearly congruent. This congruency can be largely attributed to the calculation overhead effect of the random algorithm. Further testing would be required to predict when the packet intensity thresholds would be reached by adding more dbservers and contrasting the sequential results with the load balanced results.

DISCUSSION OF RESULTS

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 200 concurrent sessions on a single dbserver node, the delay increased from 2.07 to 4.81 milliseconds respec-

tively. Adding additional dbserver nodes, distributing the workload using the load balanced method among four nodes, and increasing the concurrent sessions from 50 to 200 decreased the delay from 2.07 to 0.06 milliseconds respectively. Therefore, hypothesis H1 must be rejected.

H2. The number of nodes a database is stored upon has no affect on response time 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 was a decrease in average delay from 2.07 to 0.08 milliseconds respectively. Setting the workload to 200 concurrent sessions, using the load balanced method, and moving from one to four dbserver nodes decreased the average delay from 4.86 to 0.06 milliseconds. 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 then switching from the load balanced to the sequential method, the average delay increased from 0.08 to 0.39 milliseconds and to 0.48 milliseconds when switching to the random method. When increasing the workload to 200 concurrent sessions, using four dbservers, and switching from the load balanced method to the sequential method, the average delay increased from 0.06 to 0.34 milliseconds respectively and to

0.89 milliseconds when switching to the random method. Therefore, hypothesis H3 must be rejected.

Performance Gain as Attributed to Adding Dbservers

Average Delay

The Sequential model demonstrated 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 from 2.07 to 0.39 milliseconds respectively. This effect was amplified when the load increased to 200 concurrent sessions, reducing the delay from 4.81 to 0.34 milliseconds respectively.

It is difficult to measure the scalability with the load balanced model as it offered an immediate delay reduction from 2.07 to 0.08 milliseconds even at the 50 session level when moving to four dbservers. The effect is relatively consistent when we increase the load to 200 concurrent sessions using the same load balanced model and moving to four dbservers. The result was a reduction in delay from the single dbserver model from 4.08 to 0.06 milliseconds.

The Random Model offered the least promising results when addressing packet delay. Compared to a decrease from 2.07 to 0.39 milliseconds with the sequential model and a decrease from 2.07 to 0.08 milliseconds with the load balanced model, the random model offered a mere decrease from 2.07 to 0.47 milliseconds 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 200 concurrent sessions decreased the average delay under the random model from 4.81 to 0.89 milliseconds respectively. This delay increased from 4.81 to 13.61 milliseconds when moving to two db servers using a load of 200 concurrent sessions.

Throughput

The sequential model offered 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 using the sequential method resulted in an increase of throughput from 92,758 to 312,233 bytes per second. Moving from a single dbserver under a load of 200 concurrent sessions to four dbservers, throughput increased from 17,879 to 330,987 bytes per second respectively.

The load-balanced model demonstrated the largest increase in throughput. At a load of 50 concurrent sessions, when moving from a single dbserver to four

dbservers, throughput increased from 92,758 to 522,527 bytes per second. A load of 200 concurrent sessions, moving from one dbserver to four dbservers using the load balanced method, resulted in a respective increase in throughput from 17,878 to 724,684 bytes per second, a much lower return then the 50 session load.

The random model offered an increase in throughput when moving from a single dbserver to four dbservers under a load of 200 concurrent sessions from 17,878 to 157,894 bytes per second respectively. However, when moving from one dbserver to two dbservers under the same load, throughput decreased to 8,529 bytes per second.

Packet Intensity

The sequential model peaked with an increase of packet intensity at the 200 concurrent sessions level, when moving from one dbserver to four dbservers, from 104.04 to 1470.37 packets per second. Under a load of 200 concurrent sessions and one dbserver, there was a respective decrease in packet intensity when moving to two dbservers from 104.03 to 96.25 packets per second under the sequential model.

The random model packet intensity improvement peaked with an increase of packet intensity with one dbserver at the 100 concurrent session level, when moving to four dbservers from 128.70 to 1,113.50 packets per second. With one dbserver, under a load of 200 concurrent sessions, there was a decrease in packet intensity when moving to two dbservers from 104.04 to 36.73 packets per second respectively. However, when moving from one dbserver to four dbservers under the same load, there was an increase from 104.04 to 558.57 packets per second respectively.

The load balanced model packet intensity peeked at the 200 concurrent session load moving from one dbserver to four dbservers. As a result, there was an increase from 104.04 to 8,375.96 packets per second, which was the highest recorded increase of any method. The load balanced method showed a respective depreciated increase in packet intensity from 4,653.96 to 8,375.96 packets per second under a load of 200 concurrent sessions, when moving from two dbservers to four dbservers.

Clearly there was an increase in performance in adding more dbservers in both the random and load balanced models. With higher session load, the performance increase was 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 occurred under the load balanced method with a load of 200 concurrent sessions when moving from one dbserver to four dbservers, delivering a reduction from 4.81 to 0.06 milliseconds. When testing the average delay, the load-balanced method consistently demonstrated substantial decreases 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, was attributed with the highest recorded delay with an increase from 4.81 to 13.61 milliseconds. The random method demonstrated a promising decrease in average delay under a load of 100 concurrent sessions by peaking with a reduction from 3.88 to 0.45 milliseconds when moving from one dbserver to four dbservers.

The sequential method initially demonstrated a decrease in average delay from 2.07 to 0.75 and to 0.39 milliseconds for a 50 concurrent connection model moving from one dbserver to two dbservers and then to four dbservers respectively. The sequential model demonstrated consistent decrease in average delay when moving from one dbserver to two dbservers and to four dbservers under any load from 50 to 200 concurrent connections. The only exception occurred with 200 concurrent sessions when moving from one dbserver to two dbservers, which resulted in an increase in average delay from 4.81 to 5.19 milliseconds.

Impact of Client Intensity on Design Methodology

Higher loads resulted in inconclusive over saturation levels 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, and servers would cease functioning when their active process count rose above 285 processes. Siege would 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 while waiting for acknowledgment from saturated servers of prior requests

sent for processing. There appeared to be no immediate saturation concerns with the main query distribution server.

Server recovery time was also a noteworthy concern. In most instances, it was not necessary to restart the dbservers 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 persist through each session for results. Occasionally, tests were completed before the 100,000 packet goal was reached. This indicated 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 outperformed the random and the sequential model. Recommended enhancements to the test environment would include the following two methods: (1) Doubling the dbservers from four to eight, running two web servers, each serving different applications, dynamically allocating dbservers to web server applications as needed, and then releasing the dbserver to the other allocation servers when load increases as web client demand increased; (2) increasing the number of dbservers to 32 running, the load balanced method, and testing each power of two using 2, 4, 8, 16, and 32 dbservers under a load of 400, 800, 1600 concurrent connections. This is all the while using four to eight Siege clients and distributing the concurrent sessions among the Siege clients evenly.

Recommendations for Further Research

Modify the Test Apparatus and / or Methodology

Pretest each of the servers to determine if they are performing within a tolerable level prior to each test. This could be a 50 concurrent session test executed directly against each dbserver concurrently or successively. Determine statistical variance between each of the loads. Determine the cause of peek performance for the load-balanced model to be at 200 concurrent sessions and then reducing this cause when testing with 400 concurrent sessions.

Demonstrate Scalability by Increasing Dbservers

It was clear, as the number of dbservers increased there was a corresponding increase in performance. Determine the required load to maximize justification for adding each additional dbserver. Ask: At what point would it be advisable to add additional web servers with segmented or dynamically allocated dbserver 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. Adding additional Siege clients and distributing the load evenly between the two clients would acquire additional client loads. Data can be collected on each Siege client using TCPDUMP. The data would be interpreted and a unique port address would be assigned to each session in order to enable session time and packet throughput analysis.

Additional database nodes typically resulted in increased performance. This was especially true when a load-balanced algorithm was 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 was more significant in prototyping the process than in obtaining scaling data

REFERENCES

[1] Amiri, A. "A Coordinated Planning Model for the Design of a Distributed Database System," *Information Sciences*, Volume 164, Numbers 1-4, 2004, pp. 229-245.

[2] Anthes, G. "Grids Extend Reach," *Computerworld*, Volume 37, Number 41, 2003, pp. 29-30.

[3] Cannataro, M., Talia, D. and Srimani, P. K. "Parallel Data Intensive Computing in Scientific and Commercial Applications," *Parallel Computing*, Volume 28, Number 5, 2002, pp. 673-704.

[4] Elnikety, S., Tracey, J., Nahum, E., and Zwaenepoel, W. "A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites," *Proceedings of the 13th International Conference on World Wide Web*, 2004, pp. 276-286.

[5] Guster, D., Safonov P., Hall C., and Sundheim R. "Using Simulation to Predict Performance Characteristics of Mirrored Hosts Used to Support WWW Applications," *Issues in Information Systems*, Volume 4, Number 2, 2003, pp. 479-485.

[6] Johnson, M. "Gridlock Reality," *Computerworld*, Volume 37, Number 41, 2003, pp. 24.

[7] Jutla, D., Bodorik, P. and Wang, Y. (1999). "Developing Internet E-Commerce Benchmarks," *Information Systems*, Volume 24, Number 6, 1999, pp. 475-493.

[8] Kanitkar, V. "Collaborative and Real-Time Transaction Processing Techniques in Client-Server Database Architectures," *Polytechnic University*, Volume 61, Number 04B, 2000, pp. 2036.

[9] Kanitkar, V. and Delis, A. "Distributed Query Processing on the Grid," *IEEE Transactions on Computers*, Volume 51, Number 3, 2002, pp.269-278.

[10] Li, W., Altintas, K. and Kantarcıolu, M. "On Demand Synchronization and Load Distribution for Database Grid-Based Web Applications," *Data and Knowledge Engineering*, Volume 51, Number 3, 2004, pp. 295-323.

[11] Peddemors, A. J. H. and Hertzberg, L. O. "A High Performance Distributed Database System for Enhanced Internet Services," *Future Generation Computer Systems*, Volume 15, Number 3, 1999, pp. 407-415.

[12] Rajamani, K. "Multi-Tier Caching of Dynamic Content for Database-Driven Web Sites", *Rice University*, Volume 63, Number 03B, 2002, pp. 1433.

[13] Roussopoulos, N., Economous, N. and Stamenasm, A. "A Testbed for Incremental Access Methods," *IEEE Transactions on Knowledge and Data Engineering*, Volume 5, Number 5, 1993, pp.762-774.

[14] Simhaa, R. and Majumdarb, A. "An Urn Model with Applications to Database Performance Evaluation," *Computers and Operations Research*, Volume 24, Number 4, 1997, pp. 289-300.

[15] Smith, J. "Distributed Query Processing on the Grid," *International Journal of High Performance Computing Applications*, Volume 17, Number 4, 2003, pp.353-367.

[16] Sobol, M. G., Kagan, A., and Shimura, H. "Performance Criteria for Relational Databases in Different Normal Forms," *Journal of Systems and Software*, Volume 34, Number 1, 1996, pp. 31-42.

[17] Townsand, M. and Tsai, J. "Oracle 9i New Features," *Oracle Corporation*, Redwood City, California, 2003.

[18] Wu, C. and Befford, G. G. "Improving the Flexibility for Replicated Data Management in Distributed Database Systems," *Computers and Industrial Engineering, 18th International Conference on Computers and Industrial Engineering*, 1996, pp. 901-905.

[19] Xiong, M., Ramamritham, K., Haritsa, J. R. and Stankovic, J. A. "MIRROR: A State-Conscious Concurrency Control Protocol for Replicated Real-Time Databases," *Information Systems*, Volume 27, Number 4, 2002, pp. 277-297.

AUTHOR BIOGRAPHIES

Christopher Brown is a database architect in the Information Technology Department at St. Cloud State University. He earned a Master's degree in Computer Networking at St. Cloud State University and has been active in research involving distributed database algorithms and performance. He is also involved in research related to data warehouse database design/performance.

Dennis Guster earned his Doctorate at the University of Missouri, St. Louis in 1981. He has 25+ years teaching experience in higher education and has been Professor in the Computer Science and Statistics Departments before joining the Business Computer Information Systems Department at St. Cloud State University in 2001. Dennis has served as a consultant and provided industry training to organizations such as Compaq, NASA, DISA, USAF, Motorola, and ATT. As Director of the Business Computing Research Laboratory at SCSU he has organized numerous sponsored research projects and has published in the areas of network design, network performance analysis, and computer network security. His research interests include mathematical modeling and simulation of computer information systems and networks. He is an author of more than 50 papers in journals and conference proceedings.

Sara Krzenski is a graduate student in the graduate computer network management program at St. Cloud State University. She is scheduled to complete that Master's degree Spring 2008 and has been active in research involving distributed database performance. She is also involved in research related to denial of service on the network level.