Using Thermal Sensors to Perform CPU Scheduling in a WWW Based Distributed Data Base Application

Dennis Guster, Christopher Brown, Charles Hall and Brittany Jansen
Business Computing Research Laboratory
St. Cloud State University
St. Cloud, Minnesota, 56304
dcguster@stcloudstate.edu

Abstract

Before applications became web applications (pre-http), the maximum number of potential users was limited to the size of the private network connected, 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 the data storage and retrieval methodology. This study looked at the potential of using scheduling based on the temperature of each CPU in a database cluster, with the assumption being the coolest processor was the least busy. Data was collected using the same workload computer test-bed and workload generation software used by [1]. Only this time the distribution method was based on thermal sensing. The results indicated that the delay was reduced by a factor of about 10 when compared to the most efficient algorithm used by [1]

1 Introduction

Before applications became web applications (pre-http), the maximum number of potential users was limited to the size of the private network connected, which was 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.

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."

Therefore, given this "millions-to-one" mechanical bottleneck, it is reasonable to investigate optimization by storing data on multiple disks and distributing them across multiple devices. This methodology suggests a reduction of data access time; as [5] shows, there was an improvement of throughput by ten percent and a decrease in workstation response time by a factor of 14 when distributed databases were employed. Further, it appears that there are three variables to consider while trying optimizing a distributed data within a WWW application.

The first variable acknowledged was workload intensity. Intensity increased the need to utilize a form of a distributed database. [9] determined that distributed databases offer significant performance advantages, if the system was large enough, in terms of users. [9] 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 of nodes reduces access time. However, [8] 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. In theory, 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.

There is much agreement that optimizing the allocation method is critical to the success of any distributed processing endeavor [3], [10], and [16]. Although various operating system or middleware level algorithms have been employed successfully in all cases, there is the problem of delay caused by the processing load and internal overhead of that database host [7], [15], and [1]. In other words, a distributed database host must service requests and keep track of its utilization and inform the distribution host of its availability. Logically, this a small task that can be given a high priority but it must still contend for an interrupt and fight for a time slice with the database application and other

operating system level tasks. Given that the typical distribution model uses a single point of control to allocate the requests among the distributed nodes, any method that bears promise in reducing the communication delay back to the distribution control host merits investigation [4].

The idea of obtaining the CPU usage directly from some source other than the operating system is not new. Using power consumption as a means of determining CPU utilization has been used for some time [17] and [11]. Monitoring power consumption at the power supply level is easily accomplished, but monitoring it effectively at the CPU level would quite possibly require a specially designed motherboard. Therefore, if the goal is to test a sensing device using standard off the shelve computers, the concept of monitoring heat at the CPU level could be easily implemented by simply putting a thermal sensor in the heat sink of the desired CPU(s). Although little has been done directly in using thermal sensors to schedule database inquiries in a distributed environment, thermal management has been used to try to optimize CPU efficiency [13], [6], and [14]. Further research has been undertaken to confirm that CPU temperature is related to CPU utilization [12]. Therefore, based on the success of this previous research, it appears that scheduling database inquiries in a distributed environment by thermal activity merits investigation.

As previously stated, operating system based algorithms have been successfully employed, and the efficiency they produce can serve as a benchmark from which to evaluate the effectiveness of the thermal methodology. In fact, in some cases, the delay back to the client after a database lookup has been reduced dramatically. Specifically, [1] found that with a load-balancing algorithm end user delay could be reduced from 4.8 to .05 milliseconds. This is quite an improvement and certainly very effective at the load offered (200 www session across 4 processing nodes). However, in applications with greater loads and database complexity, there still is a need for further optimization of the distribution methodology. To ascertain the potential of using thermal scheduling, data was collected using the same workload computer test-bed and workload generation software used by [1], only this time the distribution method was based on thermal sensing. The logic was that the coolest processor was assumed to be the least busy and, therefore, that was where the next inquiry was directed.

2 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?

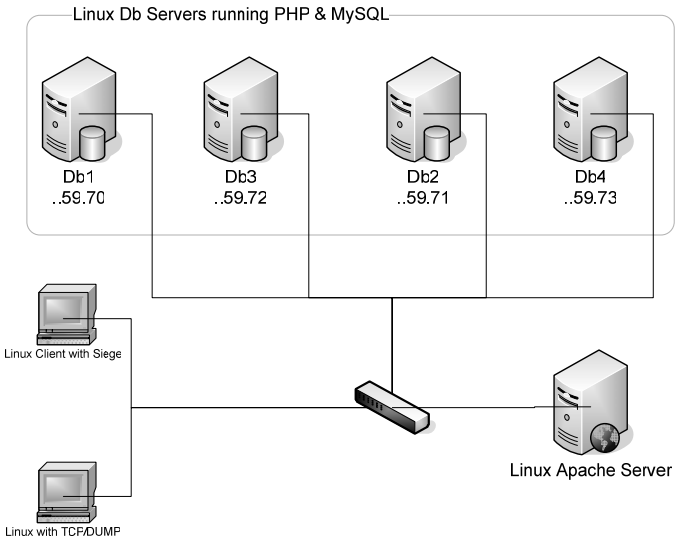
3. How does the thermal scheduling algorithm compare to the most effective algorithm (load balancing) used by [1] when assigning a given query to a specific database node?

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:



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 most effective software based URL from [1] was the load balanced (http://199.17.59.65/page/?function=load). The apache server would then redirect the output based on the predefined algorithm (either thermal or load balanced) 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 is able to generate web traffic streams of varying intensity. For the experiments run herein, the traffic of eight consecutive groups of 50 and 400 clients were generated in two 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, which consisted 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 database servers (if multiple db servers were used) would receive any given request. In the thermal method, a temperature sensor was placed in the heat sink of each database processor node and, in turn, plugged into a control panel. The control panel then fed the temperature data into the distribution server, which simply assigned the next incoming inquiry to the node with the lowest temperature. 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 two Tables.

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
Thermal	8	2	50	0.00517468	911275.131	8669.450
Thermal	8	4	50	0.00466387	982233.329	11266.901
Load Balanced	8	2	50	0.17195826	252105.315	2907.682
Load Balanced	8	4	50	0.08090560	522526.965	6180.042

The data 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, which suggests 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 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 400 Concurrent Sessions.

Query Distribution Type	Sequential Iterations	Server Nodes	Clients	Average Delay (ms)	Throughput (bytes/s)	Packet Intensity (packets/s)
N/A	8	1	400	4.20071933	23200.616	119.04
Thermal	8	2	400	0.00687468	1103245.131	10254.45
Thermal	8	4	400	0.00486387	1217343.329	11266.90
Load Balanced	8	2	400	0.43814796	99091.412	1141.16
Load Balanced	8	4	400	0.09072802	474901.359	5510.97

In all cases, delay decreased as the number of dbservers was increased. However, the methods were affected differently as the number of concurrent sessions was increased from 50 to 400. The delay for the single server and the load balanced algorithms approximately doubled whereas only slight increases appeared in the thermal method data. Both the thermal and the load balanced methods greatly reduced delay, but the thermal method further reduced delay, beyond the load-balanced method, typically by a factor of 20 to 30. A corresponding increase in packet intensity and bytes transferred was also observed in favor of the thermal method. In regard to bytes transferred, the increase was from 4 to 10 times greater at the 400 concurrent session levels.

3 Discussion, Conclusions and Recommendations

In all cases the three null hypotheses tested through experimentation can be rejected.

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. In all cases the corresponding delay increased when the number of concurrent clients was increased from 50 to 400.

H2. The number of nodes a database was stored upon had no affect on response time to the originating client. Both distributed algorithms improved the response time to the client.

H3. The algorithm used to distribute requests to a given distributed database node had no affect on the delay to the originating client. The thermal method definitely exhibited less delay than the load-balanced method.

Given the results above, it appears from the data collected that the thermal methodology can, in fact, offer a higher degree of scheduling efficiency when compared to [1] load balancing algorithm. In fact, the parameters for both delay and throughput were improved

upon. This is an encouraging finding, but it must be replicated several times before it can be totally accepted. Setting up the test-bed and collecting this data was difficult and time consuming. The sophistication by which the temperature probes were mounted coupled with lack of control of room temperature and humidity, which can detract from the potential reliability of this experiment. Further, the load-balanced data was obtained first and then test-bed was dismantled. Although every effort was made to recreate the test-bed in every detail for the collection of the thermal data, the reliability would have been better if exactly the same test-bed was used in each case.

In terms of scaling, it would be interesting to see how each method fares as additional database nodes are added to the experiment. Further, although the workload level of 400 concurrent sessions is quite intense, it would be worthwhile to increase that value and determine whether or not the current observed performance ratios hold true. Also, it would useful to replicate these experiments with other types of computer hardware and networks of higher speed.

Last, besides offering a higher degree of scheduling efficiency, the thermal solution would reduce the workload on the database node operating system; however, the thermal solution would create additional hardware cost and complexity. More than likely, given the added efficiency, the cost could be easily justified, but the complexity could be problematic, from a reliability perspective as well as the cost of trained personnel to support it. Further, if sophisticated climate control was required to ensure the accuracy of the sensors, then that cost may prove to be a limiting factor as well. Although the preliminary results obtained herein are encouraging, much work is still needed to ascertain the reliability, practicality and efficiency of the thermal method.

References

[1] C. Brown, Analysis of Three Database Distributed Algorithms, St. Cloud: St. Cloud State University, 2005.

[2] Chieuh, H., J. Draper and J. Choma, "A Programmable Thermal Management Interface Circuit for Power PC Systems," Micro Electronics Journal, no. 32, pp. 875-881, 2001.

[3] A. Dogan and F. Osgunger, "Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous," IEEE Transaction on Parallel and Distributed Systems, vol. 13, no. 3, pp. 308-323, 2006.

[4] F. Dong and S. Akl, Scheduling Algorithms for Grid Computing: State of the Art and Open Problems Technical Report, Kingston: Queen's University, 2006-504.

[5] S. Elnikety, J. Tracey, E. Nahum and W. Zwaenepoel, "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, 2004.

[6] M. Elnozahy, M. Kistler and R. Rajamony, "Energy-Efficient Server Clusters," Proceedings of the Second Workshop on Power Aware Computing Systems, 2002.

[7] D. Guster, Safonov P., Hall C. and R. Sundheim, "Using Simulation to Predict Performance Characteristics of Mirrored Hosts Used to Support WWW Applications," Issues in information systems, vol. 4, no. 2, 2003.

[8] D. Guster, R. Sultanov, M. Nordby and R. Sundheim, "Using Distributed Processing to Enhance Performance Characteristics of Hosts Used to Support WWW Applications," International Journal of Business Research, vol. 6, no. 2, pp. 67-71, 2006.

[9] V. Kanitkar and A. Delis, "Distributed Query Processing on the Grid," IEEE Transactions on Computers, vol. 51, no. 3, pp. 269-278, 2002.

[10] S. Kartik and C.S.R. Murthy, "Task Allocoation Algorithms for Maximizing Reliability of Distributed Computing Systems," IEEE Transactions on Computers, vol. 46, pp. 719-724, 2002.

[11] J. Luo and N. K. Jha, "Power-Conscious Joint Scheduling of Periodic Task Graphs and Periodic Tasks in Distributed Real-Time Embedded Systems," Proc. International Conference Computer-Aided Design, pp. 357-364, 2001.

[12] M. Naghedolfeizi, S. Arora and S. Garcia, "Performance Analysis of a High-End CPU Under a Heavy Computational Load and Varying RAM Amount Using Thermal Imaging Techniques," Autotestcon IEEE, pp. 574-577, 2005.

[13] E. Pinheiro, R. Bianchini, E.V. Carrera and T. Heath, “Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems,” Proceedings of the Workshop on Compilers and Operating Systems for Low Power, 2001.

[14] K. Rajamani and C. Lefurgy, “On Evaluating Request Distribution Schemes for Saving Energy in Server Clusters,” Proc. IEEE International Symposium on Performance Analysis of Systems and Software, 2003.

[15] P. Safonov, D. Guster, R. Sultanov and N. Krivulin, “Is Deployment of Distributed Processing for Business Applications a Sound Management Decision: Performance Analysis and Simulation,” Journal of Information Technology Management vol. 16, no. 3, 2005.

[16] S. Shatz,, J. Wang and M. Goto, “Task Allocation for Maximizing Reliability of Distributed Computer Systems,” IEEE Trans. Computers, vol. 41, no. 9, pp. 1, 156-168, 1992.

[17] C. Woodside and G. G. Monforton, “Fast Allocation of Processes in Distributed and Parallel Systems,” IEEE Trans. Parallel & Distributed System, vol. 4, no. 2, pp. 164-174, 1993.