7-2008

Resource-Constrained Load Balancing Controller for a Parallel Database

Z. Tang

J. Douglas Birdwell

J. Chiasson

Chaouki T. Abdallah

Majeed M. Hayat

IEEE Transactions on Control Systems Technology, Vol. 16, No. 4 (July 2008): 834-840. DOI.

Resource-Constrained Load Balancing Controller for a Parallel Database

Z. Tang
Department of R&D Software, Beckman Coulter Inc., Indianapolis, IN
J. D. Birdwell
Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN
J. Chiasson
Department of Electrical and Computer Engineering, Boise State University, Boise, ID
C. T. Abdallah
Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM
M. Hayat
Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM

Abstract:

This brief documents experimental results using a deterministic dynamic nonlinear system for load balancing, previously reported by Tang et al. in a cluster of computer nodes used for parallel computations in the presence of time delays and resource constraints. While previous publications by the authors have provided theoretical analysis of this load-balancing strategy using an idealized model, and have documented experiments using a simulated database, experimental results using a complete database for DNA profiles are documented here. Evaluation of the proposed load-balancing strategy using an actual database was critical because of several characteristics of the database that cannot be accurately captured using either a simulation model or database,

including the variation in times required for the database to perform search operations, the time-varying and task-dependent computational load the database imposes upon each node of the parallel computer, and the time-varying network traffic imposed by both the communication of database search requests and results, mixed with the traffic generated by the load-balancing strategy. Although the load-balancing strategy can be represented in a relatively straightforward manner using mathematics, its implementation is by necessity an approximation to its mathematical description. The reported experimental results serve to validate the superiority of using the controller based on the anticipated work loads to a controller based on local work loads, which has been predicted with experiments using a simulated database and documented in prior publications. The experiments demonstrate the efficacy of the load-balancing strategy using an anticipated pattern of work loads and provide support for scalability of the approach.

SECTION I. Introduction

Parallel computing, which uses multiple interconnected computational elements to solve a single problem, can be applied to large-scale parallel databases. DNA databases used in forensic applications have been growing rapidly, and are likely to increase to an eventual size of 10^8 profiles. The eventual size and the performance expectation for these databases necessitate the development of parallel databases. New methods developed by Wang and Birdwell [3]–[4][5] lead naturally to a parallel decomposition of the DNA database search problem while providing orders of magnitude improvements in performance over current software. Distributing the load evenly on parallel architectures is a key attribute of an efficient implementation.

Distribution of computational load across available resources is referred to as the *load-balancing* problem in the literature [6], [7]. Methods to balance computational load may be either deterministic, depending on a predefined strategy [8]–[9][10][11][12], or stochastic, distributing load in a random fashion [13]–[14][15][16][17]. Static load balancing was modelled as noncooperative game recently [18]. Iterative load-balancing methods are addressed in [8] and [19]–[20][21][22]. A comparison of several balancing methods is provided by Willebeek-LeMair and Reeves [23]. Approaches based on queuing theory appear in [24] and [25]. Control theory has shown promise in information technology applications, including web services [26] and databases [27], [28].

The work described in this brief is based upon a generalization of queue length of tasks to expected waiting time, which accounts for differences among computational elements (CEs), and aggregates the behavior of each queue. Previous results by the authors study the effects of delays in the exchange of information among CEs and the performance of a load-balancing strategy [29]–[30][31][32]. However, the model used to obtain this result did not account for processor resource constraints—the fact that load distribution and task processing cannot be carried out simultaneously.

Our prior work [1] presents a mathematical model that captures processor resource constraints in a load-balancing system. This open-loop model was shown to be self-consistent and (Lyapunov) stable. Initial results showing an extension to closed-loop control for a resource-constrained load balancing are presented in [2]. A control law has been proposed by the authors that uses estimates of *anticipated* workloads, which includes not only local estimates of the queue sizes at the other nodes, but also estimates of the number of tasks in transit to each node. A discrete event simulation of the closed-loop model using OPNET Modeler [33] is presented in [34], demonstrating good agreement between the nonlinear time-delay model and the simple experimental implementation.

While the author's prior publications document experimental work using a time delay to emulate a database search, this brief documents the closed-loop model and results using implementations of DNA profile databases containing several million profiles. Experimentation using a DNA profile database rather than an emulation of

database activity using a time delay model is important for several reasons: 1) The times required to complete searches for matching DNA profiles in a database are variable, and the distribution of search times has a long tail; 2) the DNA database requires significant processor and memory resources which compete with any load-balancing strategy; and 3) engineering trade-offs exist in the implementation of both the DNA database and the load-balancing strategy that influence performance. For these reasons, it is important to establish that the load-balancing performance predicted by simulations and implementations that emulate the DNA database using time delays can be achieved for an operational database. Experiments on a parallel DNA database are presented in this brief that document both the implementation strategy and the efficacy of the load-balancing strategy using anticipated work loads.

This brief is organized as follows. Section II briefly describes a nonlinear time-delay model of a load-balancing algorithm for a computer network that incorporates time delays and the model parameter values are given here. Section III documents the implementation of a parallel database and its load-balancing method. Section IVpresents experiments on the parallel DNA database using the closed-loop controller based on anticipated work load. Section V concludes this work.

SECTION II. Mathematical Model

The model used in this brief captures the effect of the delays in load-balancing techniques as well as the processor constraints so that system theoretic methods can be used for analysis. The mathematical model of the task load dynamics at a given computing node for load balancing is given by [1] and [2].

The model has been shown in [1] to be self-consistent and be (Lyapunov) stable, but asymptotic stability must be insured by the choice of the feedback law. A closed-loop controller is implemented that uses not only the local queue size, but also an estimate of the number of tasks in transit to the queue from other nodes [2].

The load transfer portions, i.e., p_{ij}, can be specified using the anticipated waiting time z_j of the other nodes as follows:

$$p_{ij} = \frac{\text{sat}(z_{j\_\text{avg}} - z_i(t-\tau_{ji}))}{\sum_{i\epsilon i\neq j} \text{sat}(z_{j\_\text{avg}} - z_i(t-\tau_{ji}))} \quad (1)$$

where sat$(\cdot)$ is defined as $\text{sat}(x) = \begin{cases} x \text{if} x \geq 0 \\ 0 \text{if} x < 0 \end{cases}$. The quantity sat $(z_{j\_\text{avg}} - z_i(t-\tau_{ji}))$ is a measure by node j as to how much node i is *below* node j's estimate of the network average (anticipated) waiting time. Node j then portions out its tasks among the other nodes according to the amounts they are below its estimate of the network average waiting time.

Determination of the model parameters through computation experiments on a parallel machine (discussed below) was given in [2]. Details about parallel computation experiments were discussed in [35]. On each node i, an average processing time to service a search task on a parallel DNA database of $t_{p_i} = 400\mu$s was used. The expected wait time $x_i(t) = q_i(t)t_{p_i}$ depends on the size of the task queue. We note that this is a significant approximation, as the time required to process each task depends upon the characteristics of the data associated with each search. The rate of generation of waiting time is either $\lambda_i = 1$ or $\lambda_j = 0$, for $j \neq i$ at time t. Because the model's state for each node is expected waiting time, the rate of reduction in waiting time due to processing of tasks is $\mu_i = 1$. The network bandwidth limitation was experimentally determined in [2] for an average packet size of 4 kB and is captured in the model by the data transfer rate limit $U_{m0} = 1.25 \times 10^5 \times t_{p_i}$. The network communication delay experienced by each transfer is random, with an average delay of $\tau_{ij} = 200\mu$s. The task transfer delays among nodes h_{ij} depend on the numbers of tasks to be transferred and are also random [2]. For chunked data transfers of tested search tasks, these delays h_{ij} vary from 400 μs to 4 ms.

SECTION III. Experiment Design

A parallel computer has been built to evaluate load-balancing strategies on parallel databases. A root node (search server) communicates with k groups of networked computers. Each of these groups is composed of n nodes holding identical copies of a portion of the database. Load-balancing actions are performed among the nodes inside each group. It is anticipated that the implementation will scale by multiples of eight computers, and the upper limit of this design appears to be on the order of 10^8 DNA profiles due to current memory limitations of the systems and available network bandwidth.

A database search engine is executed on each node, except the root node, of the parallel machine. The parallel database is implemented as a set of queues with associated search engine threads, typically assigned one per node of the parallel machine. The search engine accesses tree-structured indices to locate database records that match search requests, as described in [5]. Due to the structure of the search process, search requests can be formulated for any target profile and associated with any node of the index tree. These search requests are created not only by the database clients; in one implementation of the database, the search process itself can also create search requests as the index tree is descended by any search thread. Search requests that await processing may be placed in any queue associated with a search engine containing the same data, and the contents of these queues may be moved arbitrarily among the nodes of a group to achieve a balance of the load.

A search server communicates with clients, accepts incoming requests, and returns results. Clients do not interact directly with parallel nodes, but instead see a single database with rapid search capability. Fig. 1 shows a multithreaded search server using threads [36], PVM [37], and object serializations. The multithreaded search server starts a listening thread (LThread) which listens for connection requests, and manages a pool of service threads (SThread) that service these connections. Requests are distributed and results are gathered using a communication thread (CommThread), which communicates with the search engines via PVM and serialization. A logging thread (LogThread) records events, such as connection time and request information.

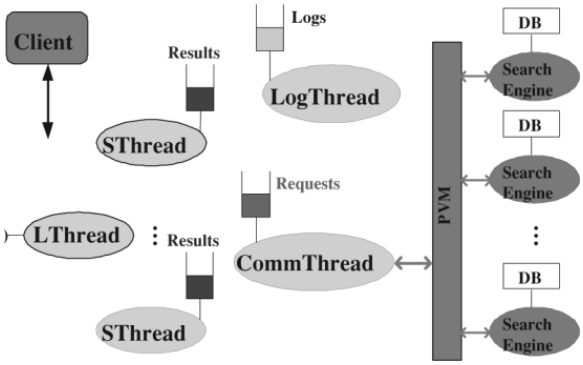

Fig. 1. Schematic diagram of a multithreaded search server using PVM and object serializations.

CommThread distributes requests to and gathers results from all search engines on the parallel computer. Each search engine on the parallel computer enrolls into PVM after building an initial DNA decision tree (indexing previously stored DNA profiles). After searching through the decision tree for matches (or other actions, depending on requests from CommThread), the search engine sends the result in a serialized buffer using PVM to the CommThread of the search server. The SThread returns the results to the client via a dedicated socket connection.

An experiment using 2000 random searches is conducted to test the performance of the multithreaded search server. The DNA database contains 48 million (4.8×10^7) profiles stored across 24 nodes of the parallel computer, with a two-million-profile portion stored on each node. Fig. 2 shows the measured search times and

transaction times for 2000 random searches. In this figure, the top part shows the average search time versus the node index (node01 through node24). The search time refers to the time to service a search request by searching the decision tree to find an exact match. Node14 has the largest average search time at about 340 μs, while node07 has the smallest average search time at about 270 μs. This variation is a function of random effects within each node and variations in the database's index structure. The average search time across the parallel machine is 318.3 μs. The bottom part shows the average transaction time for 2000 random searches as a function of the node index. The transaction time is the round-trip time required to send a search request from a client and return results. The average transaction time across the parallel machine is 3.7 ms.

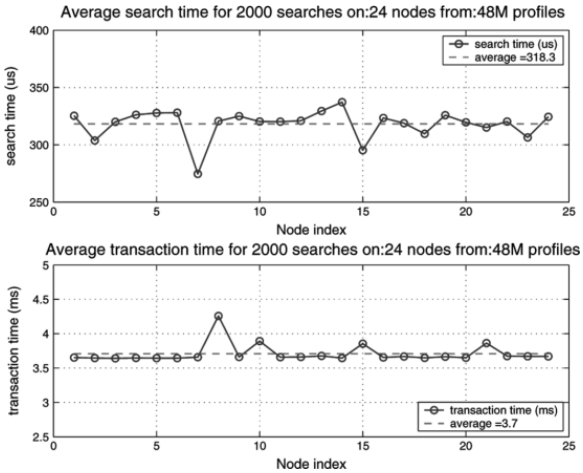

Fig. 2. Time measured for 2000 random searches on the multithreaded search server with 48 million profiles on 24 nodes. (Top) Average search time. (Bottom) Average transaction time.

SECTION IV. Experimental Results

Experimental results for parallel searches with load balancing integrated with a parallel DNA database are presented in this section. The first experiments are conducted to evaluate the load-balancing algorithm on the parallel database using a nonempty initial task distribution and no arriving tasks. The second experiments show results with randomly generated task arrivals to the search engines and compare this to searching the parallel database with load balancing disabled. The third experiments show results with load balancing on a larger network consisting of six nodes. These experimental results demonstrate the efficacy of the load-balancing strategy using anticipated waiting times on a parallel DNA database.

A. Queues of Initial Tasks

In this experiment, the performance of load balancing for a three-node group with an initial unbalanced condition and no new arrivals is evaluated. Each of the nodes (labeled node1, node2, and node3) runs a search engine, and the three DNA databases are identical. The initial conditions used for the task queues (q_1, q_2, q_3) are (0, 0, 200). On each node, a load-balancing thread broadcasts its queue size (when the queue's size changes) to the other nodes in the network, and also receives information on their queues' sizes. After loading the initial 200 search requests (tasks), node3 calculates its estimate of network average load as $q_{3\_\mathrm{avg}} = (200 + 0 + 0)/3 \approx 67$ (with 0 from both node1 and node2), and its workload relative to the network average as $q_{3\_\mathrm{diff}} = 200 - 67 = 133$. Next, node3 calculates the portions of search requests (tasks) to be transferred according to (1), and broadcasts the number of search requests to be transferred to each of the other nodes, which include the (anticipated) numbers of tasks being sent to node1 and node2 (66 each). Fig. 3(a) shows the local workloads, average estimates, and tracking differences computed by node3. ode1 receives the values broadcast from node3, and updates its estimate of average (anticipated) workload as $q_{1\_\mathrm{avg}} = ((200 - 132) + 66 + 66)/3 \approx 67$. Node1 then calculates its workload relative to the network

average as $q_{1\_\mathrm{diff}} = q_1 - q_{1\_\mathrm{avg}} = -67$, and so $\mathrm{sat}(q_1 - q_{1\_\mathrm{avg}}) = 0$. In this manner, node1 has a more up to date estimate of the (anticipated) workload at node2, and unnecessary transfers are avoided. Upon receiving the 66 requests transferred from node3, node1 inserts the search requests to its queue and continues processing them. The local workloads and average estimates on node1 are shown
in Fig. 3(b). Fig. 3(a) and (b) shows that the load-balancing algorithm, which uses a closed-loop controller based on anticipated work loads, works quite well in this situation.

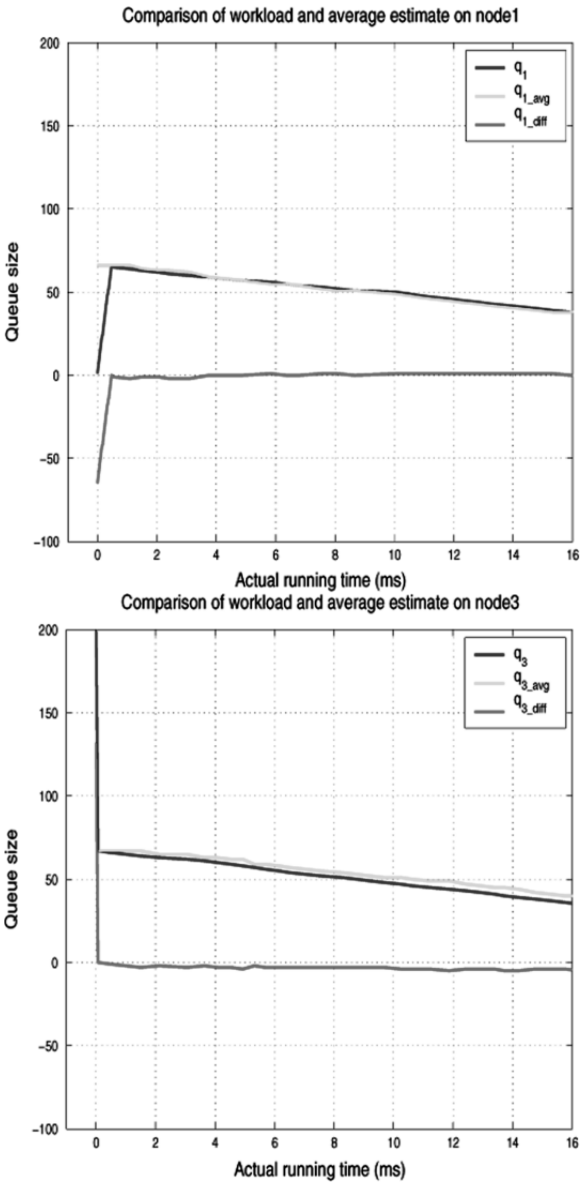

Fig. 3. Workloads and average estimates on (a) node3 and (b) on node1.

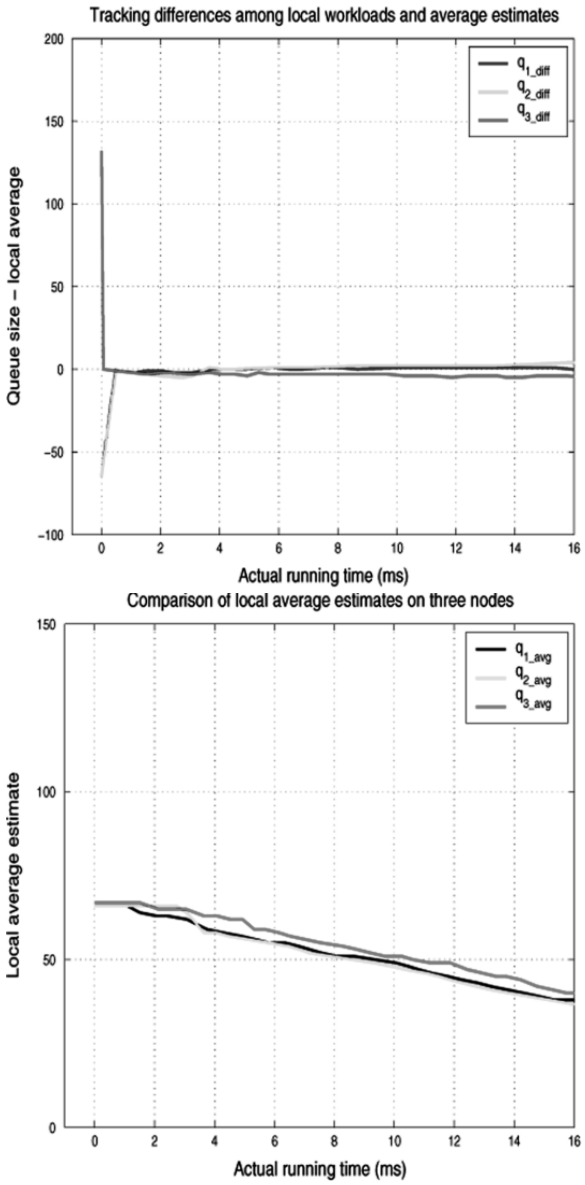

Fig. 4. Average estimates $q_{1\_\mathrm{avg}}, q_{2\_\mathrm{avg}}, q_{3\_\mathrm{avg}}$ (a) and tracking differences $q_{1\_\mathrm{diff}}, q_{2\_\mathrm{diff}}, q_{3\_\mathrm{diff}}$ (b) on three nodes with initial tasks.

A comparison of the average estimates on each of the three nodes as a function of time is shown in Fig. 4(a). The average estimates on the three nodes are similar. The deviations are due to variations in the times for different search requests and random delays in network traffic, and the small positive threshold (10) is used to prevent chattering. Fig. 4(b) compares the tracking differences between local workloads and average estimates on the three nodes. The local workloads track the average estimates very well, and the system settles quickly. Note that the database searches are running in parallel and asynchronously on each search engine node. Only the changes of queue states on each node are logged. Task processing (insertions and removals of tasks) on node1, as well as node2, starts after receiving the tasks transferred from node3.

B. Randomly Generated Requests

In this experiment, the tasks are collected into blocks of 100 by the search server. To evaluate the queuing of tasks on the search engine nodes and subsequent load balancing and task processing, these blocks of 100 tasks are sent to randomly selected nodes in the group. This experiment also uses a group of three nodes, and a total of 1000 search requests (tasks) are generated by a client program by randomly selecting DNA profiles to be used

as targets for a search. Every 5 ms the search server randomly selects a search engine node and sends a block of 100 tasks. This rate exceeds the rate at which each queue can receive tasks and insert them into a local queue; thus, the tasks are received over a period of about 140 ms. While the search engine thread on each node processes requests in its local queue, each node exchanges queue information with the other nodes and redistributes the tasks depending on the relative workload by running the load-balancing thread. Fig. 5 shows the workload, average estimate, and tracking difference on node2. The large upward transitions are caused by task arrivals (blocks of 100 tasks) from the client, while small upward transitions are caused by receipt of transferred search requests and queue insertions. The downward transitions are caused by removal of tasks from a queue for service or for transfer in blocks (which may be of different sizes, as determined by the load-balancing implementation) to other nodes.

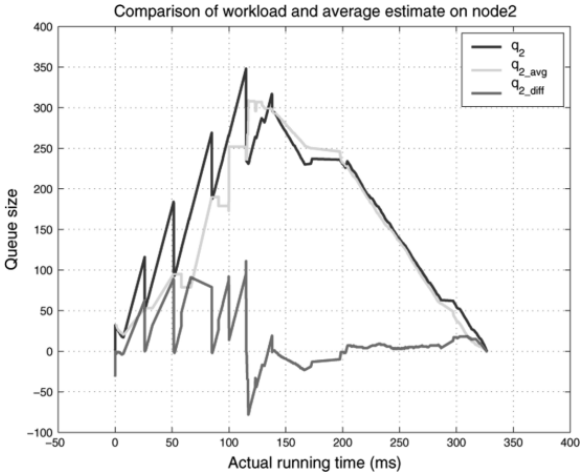

Fig. 5. Workloads, average estimates, and tracking differences on node2 with random requests. Blue curve: workload q_2, green: average estimate q_{2_avg}, and red: tracking difference q_{2_diff}.

Fig. 6 shows a comparison of average estimates computed on three nodes with randomly generated requests. When a new block of search requests arrives, the receiving node updates its average, which creates a step transient that is visible in the figure. The load-balancing algorithm then evens out the tasks and brings the average estimates together. For $t > 200$ ms, no new search requests arrive. The system settles to a balanced state, and the average estimates on three nodes closely follow each other.

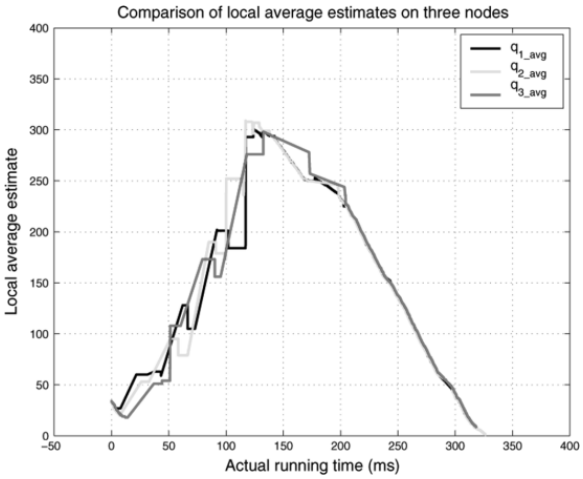

Fig. 6. Comparison of average estimates on three nodes with randomly generated requests. The black, cyan, and pink curve stands for $q_{1\_\mathrm{avg}}, q_{2\_\mathrm{avg}}, q_{3\_\mathrm{avg}}$ on node1, node2, and node3 respectively.

Fig. 7 shows the responses for 1000 tasks arriving in ten blocks on three nodes when the load-balancing thread is disabled. The search server randomly selects a search engine node for each transfer. Note the upward and downward transitions are caused by task arrivals and removals. The near-zero slopes correspond to computing time used for other non-task-processing jobs, such as data logging, network communication, and the operating system as well. From Fig. 7 the queues on the nodes are not balanced. This leads to different completion times and a larger completion time for the group.

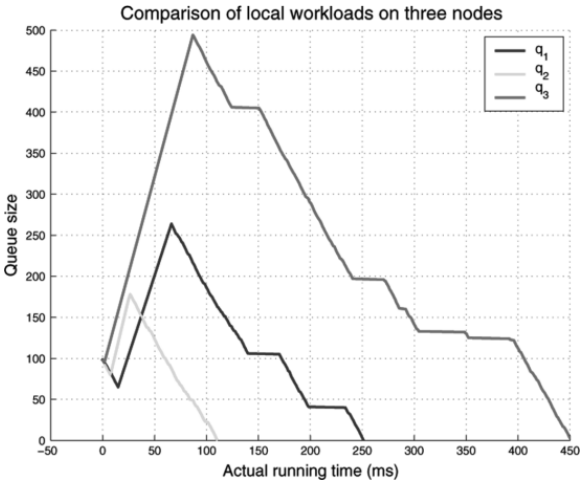

Fig. 7. Responses of queue sizes on three nodes without load balancing.

C. Scaling to Larger Networks

This set of experiments shows results for parallel searches with load balancing on a larger network of multiple nodes $(n = 6)$. In this experiment, a total of 2000 tasks are randomly generated by a client program in 20 blocks of 100 tasks each. A block of 100 requests is randomly distributed by the search server every 5 ms to a search engine node for service. The load-balancing threads on six nodes communicate with each other and even out the workloads. Responses on a representative node node3 are shown here to demonstrate the load-balancing strategy. Fig. 8(a) shows workloads, average estimates, and tracking differences computed on node3 in a network of six nodes with randomly generated requests. Node3 receives requests transferred from other nodes, shown as small upward transitions in the figure, and continues processing them. Three incoming blocks of requests (100 each) from the client are distributed to node3, shown as three large upward transitions in the figure, and node3 then balances the requests among the other five nodes according to the amount each is estimated (by node3) to be below node3's estimate of network average load. Fig. 8(b) shows the local workloads, average estimates, and tracking differences on another representative node node5.

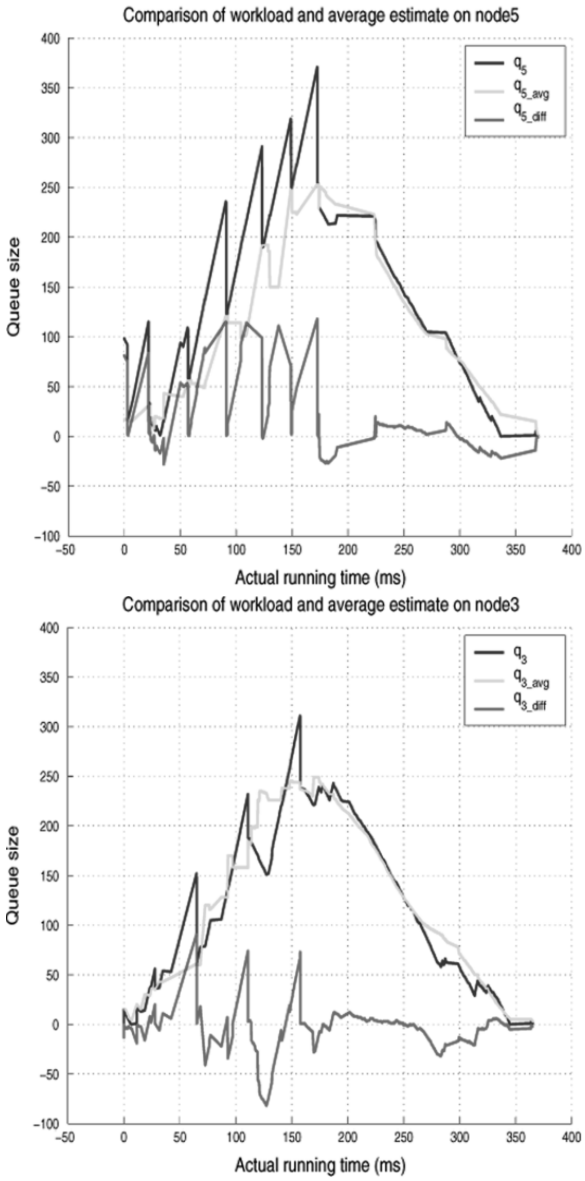

Fig. 8. Workloads, average estimates, and tracking differences on (a) node3 and (b) on node5 with randomly generated tasks.

Fig. 5 shows a comparison of average load estimates measured on six nodes. When a new block of search requests arrives, the receiving node updates its average, which creates a step transient as shown in Fig. 9. The load-balancing algorithm then evens out the tasks and brings the average estimate close to that on other nodes. For $t > 200$ ms, no new search requests arrive. The system settles to a balanced state, and the average estimates on the six nodes closely follow each other.

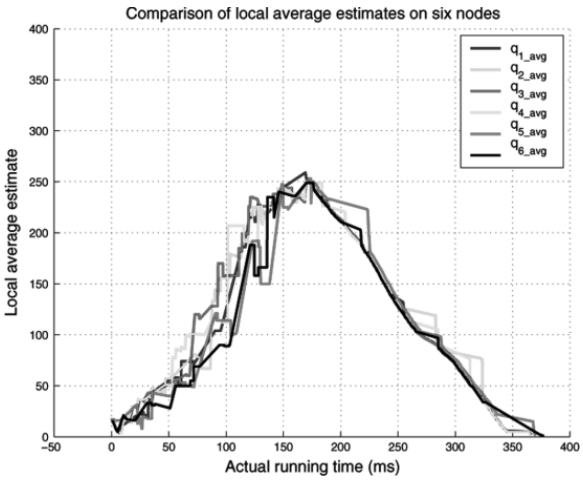

Fig. 9. Comparison of average estimates on six nodes with randomly generated requests.

Notice that the previous experiment used a group of three nodes for the incoming 1000 randomly generated tasks (ten blocks of 100 tasks each), and this experiment uses a group of six nodes to balance and service the incoming 2000 randomly generated tasks (20 blocks of 100 tasks each). The overall waiting time to complete all 2000 tasks (the maximum completion time in a group) is 377.4 ms in this experiment (see Fig. 9), while it took 327.1 ms to complete all 1000 tasks on three nodes in the previous experiment (see Fig. 6). For this case, the speedup is 73% when the number of nodes is doubled. Note that the scale-up efficiency will decrease with increasing number of nodes due to the increasing demands placed upon processor and network resources.

SECTION V. Conclusion

A load-balancing algorithm for parallel computing is modeled as a nonlinear dynamic system incorporating both time delays and processor resource constraints. A closed-loop controller is implemented that uses not only the local queue size, but also an estimate of the number of tasks in transit to the queue from other nodes. Experiments on a parallel DNA database demonstrate the efficacy of the load-balancing strategy. Future work includes scaling up the methodology for larger networks. For example, while each node broadcasts (by multicasting) both its queue size and the number of tasks in transit to other nodes, the local controller could perhaps send out tasks only to the nodes corresponding to the m largest values of load transfer portions (i.e., the m nodes with the largest p_{ij}). This would alleviate the communication cost (time) required to transfer tasks to all other nodes in the network.

ACKNOWLEDGMENT

The authors express gratitude to OPNET Technologies, Inc. for providing an academic license for their OPNET Modeler network simulation software.

References

1. Z. Tang, J.D. Birdwell, J. Chiasson, C.T. Abdallah, M.M. Hayat, "A time delay model for load balancing with processor resource constraints", *Proc. 43rd IEEE Conf. Decision and Control*, pp. 4193-4198, 2004-Dec.

2. Z. Tang, J. D. Birdwell, J. Chiasson, C. Abdallah, M. Hayat, "Closed loop control of a load balancing network with time delays and processor resource constraints" in Advances in Communication Control Networks, Germany, Berlin:Springer, vol. 308, pp. 245-268, 2004.

3. T. W. Wang, J. D. Birdwell, P. Yadav, D. J. Icove, S. Niezgoda, S. Jones, "Natural clustering of DNA/STR profiles", *10th Int. Symp. Human Identification*, 1999-Sep.

4. J. D. Birdwell, R. D. Horn, D. J. Icove, T. W. Wang, P. Yadav, S. Niezgoda, "A hierarchical database design and search method for CODIS", *10th Int. Symp. Human Identification*, 1999-Sep.

5. J. D. Birdwell, T.-W. Wang, R. D. Horn, P. Yadav, D. J. Icove, *Method of indexed storage and retrieval of multidimensional information*, 2004.
6. H. G. Rotithor, "Taxonomy of dynamic task scheduling schemes in distributed computing systems", *Proc. Inst. Elect. Eng.—Comput. Dig. Technol.*, vol. 141, no. 1, pp. 1-10, 1994.
7. M. A. Senar, A. Cortes, A. Ripoll, L. Hluchy, J. Astalos, "Dynamic load balancing" in Parallel Program Development for Cluster Computing: Methodology Tools and Integrated Environments, New York:Nova Science, pp. 69-95, 2001.
8. G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors", *J. Parallel Distrib. Comput.*, vol. 7, no. 2, pp. 279-301, 1989.
9. C. Z. Xu, F. C. Lau, "Analysis of the generalized dimension exchange method for dynamic load balancing", *J. Parallel Distrib. Comput.*, vol. 16, no. 1, pp. 385-393, 1992.
10. J. Watts, S. Taylor, "A practical approach to dynamic load balancing", *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 3, pp. 235-248, 1998.
11. A. Corradi, L. Leonardi, F. Zambonelli, "Diffusive load-balancing policies for dynamic applications", *IEEE Concurr.*, vol. 7, no. 1, pp. 22-31, Jan.–Mar. 1999.
12. H. Lin, R. Keller, "The gradient model load balancing method", *IEEE Trans. Softw. Eng.*, vol. 13, no. 1, pp. 32-38, 1987.
13. S. Zhou, "A trace-driven simulation study of dynamic load balancing", *IEEE Trans. Softw. Eng.*, vol. 14, no. 9, pp. 1327-1341, Sep. 1988.
14. R. Liiling, B. Monien, F. Ramme, "Load balancing in large networks: A comparative study", *Proc. 3rd IEEE Symp. Parallel and Distributed Processing*, pp. 686-689, 1991-Dec.
15. G. Fox, W. Furmanski, J. Koller, P. Simic, "Physical optimization and load balancing algorithms", *Proc. 4th Conf. Hypercubes Concurrent Computers and Applications*, pp. 591-594, 1989-Mar.
16. H. Chang, W.J. Oldham, "Dynamic task allocation models for large distributed computing systems", *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 12, pp. 1301-1315, Dec. 1995.
17. A. Y. Zomaya, C. Ward, B. Macey, "Genetic scheduling for parallel processor systems: Comparative studies and performance issues", *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 8, pp. 795-812, Aug. 1999.
18. D. Grosu, A. T. Chronopoulos, "Noncooperative load balancing in distributed systems", *IEEE Trans. Parallel Distrib. Syst.*, vol. 65, no. 9, pp. 1022-1034, Sept. 2005.
19. H. Kameda, E.-Z. S. Fathy, I. Ryu, J. Li, "A performance comparison of dynamic versus static load balancing policies in a mainframe", *Proc. 39th IEEE Conf. Decision and Control*, pp. 1415-1420, 2000-Dec.
20. E. Altman, H. Kameda, "Equilibria for multiclass routing in multi-agent networks", *Proc. 40th IEEE Conf. Decision and Control*, pp. 604-609, 2001-Dec.
21. H. Lin, C.S. Raghavendra, "A dynamic load-balancing policy with a central job dispatcher (LBC)", *IEEE Trans. Softw. Eng.*, vol. 18, no. 2, pp. 148-158, Feb. 1992.
22. B. Ghosh, F. T. Leighton, B. M. Maggs, S. Muthukrishnan, C. G. Plaxton, R. Rajaraman, A. W. Richa, "Tight analysis of two local load balancing algorithms", *SIAM J. Comput.*, vol. 29, no. 1, pp. 29-764, 1999.
23. M. H. Willebeek-LeMair, A. P. Reeves, "Strategies for dynamic load balancing on highly parallel computers", *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 9, pp. 979-993, Sep. 1993.
24. R. D. Nelson, T. K. Philips, "An approximation for the mean response time for shortest queue routing with general interarrival and service times", *Perf. Eval.*, vol. 17, pp. 123-139, 1993.
25. F. Spies, "Modeling of optimal load balancing strategy using queuing theory", *Microprocess. Microprogram.*, vol. 41, pp. 555-570, 1996.
26. A. Robertsson, B. Wittenmark, M. Kihl, M. Andersson, "Design and evaluation of load control in web server systems", *Proc. 2004 Amer. Control Conf.*, pp. 1980-1985, 2004-Jun.
27. J. L. Hellerstein, "Challenges in control engineering of computing systems", *Proc. 2004 Amer. Control Conf.*, pp. 1970-1979, 2004-Jun.
28. Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Pareky, C. Garcia-Arellano, "Using MIMO linear control for load balancing in computing systems", *Proc. 2004 Amer. Control Conf.*, pp. 2045-2050, 2004-Jun.

29. C. T. Abdallah, N. Alluri, J. D. Birdwell, J. Chiasson, V. Chupryna, Z. Tang, T. Wang, "A linear time delay model for studying load balancing instabilities in parallel computations", *Int. J. Syst. Sci.*, vol. 34, no. 10–11, pp. 563-573, Aug.–Sep. 2003.

30. J. D. Birdwell, J. Chiasson, Z. Tang, C. T. Abdallah, M. Hayat, T. Wang, "Dynamic Time Delay Models for Load Balancing part I: Deterministic Models" in Advances in Time-Delay Systems, Germany, Berlin:Springer-Verlag, vol. 38, pp. 355-370, 2003.

31. "Dynamic time delay models for load balancing PartII:A stochastic analysisof theeffectof delayuncertainty" in Advances in Time-Delay Systems, Germany, Berlin:Springer-Verlag, vol. 38, pp. 371-385, 2003.

32. J. Chiasson, Z. Tang, J. Ghanem, C. T. Abdallah, J. D. Birdwell, M. M. Hayat, H. Jerez, "The effect of time delays in the stability of load balancing algorithms for parallel computations", *IEEE Trans. Control Syst. Technol.*, vol. 13, no. 6, pp. 932-942, Nov. 2005.

33. *OPNET Modeler*, 2004, [online] Available: http://www.opnet.com/products/modeler/home.html.

34. Z. Tang, J. White, J. Chiasson, J. D. Birdwell, C. T. Abdallah, M. M. Hayat, "Closed-loop load balancing: Comparison of a discrete event simulation with experiments", *Proc. 2005 Amer. Control Conf.*, 2005-Jun.

35. P. Dasgupta, J. D. Birdwell, T. W. Wang, "Timing and congestion studies under PVM", *10th SIAM Conf. Parallel Processing for Scientific Computation*, 2001-Mar.

36. B. Lewis, D. J. Berg, Threads Primer: A Guide to Multithreaded Programming, CA, Mountain View:SunSoft Press, 1996.

37. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, PVM:Parallel Virtual Machine—A Users' Guide and Tutorial for Networked Parallel Computing, MA, Cambridge:MITPress, 1994.