ISSN 2047-3338

Performance Modeling of Distributed Load Balancing Algorithm Using Neural Networks

Seyed Hossein Kamali, Maysam Hedayati, Reza Shakerian and Saber Ghasempour

***Abstract*—This paper presents a new approach that uses neural networks to predict the performance of a number of dynamic decentralized load balancing strategies. A distributed multicomputer system using any distributed load balancing strategy is represented by a unified analytical queuing model. A large simulation data set is used to train a neural network using the back–propagation learning algorithm based on gradient descent. The performance model using the predicted data from the neural network produces the average response time of various load balancing algorithms under various system parameters. The validation and comparison with simulation data show that the neural network is very effective in predicting the performance of dynamic load balancing algorithms. Our work leads to interesting techniques for designing load balancing schemes (for large distributed systems) that are computationally very expensive to simulate. One of the important findings is that performance is affected least by the number of nodes and most by the number of links at each node in a large distributed system.**

***Keywords*— Load Balancing, Neural Network, Multi-Computer System and Simulation**

I. Introduction

ADVANCES in accurate performance models and appropriate measurement tools are driven by the demands of multicomputer system designs. Conventionally, these models and tools have been developed using analytical and simulation techniques. Though computationally inexpensive, analytical techniques alone do not always accurately represent the behaviour of the system under diverse conditions. In addition, the behaviour of complex systems is difficult to capture analytically. Simulations, on the other hand, are useful for analysing complex systems but are both computationally expensive and time consuming. Neural networks have been successfully applied for modelling nonlinear phenomena [3], [5], [8], [11], [12], [14], [20].

Application of neural networks for predicting the performance of multicomputer systems is our contribution to the search for new performance evaluation and prediction techniques.

Seyed Hossein Kamali , Islamic Azad University, Qazvin Branch, Qazvin,Iran (prjkamali@live.com)
Maysam Hedayati, Islamic Azad University, Ghaemshahr Branch, Ghaemshahr, Iran (hedayati_maysam@yahoo.com)
Reza Shakerian ,Payame Noor University,PO BOX 19395-3697 ,Tehran , Iran (r.shakerian@gmail.com)
Saber Ghasempour, Department of Mathematics, Payame Noor University,PO BOX 19395-3697 ,Tehran , Iran (s_ghasempor@mpnu.ac.ir)

The performance of multicomputer systems can be measured in terms of throughput, utilization, average task response time or inter-processor communication. The performance measure in our context is the average task response time which heavily depends on the underlying scheduling and load balancing mechanisms. In a wide range of environments, scheduling and load balancing cannot be done statistically and has to be done on-the-fly. For example, certain applications having dynamic structures can result in the creation of tasks at run time [6] which cannot be determined in advance. Furthermore, tasks initially assigned to processors can spawn more sub-tasks as computation proceeds [13], [18]. Fox et al. [6] show that on hypercube multi-computers, dynamic load balancing is useful for a number of problems such as event–driven simulations, adaptive meshes, many particle dynamics, searching of game trees in parallel chess programs, or simulation of neural networks with time dependent non–uniform activity. The response times of tasks can be considerably improved by migrating load from busy processors to idle processors. Dynamic load balancing is also essential for distributed computing environments such as workstation–based networks, where regular jobs can be migrated from a busy workstation to an idle workstation [10], [19]. In a study conducted for cluster of 70 Sun workstations, it was observed that one third of the workstations were idle, even at the busiest times of the day [21]. In a real–time environment, where periodically generated tasks need to be migrated from one node to another in order to meet critical deadlines, dynamic load balancing can improve the deadline missing probability.

As opposed to static scheduling techniques, dynamic scheduling strategies do not assume availability of a priori knowledge of tasks. Due to timing constraints, a dynamic scheduling algorithm needs to be fast enough to cope with time dependent fluctuations. The second feature that distinguishes dynamic task scheduling from static scheduling problems is that the notion of time is taken into consideration, that is, dynamic task scheduling acts according to the time dependent state of the system. Dynamic load balancing strategies are centralized [9], de-centralized [4], [11], [15], [19], or a combination of both [1].

Decentralized load balancing strategies have been classified into two categories: sender–initiated and receiver–initiated [23]. In a sender–initiated algorithm, the requests to transfer load are originated by heavily loaded nodes whereas an algorithm is said to be server–initiated if the requests are generated by lightly loaded nodes. Load balancing schemes can be classified further depending upon the system ar-chitecture: homogeneous or heterogeneous [17].

Given the diversity of load balancing strategies proposed in the literature and their dependence on a number of parameters, it is difficult to compare their effectiveness on a unified basis. In a previous study [2], we proposed an approach to predict and compare the performance (average response time) of different load balancing schemes on a unified basis, using simulation, statistics and analytical models. This paper presents an approach to predict the performance of different load balancing schemes using a new technique. The proposed approach, which uses neural networks, takes into account various system parameters such as system load, task migration time, scheduling overhead and system topology that can affect performance. We show that load balancing strategies, belonging to the sender–initiated class, can be modeled by a central-server queuing network. Through extensive simulation, a large number of values of the average queue length and the probability associated with task migration have been obtained. A neural network has been trained using the simulation data to model the relation between the queuing parameters and the system parameters. We have employed the back–propagation learning algorithm based on gradient–descent, to train our neural network. The network is then used to predict the response time of a system with any set of parameters, for a given load balancing strategy. Using the proposed performance evaluation approach, six load balancing algorithms have been modeled. We have compared the response time predicted by the model with the response time produced by simulation. The validation and comparison with simulation data show that the neural network is very effective in predicting the response time for dynamic load balancing. The neural network is then used to predict the response time for very large systems.

II. MODELING WITH NEURAL NETWORKS

Neural networks belong to the class of data–driven approaches, as opposed to model– driven approaches. The analysis depends on available data, with little rationalization about possible interactions. Relationships between variables, models, laws and predictions are constructed post facto after building a machine whose behaviour simulates the data being studied. The process of constructing such a machine based on available data is addressed by certain general purpose algorithms such as 'back–propagation.

Artificial neural networks are computing systems containing many simple non–linear computing units or nodes interconnected by links. In a 'feed-forward' network, the units can be partitioned into layers, with links from each unit in the k–th layer being directed (only) to each unit in the (k+ 1) j–th layer. Inputs from the environment enter the first layer, and outputs from the network are manifested at the last layer. A d-n-1 network, shown in Fig. 1, refers to a network with d inputs, n units in a single intermediate 'hidden' layer, and one unit in the output layer [24]. As we mention in the subsequent discussion, we train the neural network for learning an output versus a number of input. For example, if the neural network is trying to learn an output Y, as a function of x1, x2, x3, then it would have 3 inputs and one output. The number of hidden nodes to be chosen depends on the application. We have varied them from 3 to 5 and found that best results are obtained for the number of hidden nodes equal to 4. A weight or 'connection strength' is associated with each link, and a network 'learns' or is trained by modifying these weights, thereby modifying the network function which maps inputs to outputs.

We use such *d*-*n*-1 networks to learn and then predict the behaviour of dynamic load balancing algorithms. Each hidden and output node realizes a non-linear function of the form:

$$f(x_1, x_2, ..., x_m) = \frac{1}{1+e^{-\sum_{1 \le i \le m} w_i x_i + \Theta}} \quad (1)$$

Where *w*'s denote real-valued weights of edges leading into the node θ denotes the adjustable 'threshold' for that node, and *m* denotes the number of inputs to that node from nodes in the previous layer.

We use the error back-propagation algorithm of Rumelhart et al. [16], based on gradient-descent, to train the networks, with the goal of minimizing the mean squared deviation between the desired target values and network outputs, and averaged over all the training inputs. The training phase can be described as follows. In each step in the training phase, a J–tuple of inputs is presented to the network. The network is asked to predict the output value. The error between the value predicted (by the network) and the value actually observed (known data) is then measured and propagated backwards along the connections. The weights of links between units are modified by different amounts, using a technique which apportions 'blame' for the error to various nodes and links. A single 'epoch' (cycle of presentations of all training inputs) comprises applying all input patterns once and modifying the weights after each step. If the mean squared error exceeds some small predetermined value, a new epoch is started after termination of the current epoch.

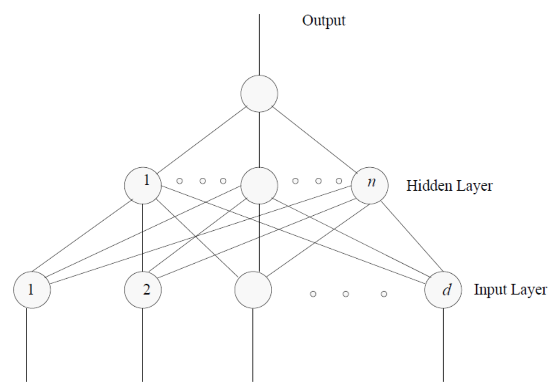

Fig 1. A d–n–1 feed forward neural network.

Learning is accomplished by the following rule that indicates how the weight of each connection is modified.

$$\Delta W_{ji}(n+1) = \eta\ (\delta_{pj} O_{pj}) + \alpha \Delta W_{ji}(n) \quad (2)$$

The parameters of the back–propagation algorithm are the 'learning rate' (η) and 'momentum' (α), which roughly describes the relative importance given to the current and past error values in modifying connection strengths. Here, n is the time index, W_{ji} is the weight from unit *i* to unit *j*, *p* is an index over the cases (input samples), and δ_{pj} is the propagated error signal seen at unit j in case *p*, and O_{pj} is the output of the corresponding unit. For the sigmoid activation function:

$$O_{pj} = \frac{1}{1 + \exp(-\sum_i w_{ji} O_\pi - \theta_j)} \quad (3)$$

The error signal is given by

$$\delta_{pj} = (t_{pj} - O_{pj}) O_{pj} (1 - O_{pj}) \quad (4)$$

For an output unit, we have

$$\delta_{pj} = O_{pj} (1 - O_{pj}) \sum_k \delta_{pk} w_{kj} \quad (5)$$

For a hidden unit where t_{pj} is the *j*–th element of the target for *p*–th input pattern.

III. CHARACTERIZING DISTRIBUTED LOAD BALANCING

We consider multicomputer systems that consist of homogeneous processing nodes connected with each other through a symmetric topology, i.e., each node is linked to the same number of nodes. The number of links per node, *L,* is called the degree of the network. We assume that the task arrival process is Poisson and tasks are submitted to each node with an average arrival rate of λ tasks per time-unit at each node. When a task arrives at a node, it is either scheduled to the local execution queue or migrated to one of the neighbours connected with it via a communication channel. Information gathering and scheduling takes a certain amount of time, which is assumed to be exponentially distributed with an average of $1/\mu_s$ time-units. A communication server at each link of a node transfers a task from one node to another with an average of $1/\mu_c$ time-units. The task communication time is also assumed to be exponentially distributed. At each node, incoming traffic from other nodes joins locally generated traffic, and all traffic is handled with equal priority. Each node maintains an execution queue in which locally scheduled tasks are served by a CPU on the FCFS basis. The load of a node is expressed in terms of the length of the execution queue. Execution time is also assumed to be exponentially distributed with an average of $1/\mu_E$ time-units. Table 1 describes the meanings of a number of symbols used in this paper.

TABLE I. SYMBOLS AND THEIR MEANINGS

Symbol	Meaning
λ	External arrival rate at each node
μ_S	Average task scheduling rate
μ_E	Average task execution rate
μ_C	Average task migration rate
ρ	Average load on each node
$E[N_E]$	Average execution queue length
T_u	Load information update period
P_0	Probability of scheduling task locally
P_j	Probability of migration task to j-th neighbor
$E[R]$	Average cumulative task response time
$E[R_{phase1}]$	Average task response time in the first phase
$E[R_{phase2}]$	Average task response time in the second phase

A. Load Balancing Strategies

In general, a load balancing strategy consists of three policies [4]: transfer policy, location policy and information collection policy. A transfer policy determines whether a task should be migrated or not. A location policy sets a criterion to select a node if the task is to be migrated.

An information policy decides how information exchange among different nodes is carried out. We have analysed six sender–initiated distributed load balancing schemes using different transfer, location and information exchange policies. In the first three load balancing strategies, information about the load and the status of other nodes is collected at the time a task is scheduled for execution or migration. In the last three strategies, nodes exchange load information with their neighbours after every (fixed) period of time. The names of strategies are prefixed by F and P, denoting fresh and Periodic information exchanges, respectively; these strategies are explained below.

FRandom: In this strategy, the task scheduler calculates the average of the local load and the load of all neighbors. If the local load exceeds the average, the task is sent to a randomly selected neighbour.

FMin: In this strategy, the task scheduler first selects the neighbour with the least load. The task is transferred to that neighbour if the difference between the local load and load of that neighbour exceeds a certain threshold (threshold is 1 in our experiments). The local node is given priority, since migrating a task to a neighbour incurs communication and scheduling delays.

FAverage: In this strategy, the task scheduler calculates the average of all neighbours' load and its own load. If the local load exceeds the average, the task is sent to the neighbor with the minimum load. Otherwise, the task is sent to the local execution queue.

PRandom: This strategy is similar to FRandom except that information is exchanged periodically.

PMin: This strategy is similar to FMin except that information is exchanged periodically.

PAverage: This strategy is similar t o FAverage except that information exchange is periodic.

B. Analytical Modeling

First, we show how the class of distributed load balancing strategies described above can be modeled by an open network central server queuing model. When a task migrates from one node to another, it enters a statistically identical node. Therefore, the steady–state behaviour of nearest neighbour load balancing can be approximated by the central-server open queuing model. A distributed multicomputer system consisting of 16–node hypercube topology, with distributed load balancing, is illustrated in Figure 2. Here, each node of the system can be represented by a central-server open queuing network. As described in the next section, simulation results obtained on actual network topologies are very close to the analytical results determined from this model, validating that the proposed model of Figure 2 indeed represents the task scheduling and migration process. The model consists of a waiting queue, *L* communication queues and an execution queue.

The duration of a task's residence time in the system consists of two phases. In the first phase, the task may repeatedly migrate: it waits in the waiting queue, gets service from the scheduler, waits in the communication queue, and then transfers to another node. At that point, the

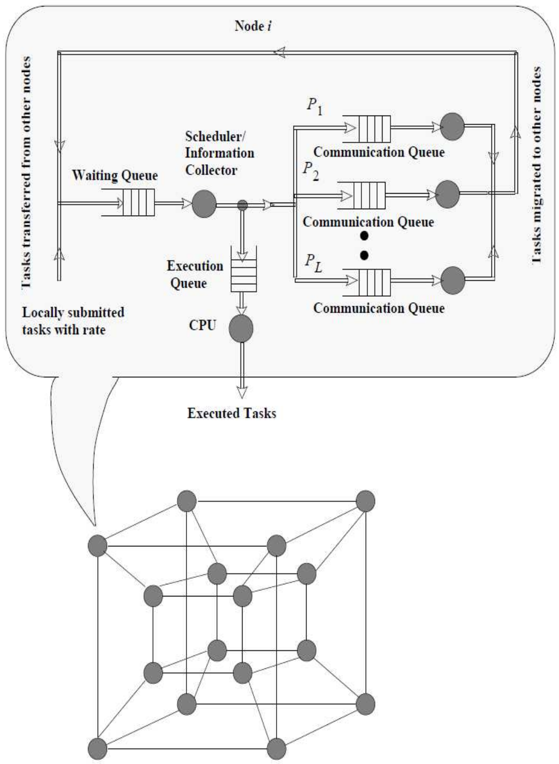

Fig 2. A multicomputer system connected in a 16–node hypercube topology

same cycle may start all over again. In the first phase, each task can be viewed as occupying either the task scheduler or one of the communication links.

Each node is represented by open network central-server queuing model. Once the task is scheduled at the execution queue of a node, the second phase starts, which includes the queuing and service time at the CPU. It follows that the open central server model can be solved by the Jacksonian network [22], which has the product form solution; the joint probability of k_j tasks at queue

j(j =0, 1, . . ., *L)* is given by the product:

$$p(k_1,k_2,...,k_L) = \prod_{j=0}^{L} p_j(k_j) \qquad (6)$$

Where $p_j(k_j)$ is the probability that k_j tasks are at the j–th queue, given by:

$$p(k) = (1-\rho_j)\,\rho_j^{k_j} \qquad (7)$$

For the J–th component, the average utilization ρ_j is equal to λ_j/μ_j the equation implies that the lengths of all queues are mutually independent in a steady state. The average queue length and the average response time are given by:

$$E[N_j] = \frac{\rho_j}{1-\rho_j} \quad and \quad E[N_j] = \frac{\rho_j}{\lambda_j(1-\rho_j)} \qquad (8)$$

The average number of tasks at a node is the sum of the average number of tasks at each component of a node and is given by:

$$E[N] = \sum_{j=0}^{L} E[N] = \sum_{j=0}^{L} \frac{\rho_j}{1-\rho_j} \qquad (9)$$

From which the average response time before the task is scheduled in the execution queue can be computed as:

$$E[R_{phase}] = \frac{1}{\lambda}\sum_{j=0}^{L}\frac{\rho_j}{1-\rho_j} = \frac{(P_0\,\mu)^{-1}}{1-\lambda(P_0\mu)^{-1}} + \sum_{j=1}^{L}\frac{P_j\,(P_0\mu_j)^{-1}}{1-\lambda P_j\,(\mu_j P_0)^{-1}} \qquad (10)$$

Where λ_0 is replaced by λ / p_0 and $\lambda\,(j \geq 1)$ is replaced by $\lambda\, P_j / P_0$. Once a task is scheduled at a local execution queue, the response time is given by:

$$E[R_{phase2}] = \frac{E[N_E]}{\lambda} \leq 1.0 \qquad (11)$$

Where *E [N_E]* is the average execution queue length. The complete response time, therefore, is:

$$E[R] = E[R_{phase1}] + E[R_{phase2}] \qquad (12)$$

The above equation implies that, for a given load, $(\rho = \lambda / \mu_E)$, μ_0 and $\mu_j's$, the response time yielded by a load balancing strategy can be calculated if P_0 and *E [N_E]* are known. Here, P_0, is the probability with which a load balancing strategy schedules the tasks locally, and *E [N_E]* is the average execution queue length. For clarity, we replace μ_0 by μ_S, representing the average task scheduling rate. We also assume that $\mu_j's$ for each link is the same and the average communication rate is represented by μ_C.

C. Simulation Environment

The above mentioned load balancing strategies were simulated. Our simulator, which is of discrete-event type, takes as input the topology of the network along with $\lambda, \mu_S, \mu_S, \mu_C, \mu_E$ length of simulation run, and choice of load balancing strategies and their associated parameters. Simulation can be run by using different sets of random number streams. Initial transients in the simulation are removed by ignoring the initial outputs until the system enters into a steady state. Each data point is produced by taking the average of a large number of independent simulation runs and then by taking their means. The confidence interval for each data point has been obtained with 99% confidence interval and the width of the interval is within 5% of the mean values.

A number of simulations were conducted to obtain 500 data values for P_0 and *E [N_E]*, for each strategy. Three different topologies have been selected, including the ring (L = 2), the 16-node hypercube (L = 4) and the 16-node folded hypercube (L = 5) [7]. Points for one particular strategy are obtained for each topology by fixing one parameter and varying the rest. In most cases, λ is varied from 0.3 to 0.9 tasks per time-unit, and μ_C is varied from 8 to 16 tasks per time-unit. We assume that the average task execution rate μ_E is 1 task per time unit. In-stead of the actual load $(\rho = \lambda / \mu_E)$, this enables us to consider λ as the parameter representing load per node. For strategies that require periodic information update, the update time period, T_u, is varied from 0.5 to 1.5 time units. The scheduling overhead includes the exchange of state information and the execution of the scheduling algorithm itself. We have assumed an average scheduling time, $1/\mu_S$ which in turn, can be normalized with respect to the execution time $1/\mu_E$. In other words, when μ_S is 10 tasks/time-unit and μ_E, is 1 task/time-unit, the average task scheduling time is 1/10 of the execution time. For the simulation data, lic is varied from 8 to 16 tasks per time unit. Since we simulated the actual interconnection network topologies, mentioned above, and not the central server model shown in Figure 2, P_0 and $E [N_E]$ were observed from the simulation data. The probability, P_0, is estimated by dividing the average number of locally scheduled tasks by the total number of tasks arrived, at each node.

IV. Results

In this section, we present the results showing the average response time predicted by the analytical model based on the values of P_0 and $E [N_E]$ predicted by the neural network. For the experiments described in this paper, the learning rate and momentum were varied to train the network to give small mean squared error. For the modeling of probability, we used a network with no hidden layer. For the estimation of queue length a network with one hidden layer was used. The number of hidden nodes was varied; the best results were obtained for 4 hidden nodes. TableII gives the root mean square errors in P_0 and $E [N_E]$ between neural networks.

TABLE II. THE ROOT MEAN SQUARE ERROR BETWEEN SIMULATION AND NEURAL NETWORK IN MODELING P_0 AND $E [N_E]$

Strategy	P_0	$E [N_E]$
FRandom	0.4705	3.6200
FMin	0.0654	0.9355
FAverage	0.1486	0.7733
PRandom	0.1358	0.3139
PMin	0.1354	0.5357
PAverage	0.3636	0.4053

Results and simulation data as can be seen the error yielded by the neural network is less than 0.5 for P0 and less than 0.1, in most cases, for $E [N_E]$.

Using the values of P_0 and $E [N_E]$ predicted by the neural network, the average response time is calculated through the queuing model and is compared with the observed simulation results. We have divided these results in three parts: training, testing and prediction (Table III).

V. Conclusion

We presented a new approach to model the performance of several distributed dynamic load balancing algorithms in a multicomputer environment. The response time predicted

TABLE III. AVERAGE RESPONSE TIMES OBTAINED BY THE NEURAL NETWORK FOR THE SIX STRATEGIES, AT LOW, MEDIUM, AND HIGH LOADING CONDITIONS, ON A 16–NODE HYPERCUBE TOPOLOGY

Load	Strategy	Simulation	Neural Net.	Difference %
λ = 0.4 (Low)	FRandom	1.243	1.277	2.73
	FMin	1.403	1.403	0.03
	FAverage	1.186	1.193	0.62
	PRandom	1.239	1.259	-1.61
	PMin	1.412	1.480	4.82
	PAverage	1.214	1.292	6.35
λ = 0.6 (Medium)	FRandom	1.515	1.545	4.40
	FMin	1.585	1.534	-3.25
	FAverage	1.378	1.348	-2.19
	PRandom	1.514	1.496	-1.15
	PMin	1.624	1.598	-1.62
	PAverage	1.475	1.401	-4.97
λ = 0.8 (High)	FRandom	2.056	2.165	5.35
	FMin	1.965	2.083	6.04
	FAverage	1.802	1.929	7.04
	PRandom	2.048	2.085	1.82
	PMin	2.080	2.123	2.09
	PAverage	1.974	1.986	-0.64

by the neural network closely approximates the response time obtained through simulation. Our performance evaluation methodology accurately determines the variations in performance of all algorithms with a wide range of system parameters. Generalization ability of the neural networks helps in successfully analyzing the performance of very large systems. The neural network model is an effective tool for modeling the performance of dynamic load balancing algorithms. The study has also revealed a number of future research problems. For example, one can consider the nodes in the system to be heterogeneous instead of homogeneous. Furthermore, one can consider an environment where tasks are initially submitted to a node from L different links rather than a single queue. However, this assumption will require new analysis of the central-server queuing model.

References

[1] I. Ahmad and A. Ghafoor, “A Semi Distributed Task Allocation Strategy for Large Hypercube Supercomputers,” Proc. of Supercomputing’90, Nov. 1990, pp. 898–907.

[2] I. Ahmad, A. Ghafoor and K. Mehrotra, “Performance Prediction for Distributed Load Balancing on Multicomputer Systems,” Proc. of Supercomputing’91, Nov. 1991, pp. 830–839.

[3] S. Ahmad and G. Tesauro, “Scaling and Generalization in Neural Networks: A Case Study,” in D. S. Touretzky et al., eds., Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann, 1988, pp. 3–10.

[4] D. L. Eager, E. D. Lazowska and J. Zahorjan, “Adaptive Load Sharing in Homogeneous Distributed Systems,” IEEE Trans. on Software Engg.,vol. SE–12, May 1986, pp. 662–675.

[5] S. E. Fahlman and C. Lebiere, “The Cascade–Correlation Learning Architecture,” Advances in Neural Information Processing Systems 2, D. Touretzky ed., Morgan Kaufmann, San Mateo (CA), 1990, pp. 524–532.

[6] G. C. Fox, A. Kolawa and R. Williams, “The Implementation of a Dynamic Load Balancer,”Proc. of SIAM Hypercube Multiprocessors Conf., 1987, pp. 114–121.

[7] A. Ghafoor, T. R. Bashkow and I. Ghafoor, "Bisectional fault–Tolerant Communication Architecture for Supercomputer Systems, "IEEE Trans. on Computers, vol. 38, pp. 10, October 189, pp. 1425–1446.

[8] A. Lapedes and R. Farber, "Nonlinear Signal Processing Using Neural Networks: Prediction and System Modeling," Tech. Rep. LA–UR–87–2662, Los Alamos National Lab., NM, 1987.

[9] H.–C. Lin and C. S. Raghavendra, "A Dynamic Load balancing Policy with a Central Job Dispatcher (LBC)," Proc. of The 11–th Int'l conf. on Distributed Computing systems, May 1991, pp. 264–271.

[10] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," Proc. of ACM Computer Network Performance Symposium, April 1982, pp. 47– 55.

[11] R. Luling, B. Monien and F. Ramme, "Load Balancing in Large Networks: A Compar ative Study," Proc. of The Third Symposium on Parallel and Distributed Processing, December 1991, pp. 686–689.

[12] J. Moody and C. Darken, "Fast Learning in Networks of Locally–Tuned Processing Units," Neural Computing, 1(2), 1989, pp. 281–294.

[13] B. A. A. Nazief, "Empirical Study of Load Distribution Strategies on Multicomputers," Ph.D Dissertation, University of Illinois at Urbana–Champaign, 1991.

[14] J. Platt, "A Resource–Allocation Network for Function Interpolation," Neural Computing, 3(2), 1990, to appear.

[15] X. Qian and Qing Yang, "Load Balancing on Generalized Hypercube and Mesh Multiprocessors with LAL," Proc. of The 11th Int'l conf. on Distributed Computing systems, May 1991, pp. 402–409.

[16] D. E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning internal representations by error propagation," Parallel Distributed Processing, Vol. 1, Ch. 8, 1986, MIT Press, Cambridge (MA).

[17] M. Schaar, K. Efe, L. Delcambre and L. N. Bhuyan, "Load Sharing with Network Cooperation," Proc. of The Fifth Distributed Memory Computing Conference, April 1990, pp. 994–999.

[18] W. Shu, "Chare Kernel and its Implementation on Multicomputers," Ph.D Dissertation, University of Illinois at Urbana–Champaign, 1990.

[19] A. Svensson, "History, an Intelligent Load Sharing Filter," Proc. of 10–th Intl. Conf. on Distributed Computing Systems, 1990, pp. 546–553.

[20] M. F. Tenorio and W. Lee, "Self–Organizing Neural Networks for the Identification Problem," Advances in Neural Information Processing Systems 1, D. Touretzky, ed., Morgan–Kaufmann, San Mateo, 1989, pp. 57–64.

[21] M. M. Theimer and K. A. Lantz, "Finding Idle Machines in a Workstation–based Distributed System,"Proc. of 8–th Intl. Conf. on Distributed Computing Systems, 1988, pp. 112–122.

[22] K. S. Trivedi, Probability & Statistics with Reliability, Queuing and Computer Science Applications, Prentice–Hall, Inc., Englewood Cliffs, NJ, 1982.

[23] Y. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems," IEEE Trans. on Computers, C–34 no. 3, March 1985, pp. 204–217.

[24] A. S. Weigend, B. A. Huberman and D. E. Rumelhart, "Predicting the Future: A Con-nectionist Approach," submitted to the International Journal of Neural Systems, April 1990.