Dynamic Fuzzy Load Balancing on LAM/MPI Clusters with Applications in Parallel Master-Slave Implementations of an Evolutionary Neuro-Fuzzy Learning System

Lotika Singh, Apurva Narayan, Satish Kumar

***Abstract*— In the context of parallel master-slave implementations of evolutionary learning in fuzzy-neural network models, a major issue that arises during runtime is how to balance the load—the number of strings assigned to a slave for evaluation during a generation—in order to achieve maximum speed up. Slave evaluation times can fluctuate drastically depending upon the local computational load on the slave (given fixed node specifications). Communication delays compound the problem of proper load assignment. In this paper we propose the design of a novel dynamic fuzzy load estimator for application to load balancing on heterogeneous LAM/MPI clusters. Using average evaluation time and communication delay feedback estimates from slaves, strings assignments for evaluation to slaves are dynamically changed during runtime. Extensive tests on heterogenous clusters shows that considerable speedups can be achieved.**

I. Motivation and Review

Evolutionary algorithms (EAs) are being increasingly employed for learning applications in computational intelligence model development due to their powerful search capabilities. They can be used to identify optimal network architectures, estimate network parameters, and also eliminate redundant features through global search. However, EAs don't come for free: they require problem specific and careful tuning of parameters, and require vast computational resources. Fortunately, the intrinsically parallel nature of EAs permits them to be implemented on computing clusters which are becoming increasingly common in academic and commercial environments [1]. Research on parallel implementations of EAs in the context of neuro-fuzzy systems (or even otherwise) is therefore of significant importance if any realistic simulations are to be performed.

The basic approach to applying EAs to CI model learning is to encode the phenotypic representation into a genotypic representation (encoding it as a string of mixed integer-floating point numbers) and then allowing a population of such strings to evolve in time. Evolution of the population involves various operations such as crossover, mutation, and selection of strings for the new generation based on an individuals' fitness by evaluation of a fitness function. It turns out that the fitness function evaluation is the most costly in terms of CPU resource (and therefore time). Parallel implementations of evolutionary learning algorithms thus typically distribute strings for evaluation across multiple processors in order to bring down program run times. Parallel implementations are possible on several available platforms: parallel virtual machine (PVM) [2], message passing interface (MPI) [3, 4], and Globus [5]. Of these, MPI has emerged as the *de facto* standard for parallel implementations [1].

A major issue that arises in implementing a parallel algorithm on a cluster is that of load balancing. If we consider a standard master-slave kind of implementation, then in each cycle, the master is responsible for assigning work units to slaves, collecting the intermediate results back from the slaves, and then assigning the next unit of work to slaves, and so on. Good load balancing leads to a situation where once the master has assigned work units to slaves in round-robin fashion, slaves are ready to begin sending back results to the master for collation, and thus the master (and slave) idle times are minimized. When we say "idle" we mean idle from the point of view of the work assignments from the master, and not the true processor loads. If the load on the individual slaves is not properly balanced, then the master wastes time waiting for jobs to finish on slow slaves, and fast slave resources go waste since they remain idle—waiting for the master to assign them the next batch of work. Further, one usually employs clusters whose processor specifications are not known, and even if they were known, the load on each processor is random in the sense that it is not known in advance as to how many users or which programs will be running on a specific processor at any point of time. Load balancing on heterogenous clusters is thus an important research problem.

The issue of load balancing in various contexts has been studied in the literature by several researchers. In [6] a Java-based distributed evolutionary computing software is used to enhance the concurrent processing and performance of evolutionary algorithms by allowing inter-communications of sub-populations among various computers over the Internet. Reference [7] introduces dynamic mapping and load balancing of parallel programs in MIMD multicomputers, based on coordinated migration of processes of a parallel program. Load balancing has also been introduced for grid computing applications based on the modification of the data distributions used in scatter operations [8]. A parallel

The authors are with the Dayalbagh Educational Institute, Dayalbagh, Agra 282005 India.

Lotika Singh, *Student Member, IEEE*, is a Junior Technical Assistant in the Department of Physics and Computer Science–email: lotikasingh@ieee.org.

Apurva Narayan, *Student Member, IEEE*, is a student of B.Sc. (Engg.) Electrical Engg. Final Year in the Faculty of Engineering–email: apurvanarayan@gmail.com.

Satish Kumar, *Senior Member, IEEE*, is a Professor in the Department of Physics and Computer Science–email: skumar_db@ieee.org.

load balanced implementation of genetic programming (GP) based on the cellular model has been presented in [9]. In [10] an adaptive memory allocation technique has been introduced for clusters to handle large data-intensive jobs. In [11], an on-line dynamic scheduling policy that manages multiple job streams across both single and multiple cluster computing systems improving the mean response time and system utilization. The idea of dynamic scheduling system and a load balancing system for clusters based on PVM was introduced in [12]. Distributed load-balancing algorithms for dynamic networks have been investigated in [13], and in [14], sender-receiver pair negotiation takes into consideration the processing speeds of the sender and receiver, and relative workloads to decide allocations of work units. The design and implementation of a dynamic fuzzy load balancing service running in a distributed object computing environment is described in [15, 16]. From the above review, clearly, the application of fuzzy logic estimation for load distribution in the context of LAM/MPI clusters is a new area of research, and the present work represents one step in furthering these ideas.

The paper is organized as follows: Section II briefly provides an overview of the mater-slave model for evolutionary learning in the ASuPFuNIS model; Section III describes the fuzzy logic estimation model; Section IV describes the load estimation algorithm; Section V presents experimental results; and Section VI concludes the paper.

II. Master-Slave Model for Parallel Evolutionary Learning

Due to the dynamic nature of both network traffic (which can affect the efficiency of message transfer) as well as the fluctuating and unpredictable nature of CPU loads on nodes in the cluster, the problem of load balancing becomes an immediate candidate for fuzzy logic based load dispatch. Towards facilitating the presentation of our strategy for fuzzy load balancing we will consider the problem of evolutionary learning as applied to a fuzzy-neural network model: *asymmetric subsethood product fuzzy-neural inference system* (ASuPFuNIS) developed in our laboratory [17].

Evolutionary learning in ASuPFuNIS is implemented by assuming a population of strings that encode different network instances and by allowing a population of such strings to evolve using *differential evolution* (DE) [18, 19] to identify network architecture, estimate parameters and eliminate redundant features. In the present example, to achieve a speedup, the evaluation operation is parallelized using a Master-Slave implementation in which the population is divided into S segments assuming S slave processors, and each segment is sent to a slave for evaluation, this being the most costly operation in the cycle [20]. In this model, the master is responsible for basic DE operations such as crossover and mutation, which are not very time intensive. Evaluations are performed on slaves. Using this master-slave approach, we have the task distributions for parallelization of DE as given in Table I.

TABLE I
MASTER-SLAVE PARALLELIZATION STRATEGY EMPLOYED FOR LOAD BALANCING EXPERIMENTS

```
Master
1) Initialize population.
2) Evaluate population for the first time.
3) Identify the best string in the population: bestinpop.
4) For each member of the population, select a pair of random but
   unique indices R1 and R2 used to select mutation-driving members
   from the population.
5) Select corresponding $X_{r1,g}, X_{r2,g}$ vectors to drive mutations.
6) Select the best member in population bestinpop.
7) Drive mutations $T_{i,g+1} = X_{best,g} + F(X_{r1,g} - X_{r2,g})$.
8) Divide the total population equally between each slave (Cluster 1 is a
   uniform cluster) and send share of population members (trial vectors)
   to each slave.
9) Receive the new updated population segment from each slave and
   perform selection.
10) Repeat Steps 3-9 until the maximum number of generations has not
    elapsed.

Slaves
1) Receive share of population members from Master.
2) Evaluate fitnesses of trial vectors.
3) Send updated population back to master.
4) Repeat Steps 1-4 until the maximum number of generations has not
   elapsed.
```

Our motivation for the present work is to balance the string evaluation load on a slave such that the master does not *stall*—waiting for slaves to complete their task. This load dispatch will depend on various factors:

- Population size
- String length (which depends on the phenotype architecture and the problem input-output dimension)
- The number of patterns in the training set (since this would affect the evaluation time)
- Load on the slave processor
- Network congestion and buffering delays (which can become more pronounced when cluster nodes are physically distributed, and if for example, some slave nodes are connected via WiFi.)

On an unloaded and uniform cluster, assignment of string populations is trivial; on heterogeneous clusters this presents a challenge.

III. Fuzzy Logic Load Dispatch Model

A. Fuzzy Variables: Relative Slave Evaluation Time and Relative Slave Transfer Delay

Given a specific problem (which determines the number of inputs and outputs), and fixing the number of rule nodes in the network, sets the string length. If we assume that the population comprises only genotypes corresponding to a single network architecture i.e., all the strings have the same length, the population is uniform.[1] We assume that the number of strings to be allocated to a slave for evaluation depends on two factors: the *relative slave evaluation time* and the *relative slave communication delay*.

[1]Some variants of evolutionary algorithms admit variable string lengths.

Towards defining these variables, we introduce the following notation. For slave i, let the total time elapsed at the master between a send and a receive at generation g be $\tau_{i,g}^{T}$; the time for evaluation of an allocated segment of population on a slave at generation g be $\tau_{i,g}^{E}$; and the population segment size allocated at generation g be n_i. Then, the total time lost in network send-receive transfers excluding the slave evaluation time will be

$$\tau_{i,g}^{N} = \tau_{i,g}^{T} - \tau_{i,g}^{E}. \tag{1}$$

In order to remove dependence on exact values of delay, and since the allocation to a slave essentially depends on its *relative* performance in the cluster (since all strings must be allocated to some evaluating entity), we normalize the values by introducing the following definitions. The *relative slave evaluation time* is defined as,

$$\bar{\tau}_{i,g}^{E} = \frac{\tau_{i,g}^{E}}{\sum_{j=1}^{S} \tau_{j,g}^{E}} \tag{2}$$

and the *relative slave communication delay* is

$$\bar{\tau}_{i,g}^{N} = \frac{\tau_{i,g}^{N}}{\sum_{j=1}^{S} \tau_{j,g}^{N}} \tag{3}$$

In this paper, we propose a fuzzy logic load dispatch model based on these two input variables.

B. Definition of Fuzzy Sets on Input and Output Universes of Discourse

The two input universes of discourse considered are: $\bar{\tau}_{i,g}^{E}$ and $\bar{\tau}_{i,g}^{N}$, and the output universe of discourse is the *estimated slave load absorption capacity* c_i. For convenience of definition, we assume each universe of discourse to range in the interval [0,10], and values generated from Eqns. 2 and 3 will be scaled appropriately.

On each universe of discourse we introduce nine linguistic variables: Extremely low (EL), Very low (VL), Low (L), Low-medium (LM), Medium (M), Medium-high (MH), High (H), Very high (VH), and Extremely high (EH). For simplicity, these are kept common for both the input as well as output universes of discourse. As we mention in the discussion at the end of the paper, these sets can have adaptive parameters dependent on the dynamics of the cluster. The input-output fuzzy sets are modelled by triangular membership functions shown in Fig. 1.

C. Definition of Fuzzy Rule Base

Load balancing is achieved by introducing fuzzy rules that specify a mapping from the two-dimensional input space to the output space. Considering input variables for slave i, one can formulate MA type fuzzy rules of the following form:

$$\text{If } \bar{\tau}_{i,g}^{E} \text{ is EL} \wedge \bar{\tau}_{i,g}^{N} \text{ is EL then } c_i \text{is EH} \tag{4}$$

reflecting the heuristic that if the slave communication delay and the slave evaluation time are both Extremely Low (EL), then the estimated load absorption capacity for that slave is Extremely High (EH). Table II portrays a fuzzy rule base of

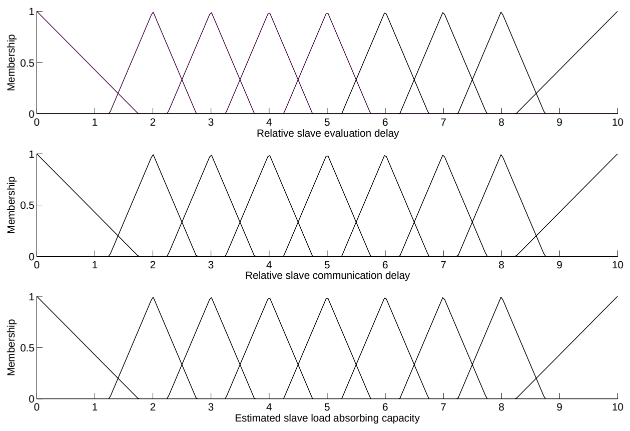

Fig. 1. Fuzzy sets defined on input and output universes of discourse

TABLE II
RULE BASE I: SYMMETRIC FUZZY RULE BASE OF 81 FUZZY RULES

$\bar{\tau}_{i,g}^{E}$									
EL	EH	EH	VH	VH	H	MH	M	ML	L
VL	EH	VH	VH	H	MH	M	ML	L	L
L	VH	VH	H	MH	M	ML	L	L	L
LM	VH	H	MH	M	ML	L	L	L	VL
M	H	MH	M	ML	L	L	L	VL	VL
MH	MH	M	ML	L	L	L	VL	VL	VL
H	M	ML	L	L	L	VL	VL	VL	EL
VH	ML	L	L	L	VL	VL	VL	EL	EL
EH	L	L	L	VL	VL	VL	EL	EL	EL
$\bar{\tau}_{i,g}^{N}$	EL	VL	L	LM	M	MH	H	VH	EH

81 rules formulated along these lines, in which the fuzzy estimator is equally sensitive to both $\bar{\tau}_{i,g}^{E}$ and $\bar{\tau}_{i,g}^{N}$. The estimation surface generated by this rule base is shown in Fig. 2.

Table III portrays a fuzzy rule base of 81 rules which reflects an estimation strategy that is more sensitive to $\bar{\tau}_{i,g}^{E}$ than to $\bar{\tau}_{i,g}^{N}$. By way of comparison, the estimation surface generated by this rule base is shown in Fig. 3.

IV. LOAD ESTIMATION ALGORITHM

The vanilla version of the fuzzy load balancing algorithm is as follows.

▮ *Phase I: Initial cluster sensing phase*

TABLE III
RULE BASE II: ASYMMETRIC FUZZY RULE BASE OF 81 FUZZY RULES

$\bar{\tau}_{i,g}^{E}$									
EL	EH	EL	VH	VH	H	H	H	MH	MH
VL	VH	VH	H	H	MH	MH	MH	MH	MH
L	H	H	MH	MH	M	M	M	M	ML
LM	MH	MH	M	M	ML	ML	ML	ML	ML
M	M	M	ML	ML	ML	ML	L	L	L
MH	ML	ML	L	L	L	L	L	L	L
H	L	L	VL	VL	VL	VL	VL	VL	VL
VH	VL	VL	VL	VL	VL	EL	EL	EL	EL
EH	VL	EL	EL	EL	EL	EL	EL	EL	EL
$\bar{\tau}_{i,g}^{N}$	EL	VL	L	LM	M	MH	H	VH	EH

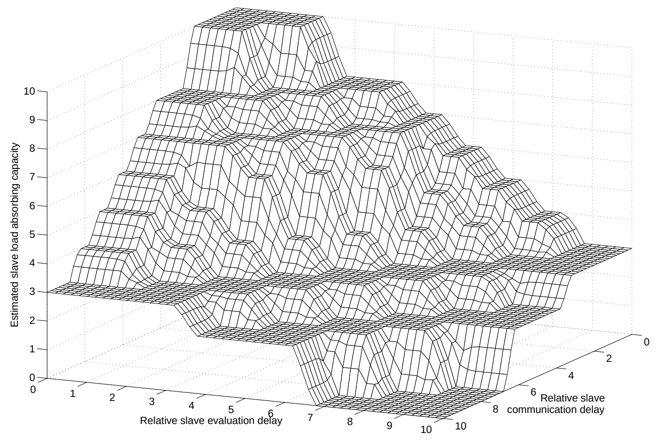

Fig. 2. Fuzzy estimation surface portraying output c_i as a function of $\bar{\tau}^E_{i,g}$ and $\bar{\tau}^N_{i,g}$ in accordance with Rule Base I.

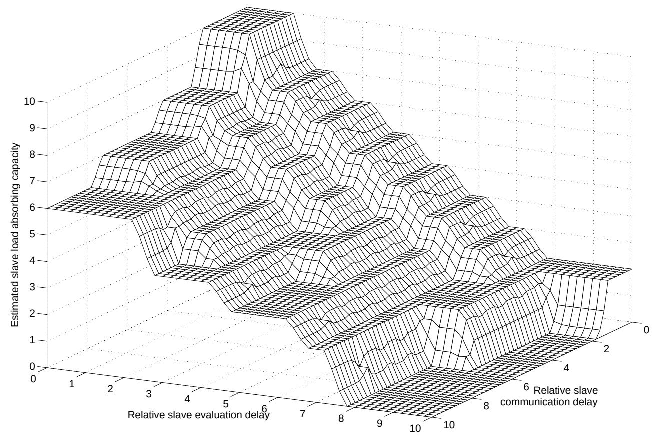

Fig. 3. Fuzzy estimation surface portraying output c_i as a function of $\bar{\tau}^E_{i,g}$ and $\bar{\tau}^N_{i,g}$ in accordance with Rule Base II.

1) Divide the population N into S segments (for S slaves). If $r = N \mod S$ then assign $\lceil N/S \rceil$ strings to r slaves; $\lfloor N/S \rfloor$ strings to the remaining $N - r$ slaves.
2) Time stamp each slave and send population for evaluation only
3) Receive local slave evaluation time, and compute communication delays for each slave
4) Repeat the Steps 1–3 for two send-receive rounds, and average out statistics for estimate of communication delay and evaluation time for each slave.

▮ *Phase II: Regular run phase*

1) Re-assign segments using fuzzy estimator
2) Perform DE for two generations using the string assignments from Phase II Step 1, collecting statistics at both generations
3) Calculate averages: $\bar{\tau}^E_{i,g}$ and $\bar{\tau}^N_{i,g}$
4) Repeat Steps 1–3 until maximum number of generations elapsed.

TABLE IV
SUMMARY OF EXPERIMENTAL RESULTS

Exp. ID	Cluster	Balancer	Load Type/Node	Runtime
A	I	None	None	2m 47s
B	I	Symmetric	None	2m 20s
C	I	Asymmetric	None	2m 12s
D	I	None	Program/Slow Node	11m 28s
E	I	Symmetric	Program/Slow Node	2m 32s
F	I	Asymmetric	Program/Slow Node	2m 13s
G	I	None	Network Download	3m 06s
H	I	Symmetric	Network Download	2m 26s
I	I	Asymmetric	Network Download	2m 11s
J	II	None	Unpredictable	37s
K	II	Symmetric	Unpredictable	30s
L	II	Asymmetric	Unpredictable	34s

V. Cluster Specifications

Experiments with the fuzzy load balancer were performed on two heterogenous clusters. Cluster I (Linux Laboratory, Dayalbagh Educational Institute) is a 15 node cluster comprising 10 IBM X206 servers (Pentium IV 3.2 GHz 512 MB RAM), 2 IBM ThinkCentres (Pentium IV 2.8 GHz 1GB RAM), 2 IBM NeVista Pentium III 1 GHz PCs (256 MB RAM), 1 Pentium PC (166 MHz 64MB RAM). All nodes run Red Hat Fedora Core 4/7 except for the slow Pentium node which runs Red Hat 8 Linux operating systems. LAM/MPI 7.1.1 was used on all nodes. Cluster II (UMIACS, University of Maryland, College Park, USA) comprises sixteen red nodes (Dual Pentium II 450MHz 1 GB RAM), twelve blue nodes (Dual Pentium III 550MHz 1GB RAM), and a single 8-way node called deathstar (Pentium III 550MHz 4GB RAM). Loading and network traffic on Cluster I are controllable; these parameters on Cluster II are not controllable due to administrative issues, and therefore experiments on Cluster II are blind in this sense.

VI. Experimental Results

A total of twelve experiments were performed using the two versions of the controller, two clusters, and various loading patterns. Table IV summarizes the essential results of these experiments. All experiments involve running the master-slave DE learning on ASuPFuNIS as specified in Table I, for 100 generations. Each experiment reported below was repeated for a minimum of four times. As given in the fuzzy load balancer algorithm outline, the string distribution to slaves was reviewed every two generations. This means that evaluation and communication delay statistics were collected for two generations, averaged, and then input to the fuzzy estimator to revise the string distribution to slaves for the next two generations.

A. No Load Balancer, No Load

For experiments A through F eight nodes (one master, seven slaves from Cluster I were employed (6 X206, 1 NetVista, 1 P-166). Experiment A ran master-slave DE learning on ASuPFuNIS for 100 generations assuming a total population size of 1400 strings distributed equally on to

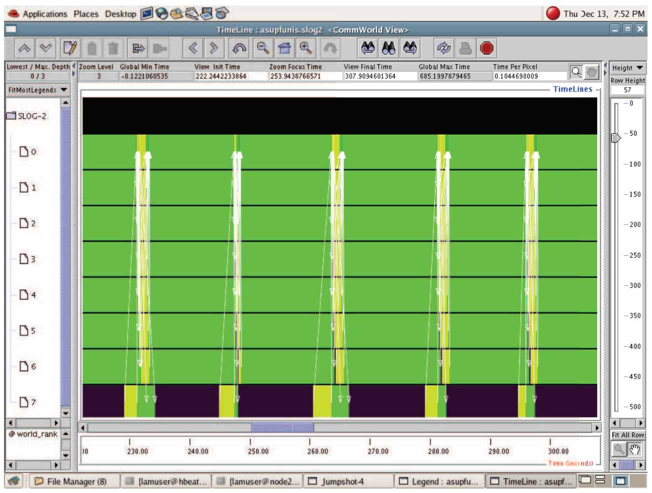

Fig. 4. JumpShot capture for program run with no load balancer, single node loaded.

seven nodes—each taking 200 string for evaluation. No load was applied to any of the nodes, which means that none of the slaves was running any other program. A run time of 2 minutes 47 seconds was observed.

B. Symmetric Load Balancer, No Load

All specifications were the same as in Experiment A, the fuzzy load balancer with the symmetric rule base (Table II) was employed. The program run time came down to 2 minutes and 20 seconds. The total speed up observed was 19.2%.

C. Asymmetric Load Balancer, No Load

All specifications were the same as in Experiment A, the fuzzy load balancer with the asymmetric rule base (Table III) was employed. The program run time came down to 2 minutes and 12 seconds. The total speed up observed was 26.5%.

D. No Load Balancer, Single Slow Node Loaded

All specifications were the same as in Experiment A, no fuzzy load balancer was employed, and the slow P-166 node was loaded by running the same parallel program with 5 threads on the same node. Much insight can be gained by observing the relative delays in communication and evaluation as portrayed in Fig. 4. The JumpShot capture shows the slow node at the bottom of the graph, with evaluation time shown in black. Evaluation times for other nodes are hardly visible given their much higher speeds. Figure 5 show the manner in which the slow node was loaded, and the variation of the corresponding evaluation time for 200 strings. The master continues to assign 200 strings to the slave even in the event of extra load on the slave. Fig. 4 shows the master has to wait for the slave to return the results, leading to long idle times on both the master as well as other faster slaves. The total run time for this simulation was 11 minutes 28 seconds.

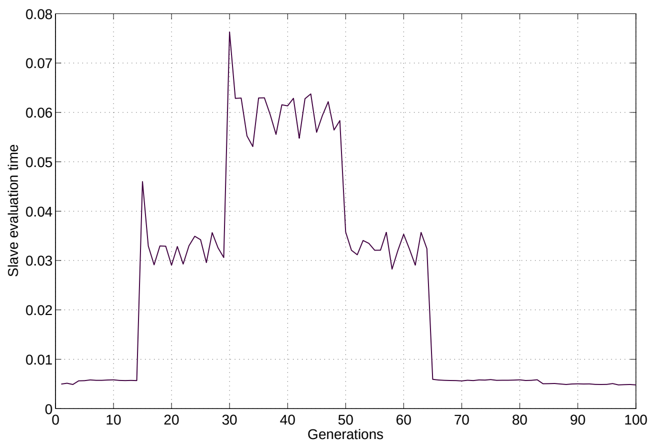

Fig. 5. Loading pattern for single slow node (P-166).

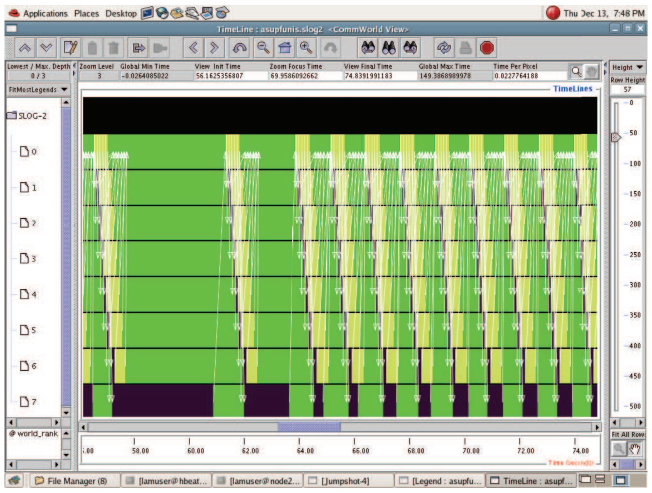

Fig. 6. JumpShot capture for symmetric load balancer with the load pattern of Fig. 5

E. Symmetric Load Balancer, Single Slow Node Loaded

The loading pattern was kept the same as in Experiment D, but with the symmetric load balancer running. Fig. 6 shows the JumpShot statistics for the first few generations. During the first two generations no load balancing takes place since the estimator is collecting statistics. Thereafter, differential assignment of strings starts taking place. It is clear, that the master and fast slave idle times have reduced considerably, leading to a drastic speed up bringing the total run time of the program down to 2 minutes 32 seconds. This represents a speed up of 352%. This reinforces the intuition that in heterogenous clusters with a mix of very fast and very slow nodes, load balancing is critical to obtain the best (smallest) run times of programs. The dynamic allocation of string to slaves is shown in Fig 7. The fuzzy estimator assigns about a 100 strings from the start of the simulation to the slow node, bringing this down to 67 strings when the loads are applied (see Table V).

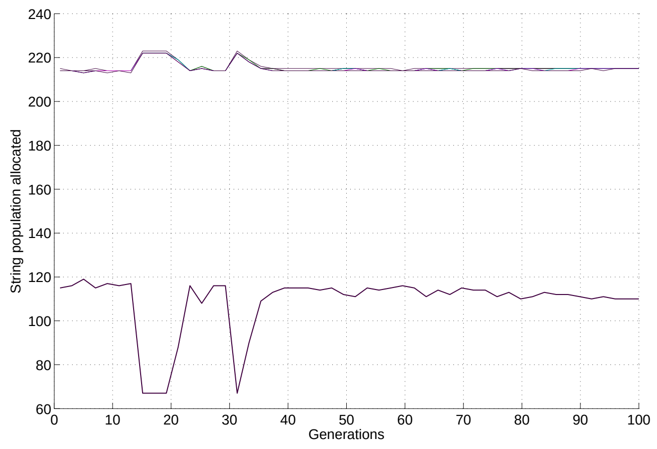

Fig. 7. Dynamic allocation of strings for symmetric load balancer.

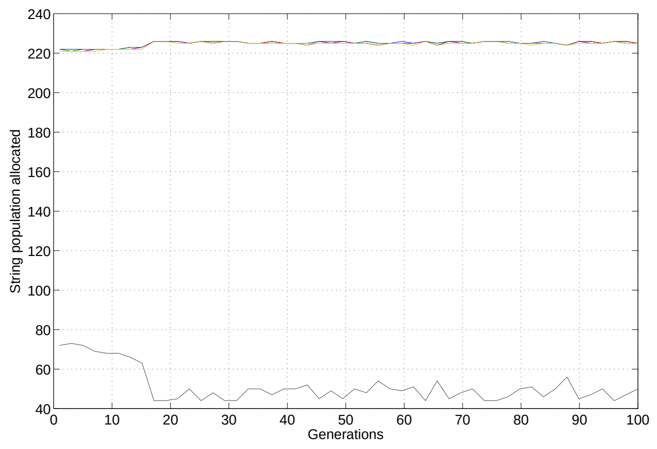

Fig. 8. Dynamic allocation of strings for asymmetric load balancer.

TABLE V
STRING ALLOCATION SNAPSHOT FOR EXPERIMENT E

Gen ↓ Slave ID →	1	2	3	4	5	6	7
12	214	214	214	214	214	214	116
14	214	214	214	214	214	213	117
16	223	222	222	222	222	222	67
18	223	222	222	222	222	222	67
20	223	222	222	222	222	222	67
22	219	219	219	219	218	218	88
24	214	214	214	214	214	214	116
26	216	216	215	215	215	215	108
28	214	214	214	214	214	214	116
30	214	214	214	214	214	214	116

F. *Asymmetric Load Balancer, Single Slow Node Loaded*

Conditions for this experiments were the same as in Experiment E, with the asymmetric load balancer being used in place of its symmetric counterpart. The total run time came down to 2 minutes 13 seconds, leading to similar speedups as in Experiment E. Fig. 8 shows how the dynamic estimator allocates strings to slaves. The allocations to the slow node are low from the beginning since this estimator is more sensitive to evaluation times than the symmetric version.

G. *No Load Balancer, Network Load on Single Node*

In this experiment, no load balancer was applied and two downloads were initiated on the IBM NetVista, each with an average download speed of 100 kBytes/s. The first download was started at Generation 15, and the second at Generation 30. The first ended at Generation 60, and the second at Generation 80. The total run time of the program was 3 minutes 6 seconds.

H. *Symmetric Load Balancer, Network Load on Single Node*

Experiment G was repeated with the symmetric load balancer. The run time of the program reduced to 2 minutes 26 seconds representing a speed up of 27%.

I. *Asymmetric Load Balancer, Network Load on Single Node*

Experiment G was repeated with the asymmetric load balancer. The run time of the program reduced to 2 minutes 11 seconds representing a speed up of 42%.

J. *No Load Balancer, Unpredictable Load on Cluster Nodes*

This experiment was run on Cluster II (UMIACS), using 15 nodes: 14 were red nodes, and 1 was blue (see cluster specifications given above). The total run time of the program was 37 seconds.

K. *Symmetric Load Balancer, Unpredictable Load on Cluster Nodes*

The program was run on Cluster II (UMIACS) with the same number of nodes. The total run time of the program was 30 seconds, representing a speed up of 23%.

L. *Asymmetric Load Balancer, Unpredictable Load on Cluster Nodes*

The program was run on Cluster II (UMIACS) with the same number of nodes. The total run time of the program was 34 seconds, representing a speed up of 8%. It is clear that the symmetric load balancer worked much better than the asymmetric version, which was governed by a biased rule base. The symmetric load balancer makes no such assumptions and assigns an equal priority to the slave communication delay and the slave evaluation time. In the event of a blind cluster, this is the best option to follow.

VII. CONCLUSIONS AND DISCUSSION

This paper introduces the idea of employing a simple fuzzy estimation system to predict the allocation of strings to slave nodes in a master-slave implementation of differential evolution learning as applied to a fuzzy-neural network: ASuPFuNIS. The experimental results presented in the previous section clearly demonstrate the effectiveness of the fuzzy logic load estimator in the presence of both CPU and network loads. Its effectiveness on the UMIACS cluster over which we have no administrative control also shows that the estimation mechanism is very effective. It is worth emphasizing that the ideas presented herein are very general, and can be applied to load balancing in evolutionary learning systems in general.

One important fact is that the rule base of the controller can be made adaptive so that the fuzzy set parameters are learnt by the system over time. This will have the advantage that one can start out with a raw estimator, which fine tunes its internal parameters in accordance with the nature of the cluster—both in terms of the node performances, and the communication delays experienced. This is important, for as we have observed, in Cluster I the asymmetric rule base gave better performance, whereas in Cluster II the symmetric rule base performed better. Both these rule bases were based on heuristics, rather than hard numeric information. We believe that adaptive principles should be incorporated into the estimator in order to make it even more effective. Use moving averages instead of simple averages is another matter for consideration. These issues are presently under investigation.

Acknowledgment

This work was supported by the Department of Science and Technology, Government of India, under Research Grant No. DST SR/S3/EECE/65/2004-ET. The authors gratefully acknowledge the assistance of Prof. Ashok K. Agrawala, Director MIND Lab., University of Maryland, College Park, USA, in providing access to the UMIACS Cluster Facility, under the MoU with Dayalbagh Educational Institute.

References

[1] E. Alba and M. Tomassini, “Parallelism and evolutionary algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 443–462, October 2002.

[2] V. S. Sunderam, “PVM: A framework for parallel distributed computing,” *Jnl. of Concurr. Practice and Experience*, vol. 2, no. 4, pp. 315–339, 1990.

[3] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI—The Complete Reference*. Cambridge, MA: MPI Press, 1998.

[4] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*. Cambridge, MA: MIT Press, 2nd ed., 1999.

[5] I. Foster and C. Kesselmann, “Globus: A metacomputing infrastructure toolkit,” *Jnl. Supercomput. Applic.*, vol. 11, no. 2, pp. 115–128, 1997.

[6] K. C. Tan, A. Tay, and J. Cai, “Design and implementation of a distributed evolutionary computing software,” *IEEE Trans. on Systems, Man and Cybernetics—Part C: Applications and Reviews*, vol. 33, pp. 325–338, August 2003.

[7] F. Seredynski, “Dynamic mapping and load balancing with parallel genetic algorithms,” in *IEEE World Congress on Computational Intelligence*, pp. 834–839, 1994.

[8] S. Genaud, A. Giersch, and F. Vivien, “Load-alancing scatter operations for grid computing,” *Parallel Computing*, vol. 30, pp. 923–946, July 2004.

[9] G. Folino, C. Pizzuti, and G. Spezzano, “A scalable cellular implementation of parallel genetic programming,” *IEEE Trans. on Evolutionary Computation*, vol. 7, pp. 37–53, February 2003.

[10] L. Xiao, S. Chen, and X. Zhang, “Adaptive memory allocations in clusters to handle unexpectedly large data intesive jobs,” *IEEE Trans. on Parallel and Distributed Systems*, vol. 15, pp. 577–592, July 2004.

[11] J. H. Abawajy and S. P. Dandamundi, “Parallel job scheduling on multicluster computing systems,” in *Proceedings of the IEEE International Conference on Cluster Computing(CLUSTER’03)*, IEEE Computer Society, 2003.

[12] H. Liang, M. Faner, and H. Ming, “A dynamic load balancing system based on data migration,” in *The 8th International Conference on Computer Supported Cooperative Work in Design*, pp. 493–499, IEEE, 2003.

[13] J. Bahi, R. Couturier, and F. Vernier, “Synchronous distributed load balancing on dynamic networks,” *Journal of Parallel and Distributed Computing*, vol. 65, pp. 1397–1405, 2005.

[14] S. M. Lau, Q. Lu, and K. S. Leung, “Adaptive load distribution for heterogeneous distributed systems with multiple task classes,” *Journal of Parallel and Distributed Computing*, vol. 66, pp. 163–180, 2006.

[15] L. S. Cheung, “A fuzzy approach to load balancing in distributed object computing network,” in *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 694–699, IEEE, 2001.

[16] Y. K. Kwok and L. S. Cheung, “A new fuzzy-decision based load balancing system for distributed object computing,” *Journal of Parallel and Distributed Computing*, vol. 64, pp. 238–253, 2004.

[17] C. S. Velayutham, S. Paul, and S. Kumar, “Asymmetric subsethood product fuzzy neural network (ASuPFuNIS),” *IEEE Transactions on Neural Networks*, vol. 16, pp. 160–174, January 2005.

[18] R. Storn and K. Price, “Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces,” *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[19] K. Price, *New Ideas in Optimization*, ch. An Introduction to Differential Evolution, pp. 79–108. Cambridge, U.K.: McGraw Hill, 1999.

[20] L. Singh and S. Kumar, “Parallel evolutionary asymetric subsethood product fuzzy-neural inference system with applications,” in *2006 IEEE International Conference on Fuzzy Systems*, pp. 8517–8524, July 2006.