Improving Accuracy of Host Load Predictions on Computational Grids by Artificial Neural Networks

Truong Vinh Truong Duy[1] Yukinori Sato[2] Yasushi Inoguchi[2]
[1]*Graduate School of Information Science,*
[2]*Center for Information Science,*
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa, 923-1292 Japan
[duytvt, yukinori, inoguchi]@jaist.ac.jp

Abstract

The capability to predict the host load of a system is significant for computational grids to make efficient use of shared resources. This paper attempts to improve the accuracy of host load predictions by applying a neural network predictor to reach the goal of best performance and load balance. We describe feasibility of the proposed predictor in a dynamic environment, and perform experimental evaluation using collected load traces. The results show that the neural network achieves a consistent performance improvement with surprisingly low overhead. Compared with the best previously proposed method, the typical 20:10:1 network reduces the mean and standard deviation of the prediction errors by approximately 60% and 70%, respectively. The training and testing time is extremely low, as this network needs only a couple of seconds to be trained with more than 100,000 samples in order to make tens of thousands of accurate predictions within just a second.

1. Introduction

Grid computing [1] is designed to meet the needs of performing large numbers of complex computations by aggregating heterogeneous resources located in different places over a network using open standards. In order to achieve the best performance in such an open and highly dynamic computing environment, efficient task scheduling is essential to choose which collection of distributed resources to use [2,3]. Also, by using mechanisms such as CORBA [4], Java RMI [5], and emerging GridRPC [6], a task can be scheduled to execute on any of the Grid nodes. Obviously, choosing a node would become much easier if the scheduler could know the task's running time on the nodes beforehand.

In fact, the running time of a task, which varies as a result of CPU availability, is directly related to the average host load. The running time was pointed out to be almost linear to the correspondingly measured host load during execution [7]. As a result, the running time can be determined by predicting the host load of the system.

Fortunately, the host load is discovered to be consistently predictable to a very useful degree from historical data and self-similarity [8]. There have been many efforts to make reliable host load predictions from load history, ranging from traditional linear models [9,10] to recently proposed tendency-based models [11,12]. Nonetheless, such predictions are unlikely to be accurate due to their limitations in capturing all the underlying features of the host load history and the dynamic nature of Grid computing systems.

In this paper, we aim to address the following three issues by applying artificial neural networks to the task of host load prediction. First, we discuss whether neural networks produce more accurate results than previously proposed linear models and tendency-based models in this context. Second, we consider the cost, namely the cost for training, validating and testing, to generate such good predictions. Finally, and most importantly, we examine if an artificial neural network based solution is applicable in a dynamic real-time setting, such as computational grids facing a tradeoff between benefits and expenses.

We perform experimental evaluation and show that the neural network predictor achieves a consistent improvement over the competitors in predicting future load values. Once being trained for only a couple of

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

seconds with all the load traces collected over 10 days, the simple 20:10:1 network is able to predict load values over the next 10 days with very low mean prediction errors, and without the need of being trained again. The time required for producing thousands of predicted load values is also just within a second, almost the same as other methods. The results strongly support the feasibility of bringing a neural predictor to real world scheduling systems to exploit its accurate prediction ability with surprisingly low overhead.

The remaining parts of this paper are organized as follows. Section 2 introduces background and related work. The neural network predictor is detailed in Section 3. Section 4 analyzes experimental results when our neural predictor is applied to actual measurements and compared to results of other previous work. Finally, we conclude our study in Section 5.

2. Related work

Perhaps the most influential research on prediction-based real-time systems for distributed interactive applications is the work by P. Dinda et al. [7,8]. In [7], the authors improved understanding of how host load changes over time by collecting the traces of the Digital Unix 5 second exponential load average on over 35 different machines. By analyzing the traces, they found that load exhibits a high degree of self-similarity, with Hurst parameters ranging from 0.73 to 0.99, and that load displays epochal behavior, with the local frequency content of the load signal remaining quite stable for long periods.

Before long they presented a study on the performance of different linear models for host load prediction based on these load traces [8]. Multiple linear models, including AR, MA, ARMA, ARIMA, and ARFIMA models were rigorously evaluated. The main conclusions are that load is consistently predictable to a very useful degree, and that the simple AR model is the best model of this class, due to its relatively good prediction ability and low overhead. However, these linear models themselves are limited, and may not be able to capture some kinds of nonlinear behavior in the host loads.

Another more accurate approach is based on tendency-based prediction techniques [11,12]. Generally, they assume that if the current value increases, the next value will also increase, and vice versa. In [11], Yang et al. proposed a number of one-step-ahead prediction strategies which give more weight to more recent measurements than to other historical data, while paying attention to different behaviors when “ascending” and “descending”. One better strategy based on the tendency with several steps backward was introduced by Zhang et al. in [12], using polynomial fitting method to produce the prediction values. Although these models generally perform well, they have a glaring error source, committing great errors when the time series changes its direction.

We believe artificial neural networks are able to overcome those limitations. They have many important advantages over the traditional statistical models, most notably nonlinear generalization ability [13]. With this remarkable ability, they can learn from data examples and capture the underlying functional relationships between input and output values. Neural networks have been applied to modeling nonlinear time series in various areas, for example, stock market [14], sports results [15], road surface temperature [16] seasonal time series [17], quarterly time series [18] and even scheduling problems [19].

In [19], a NARX neural network based load prediction was presented to define data mappings appropriate for dynamic resources with the aim of improving the scheduling decision in grid environments. A major difference between this approach and our approach is that it utilizes a recurrent network while our method employs feedforward networks for fast convergence. Also, their work merely focused on scheduling performance, in particular the execution time of application running in the proposed scheduling method. Not only do we improve accuracy of load predictions, but we also consider the cost to examine if such a neural network based solution is feasible in dynamic environments. Lastly, their experiments were simply carried out with only one network architecture, a constant learning rate of 0.2, and 20 minutes of the load trace beside our combinations of different architectures and learning rates with 4 load traces collected over tens of days.

3. Host load prediction with artificial neural networks

3.1. An overview of ANNs

Artificial neural network, originally developed to mimic biological neural networks, is a computational model which is composed of a large number of highly interconnected simple processing elements called neurons or nodes. Each node receives input signals generated from other nodes or external inputs, processes them locally through an activation function, and produces an output signal to other nodes or external outputs. Given a training set of data, the ANN can learn the data with a learning algorithm, and forms

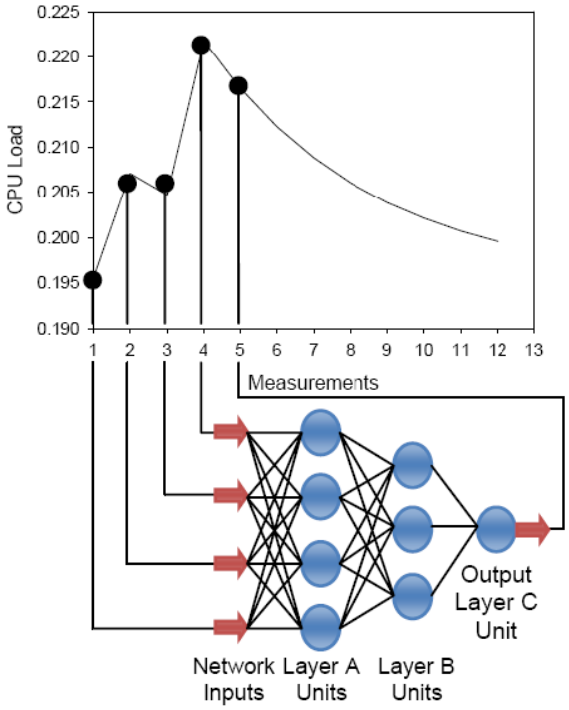

Figure 1. A three-layer feedforward network

a mapping between inputs and desired outputs from the training set through learning. After the learning process has finished, it is able to catch the hidden dependencies between the inputs and outputs and generalize to data never before seen.

Among many different ANN types, the multi-layer feedforward network, Hopfield network, and Kohonen self-organizing network are probably the most important ones. A multi-layer feedforward network consists of any number of layers, nodes per layer, network inputs and network outputs. In this network, the information moves sequentially forward in only one direction from the inputs to the outputs. Hopfield network is a recurrent neural network in which all connections are symmetric. This network guarantees that its dynamics will converge. Kohonen network is motivated by the self-organizing behavior of the human brain. A set of artificial neurons learn to map points in an input space to coordinates in an output space.

In this study, the multi-layer feedforward network [20], accompanied by backpropagation, which is the most widely used training algorithm for multi-layer networks, is chosen for predicting the host loads. It has been applied in a variety of problems, especially in the field of prediction, because of its inherent capability of arbitrary input–output mapping. More importantly, simplicity and fast convergence ability make it feasible

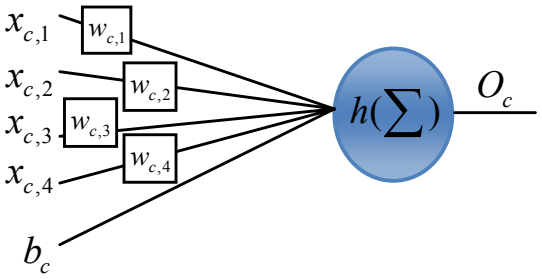

Figure 2. Node with its inputs, weights and bias

for deployment in a real-time dynamic setting. Also, the availability of host load traces meets the needs of the multi-layer feedforward network for input and output data.

3.2. Feedforward neural networks

Figure 1 depicts an example feedforward neural network in operation with a host load time series. The network has 4 network inputs where external information is received, and 1 output layer C with one node where the solution is obtained. The network inputs and output layer are separated by 2 hidden layers: layer A with 4 nodes and layer B with 3 nodes. The connections between the nodes indicate the flow of information from one node to the next, i.e., from left to right.

Each node has the same number of inputs as the number of units in the preceding layer. As shown in Figure 2, each connection is modified by a weight, and each node has an extra input assumed to have a constant value of 1. The weight that modifies this extra input is called the bias.

When the network is run, each layer node performs the calculation in Equation 1 on its input, and transfers the result O_c to the next layer.

$$O_c = h\left(\sum_{i=1}^{n} x_{c,i} w_{c,i} + b_c\right)$$
$$where\ h(x) = \begin{cases} \dfrac{1}{1+e^{-x}} & if\ hidden\ layer\ node \\ x & if\ output\ layer\ node \end{cases} \quad (1)$$

where O_c is the output of the current node, n is the number of nodes in the previous layer, $x_{c,i}$ is an input to the current node from the previous layer, $w_{c,i}$ is the weight modifying the corresponding connection from $x_{c,i}$, and b_c is the bias. In addition, h(x) is either a sigmoid activation function for hidden layer nodes, or a linear activation function for the output layer nodes.

3.3. Backpropagation training

In order to make meaningful predictions, the neural network needs to be trained on an appropriate data set. Basically, training is a process of determining the connection weights in the network. Examples of the training data set are in the form of <input vector, output vector> where input vector and output vector are equal in size to the number of network inputs and outputs, respectively. Then, for each example, the backpropagation training process involves the following steps.

Step 1: An input vector is fed to the network inputs and the network is run: Equation 1 is computed sequentially forward from left to right until eventually the network output activation values are found.

Step 2: The error terms for the output layer nodes, i.e., the difference between the desired output D_c (the output vector) and the actual network output O_c, is computed with Equation 2.

$$\delta_c = D_c - O_c \tag{2}$$

Then these errors are propagated sequentially backward from right to left to calculate errors for hidden layer nodes based on Equation 3.

$$\delta_c = O_c(1-O_c)\sum_{i=1}^{n}\delta_i w_{i,c} \tag{3}$$

where O_c is the output of the current hidden layer node, n is the number of nodes in the next layer, δ_i is the error for a node in the next layer and $w_{i,c}$ is the weight modifying the corresponding connection from the current node to that node.

Step 3: For each connection, the change in the weight modifying the connection from node c to node p is computed using Equation 4 and added to the weight.

$$\Delta w_{c,p} = \alpha \delta_c O_p \tag{4}$$

where α is the learning rate of the network, δ_c is the error of node c and O_p is the output of node p. The learning rate controls how quickly and how finely the network converges to a solution. The network is trained with different learning rate values ranging from 0.01 to 0.3 in our experimentation.

After the training process has been performed for every example of the training data set, one epoch occurs. The final goal is to find the weights that minimize some overall error measure such as the sum of squared errors or mean squared errors. We evaluate the accuracy of our predictions using the widely used normalized mean squared errors defined as follows:

$$NMSE = \frac{1}{\sigma^2 N}\sum_{i=1}^{N}(O_i - D_i)^2 \tag{5}$$

where σ^2 is the variance of the time series in the period with N examples, and O_i and D_i are, respectively, the predicted and the desired outputs at time i.

4. Experimental evaluation

4.1. Input parameters

A neural predictor was developed with the aim of evaluating performance of neural network in host load prediction. Even though there are many neural network applications freely and commercially available, we developed our own program to customize and extend various features and parameters using Microsoft Visual Studio C++ 6.0. Moreover, we believe that developing is the best way to gain a deep understanding of its internal operation.

A number of experiments were conducted with several input parameters. A challenging parameter which must be identified before running is the appropriate network architecture, i.e., the number of hidden layers, and the number of nodes for each layer. Unfortunately, there is no rule for choosing the exact numbers in neural networks, as they are more art than science! However, through observation and trial-and-error we found that two hidden layers are sufficient to perform accurate predictions with a couple of dozens of inputs within a reasonable time. The number of output layer nodes is the easiest, since it is necessarily one. As a result, the networks that were tested include 20:10:1, 30:10:1, 50:20:1 and 60:30:1. Another important parameter is the value of learning rate. In the experiments, we evaluated each of these networks with the learning rates valued at 0.01, 0.05, 0.1, 0.2, and 0.3.

Once the network parameters have been determined, we are able to feed the data series to the networks for evaluation. The time series we chose here are the 4 most interesting load traces: axp0, axp7, sahara, and themis in a number of load traces on Unix systems collected by Dinda [23]. The load is the number of processes that are running or are ready to run, which is the length of the ready queue maintained by the scheduler. The kernel samples the length of the ready queue at some rate, and averages some number of previous samples to produce a load average which can be captured by a user program. These 4 load traces represent diversity both in capture periods and machine types, as displayed in Table 1.

Table 1. Descriptions of 4 load traces

Name	Description	Collected load traces	Mean	Standard Deviation
axp0	heavily loaded, highly variable interactive machine	1,296,000 (in 75 days)	1	0.54
axp7	lightly loaded batch machine	1,123,200 (in 65 days)	0.12	0.14
sahara	moderately loaded, big memory computing server	345,600 (in 20 days)	0.22	0.33
themis	moderately loaded desktop machine	345,600 (in 20 days)	0.49	0.5

In addition, the load traces have to be normalized to a suitable range of values because the sigmoid activation function h(x) has output of the range [0, 1]. Also, normalization tends to make the training process better by improving the numerical condition and ensuring that various default values involved in initialization and termination are appropriate. There are several different ways to normalize the network inputs well suited to different ranges of input values and applications. By observing the means and standard deviations of the load traces, we confirm that all of the load traces are positive and most of them are distributed over an interval of [0, 1]. Hence, rescaling these values to a range of [0.1, 0.9] is thought to maintain the original conditions and does not discard much information. The following normalization formula is applied to every x in each load trace.

$$x_i = LWB + \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}(UPB - LWB) \qquad (6)$$

where x_{max} and x_{min} are the maximum and minimum value of each load trace respectively, and LWB is the lower bound and UPB is the upper bound of the interval [0.1, 0.9]. The 4 normalized load traces are presented in Figure 3.

Finally, each normalized load trace is divided into 3 sets: learning set, validating set and testing set, and each accounts for a percentage of 50%, 30%, and 20% of the total number of traces, respectively, as detailed in Table 2. The learning set is fed to the neural networks to fit their connection weights during the learning process. The normalized mean squared error is measured using the validating set and used to validate whether to finish the learning process or not. If the current error is 20% higher than the minimum error, the training process is stopped and weights are restored to the previous values. The last set, the testing

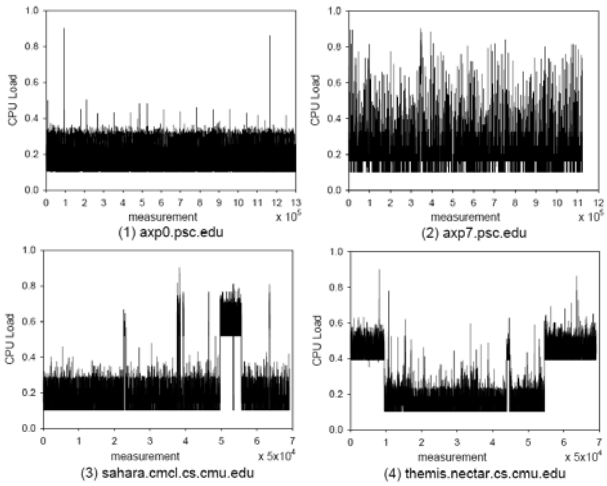

Figure 3. The normalized load traces

set, is then used for assessing the performance of the neural networks.

Table 2. Load trace partitioning

Name	Training set	Validating set	Testing set	Total
axp0	648,000	388,800	259,200	1,296,000
axp7	561,600	336,960	224,640	1,123,200
sahara	172,800	103,680	69,120	345,600
themis	172,800	103,680	69,120	345,600

4.2. Host load prediction results

Figure 4 shows the experimental results. The y-axis represents the normalized mean squared error (NMSE), Mean and standard deviation (SD) of the prediction errors for those 4 testing sets with 4 different network architectures, and x-axis represents their learning rate. The prediction error for each load value in the testing set is calculated using the following equation:

$$\text{Prediction error} = \frac{1}{N}\sum_{i=1}^{N}\frac{|O_i - D_i|}{D_i} * 100\% \qquad (7)$$

where N is the number of load values in the testing set, and O_i and D_i are the predicted and the actual values at the i^{th} trace respectively.

We find that no combination of learning rate and network architecture is the best for all the load traces,

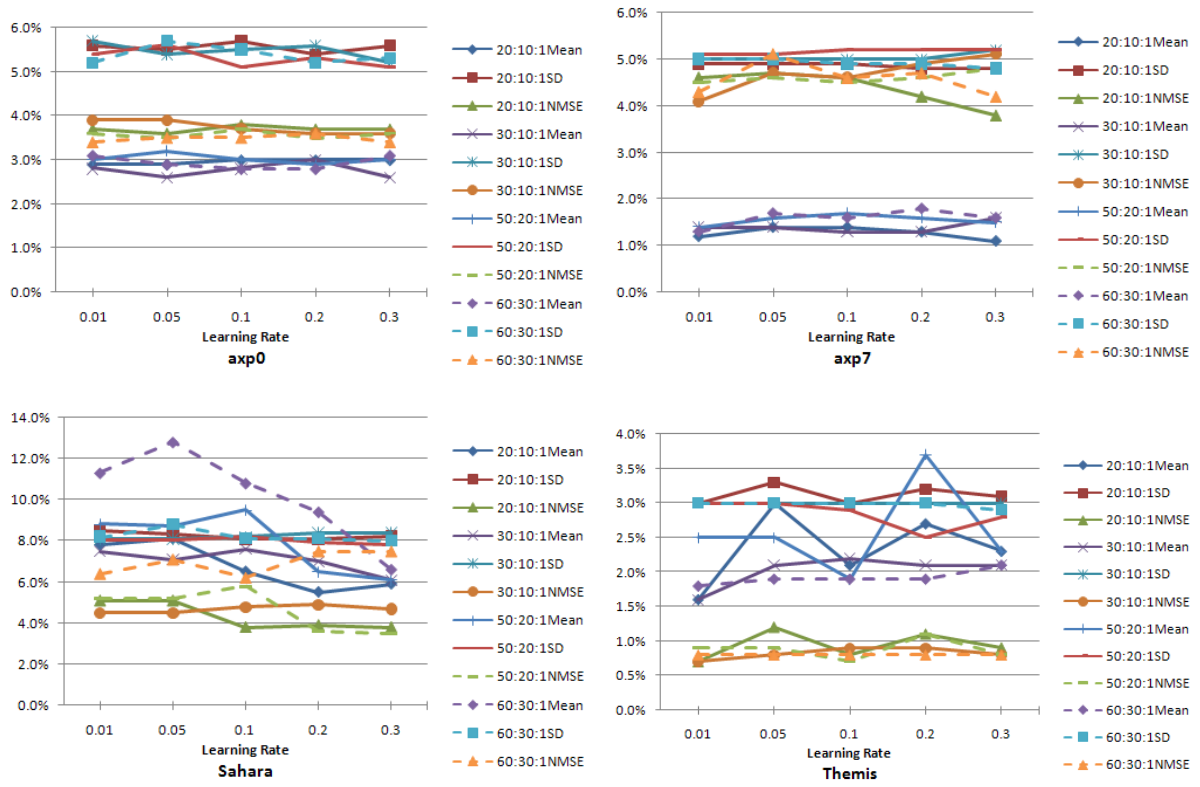

Figure 4. Mean, standard deviation and normalized mean squared error

as the values of the Mean, SD and NMSE keep changing with the changes in learning rate and network architecture. For example, NMSE has values in the range [3.4%, 3.9%] in axp0, [3.8%, 5.1%] in axp7, [3.5%, 7.5%] in sahara, and [0.7%, 1.2%] in themis. The Mean has values in the range [2.6%, 3.2%] in axp0, [1.1%, 1.8%] in axp7, [5.5%, 12.8%] in sahara, and [1.6%, 3.7%] in themis. Likewise, SD has values in the range [5.1%, 5.7%] in axp0, [4.8%, 5.2%] in axp7, [7.8%, 8.5%] in sahara, and [2.5%, 3.3%] in themis. Such low values of these error terms imply that the neural network has successfully captured hidden behavior of the host load. The choices of learning rate and network architecture, however, do not affect the variation, since it is quite small.

The learning rate and network architecture have a significant impact on the networks' training time, as shown in Table 3. In the same training set and the same network architecture, the lowest learning rate of 0.01 always results in the longest times, while the highest learning rate of 0.3 is involved in most cases with the shortest times. Similarly, it is easy to see that with the same training set and learning rate, there is a connection between most cases with the longest times

Table 3. Training time

Name	Learning Rate				
	0.01	0.05	0.1	0.2	0.3
20:10:1axp0	725681	68548	49436	26880	18641
30:10:1axp0	589741	89564	48653	65483	15867
20:10:1axp7	163102	9510	7402	2490	2492
30:10:1axp7	67728	9657	5756	5363	578
50:20:1axp7	218219	8182	1238	1199	1203
60:30:1axp7	235868	6408	6420	4181	4185
20:10:1sahara	80	20	30	14	10
30:10:1sahara	183	43	26	14	9
50:20:1sahara	248	56	42	109	68
60:30:1sahara	223	48	48	71	71
20:10:1themis	16752	56	57	6	7
30:10:1themis	21431	95	56	14	14
50:20:1themis	1036	207	207	235	82
60:30:1themis	130086	1027	1033	464	218

and the largest network architecture of 60:30:1. In contrast, the smallest network architecture, 20:10:1, proves to contribute substantially to the time reduction in most cases linked to the shortest times, when the learning rate is kept unchanged in the same training set. Hence, a mixture of the learning rate of 0.3 and network architecture of 20:10:1 can be the most efficient choice in the sense of training time.

We also note that the size of training set obviously has a strong effect on the time needed for training the networks. The training time sharply increases from only a couple of seconds to several days as the size of the training set is increased. For example, the size of training set ranges from 172,800 (sahara and themis) to 648,000 (axp0), as shown in Table 2. Selection of the training set size is a tradeoff between accuracy and time; a training set which is too large takes the network a long time to complete, while a too small set is probably inadequate to get a complete picture of the load traces. The outcome suggests that a training size of about 100,000, equivalent to the number of loads collected in 5 days, is reasonable to make it possible for the network to quickly respond with correct solutions in a dynamic environment.

Unlike the training time, which heavily depends on the network architecture and learning rate, the validating and testing time behaves in a different way and remains fairly stable. Figure 5 demonstrates the average validating and testing time for each load trace in proportion to the network architecture, since the learning rate has almost no effect. The bigger the data set is, the longer the validating and testing time is. The lines representing sahara and themis nearly overlap each other owing to the same size data set. In the same set, larger network architecture causes a larger increase in the validating and testing time. However, in general this time is trivial compared with the training time. In the case of sahara and themis, for instance, the predicted results for about 70,000 future load values can be obtained in just a second.

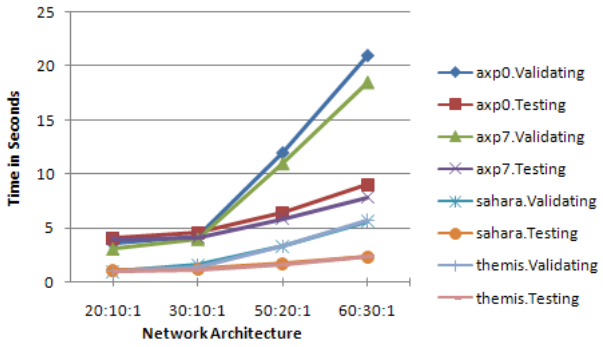

Figure 5. Validating and testing time

4.3. Performance comparison

We compare performance of the proposed neural predictor with that of two previously proposed prediction methods in terms of Mean and standard deviation of the prediction errors. We selected the networks 20:10:1 and 30:10:1, both with the same learning rate of 0.3. The following results were taken from previously published papers. One method (Yang et al.) is a tendency-based strategy which has been proven to outperform the widely used NWS method [11]. The other (Zhang et al.) is also based on tendency observation of the load traces proposed by Zhang et al. in [12]. At the time of writing, this method remains the best performer in host load prediction.

Table 4 shows the results of performance comparison. We find that the neural network has achieved consistent improvement over the competitors in predicting future load values of all the load traces. In the case of axp7 where both method 1 and 2 are especially effective, the 20:10:1 network still outdoes them, reducing the mean by approximately 60% and standard deviation by 70% compared to method 2. The smallest difference in the mean between this network and method 2, about 30%, is in the case of sahara. In other load traces, the reduction ratio is much higher, from 3 times (axp0) to 5 times (axp7).

Table 4. Performance comparison

METHOD		axp0	axp7	sahara	themis
Yang et al. [11]	Mean (%)	18.99	15.02	18.28	27.37
	SD	0.44	0.38	0.37	0.48
Zhang et al. [12]	Mean (%)	10.57	2.73	9.2	10.6
	SD	0.37	0.08	0.23	0.25
20:10:1 Network	Mean (%)	3.00	1.11	5.95	2.34
	SD	0.05	0.05	0.08	0.03
30:10:1 Network	Mean (%)	3.15	1.6	6.12	2.18
	SD	0.05	0.05	0.08	0.03

5. Conclusion

In this paper, we have studied the performance and cost of a neural network predictor by applying it to load trace analysis in the area of host load prediction. Experimental evaluation has shown that the neural network consistently improves performance compared with previously proposed linear models and tendency-based models in all the load traces. The 20:10:1 network with a learning rate of 0.3 outperforms the method which remains the best at the time of writing, by reducing the mean by approximately 60% and standard deviation by 70%. The cost, particularly the training time, is extremely low, as this network needs only a few seconds to be trained with more than 100,000 samples, and then makes tens of thousands of accurate predictions within a second.

The prediction ability and low overhead convincingly demonstrate application feasibility of the neural predictor in dynamic computing environments such as computational grids. Inspired by these positive signals, we are now in the process of integrating the neural predictor into a real-time scheduler in a GridRPC computing system using NetSolve and GridSolve. We also plan to show efficiency in other areas beyond the area of host load predictions, for instance, in relation to network traffic and latency, etc.

References

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1999.

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks", *Supercomputing'96*, 1996.

[3] S. Jang, X. Wu, V. Taylor, "Using Performance Prediction to Allocate Grid Resources", *GriPhyN Technical Report 2004-25*, 2004, pp. 1-11.

[4] A. S. Gokhale, and B. Natarajan, "GriT: a CORBA-based grid middleware architecture", *Proc. of the 36th Annual Hawaii International Conference on System Sciences,* 2003, pp. 1-10.

[5] M. Alt, and S. Gorlatch, "Adapting Java RMI for grid computing", *Parallel computing technologies,* Vol. 21, Issue 5, 2005, pp. 699 – 707.

[6] Y. Tanaka, H. Takemiya, H. Nakada, and S. Sekiguchi, "Design, implementation and performance evaluation of GridRPC programming middleware for a large-scale computational Grid", *Proc. of 5th IEEE/ACM International Workshop on Grid Computing*, 2004.

[7] P. Dinda, "The Statistical Properties of Host Load", *Scientific Programming*, 7:3-4, 1999.

[8] P. Dinda, and D. O'Hallaron, "Host Load Prediction Using Linear Models", *Cluster Computing*, Volume 3, Number 4, 2000.

[9] P. Dinda, "A prediction-based real-time scheduling advisor", *Proc. of 16th International Parallel and Distributed Processing Symposium*, 2002, pp. 35-42.

[10] R. Wolski, "Dynamically forecasting network performance using the network weather service," *Journal of Cluster Computing*, Vol.1, pp.119-132, 1998.

[11] L. Yang, I. Foster, and J. Schopf, "Homeostatic and Tendency-based CPU Load Prediction", *Proc. of International Parallel and Distributed Processing Symposium*, 2003, pp. 42-50.

[12] Y. Zhang, W. Sun, and Y. Inoguchi, "CPU Load Predictions on the Computational Grid", *Proc. of IEEE International Symposium on Cluster Computing and the Grid*, 2006, pp. 321-326.

[13] G. Zhang, B.E. Patuwo, and M.Y. Hu, "Forecasting with artificial neural networks: The state of the art", *International Journal of Forecasting* 14(1), 1998, pp. 35-62.

[14] B. W. Wah, and M. L. Qian, "Constrained formulations and algorithms for predicting stock prices by recurrent FIR neural networks", *International Journal of Information Technology & Decision Making*, Vol.5, No. 4, 2006, pp. 639-658.

[15] B. G. Aslan, and M. M. Inceoglu, "A Comparative Study on Neural Network based Soccer Result Prediction", *Proc. of Seventh International Conference on Intelligent Systems Design and Applications*, 2007.

[16] F. Liping, H. Behzad, F. Yumei, and K. Valeri, "Forecasting of Road Surface Temperature Using Time Series, Artificial Neural Networks, and Linear Regression Models", *Proc. of Transportation Research Board 87th Annual Meeting*, 2008.

[17] S. F. Crone, and R. Dhawan, "Forecasting Seasonal Time Series with Neural Networks: A Sensitivity Analysis of Architecture Parameters", *Proc. of International Joint Conference on Neural Networks*, 2007, pp. 2099-2104.

[18] G. P. Zhang, and D. M. Kline, "Quarterly time-series forecasting with neural networks", *IEEE transactions on neural networks*, Vol. 17, No. 6, 2007, pp. 1800-1814.

[19] J. Huang, H. Jin, X. Xie, and Q. Zhang, "Using NARX Neural Network based Load Prediction to Improve Scheduling Decision in Grid Environments", *Proc. of Third International Conference on Natural Computation*, 2007.

[20] Reed, R. D., and R. J. Marks, *Neural Smithing*, The MIT Press, 1999.

[21] Husmeier, D., *Neural Networks for Conditional Probability Estimation*, Springer, 1999.

[22] Witten, I. H., and E. Frank, *Data Mining – Practical Machine Learning Tools and Techniques*, 2nd Edition, MK Publishers, 2005.

[23] Load Traces Archive, http://www.cs.northwestern.edu/~pdinda/LoadTraces/