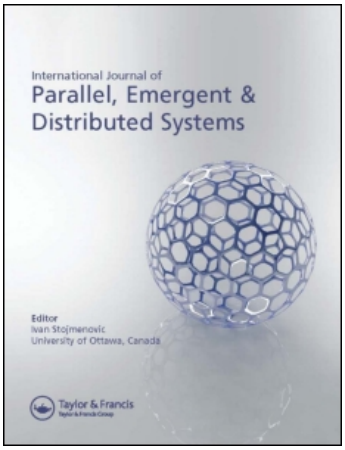

International Journal of Parallel, Emergent and Distributed Systems
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713729127

Improving accuracy of host load predictions on computational grids by artificial neural networks

Truong Vinh Truong Duy[a]; Yukinori Sato[b]; Yasushi Inoguchi[b]
[a] School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa, Japan [b] Center for Information Science, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa, Japan

First published on: 09 August 2010

To cite this Article Duy, Truong Vinh Truong , Sato, Yukinori and Inoguchi, Yasushi(2010) 'Improving accuracy of host load predictions on computational grids by artificial neural networks', International Journal of Parallel, Emergent and Distributed Systems,, First published on: 09 August 2010 (iFirst)
To link to this Article: DOI: 10.1080/17445760.2010.481786
URL: http://dx.doi.org/10.1080/17445760.2010.481786

International Journal of Parallel, Emergent and Distributed Systems
iFirst article, 2010, 1–16

Improving accuracy of host load predictions on computational grids by artificial neural networks

Truong Vinh Truong Duy[a]*, Yukinori Sato[b] and Yasushi Inoguchi[b]

[a]*School of Information Science, Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan;* [b]*Center for Information Science, Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan*

(*Received 20 July 2009; final version received 20 February 2010*)

The capability to predict the host load of a system is significant for computational grids to make efficient use of shared resources. This work 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 the 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 consistent performance improvement with surprisingly low overhead in most cases. Compared with the best previously proposed method, our typical 20:10:1 network reduces the mean of prediction errors by approximately up to 79%. 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.

Keywords: host load; neural networks; predictor; grid computing; scheduling

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 in choosing 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 found to be almost linearly correlated with 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

*Corresponding author. Email: duytvt@jaist.ac.jp

ISSN 1744-5760 print/ISSN 1744-5779 online

DOI: 10.1080/17445760.2010.481786
http://www.informaworld.com

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 work, we aim to address the following three issues by applying artificial neural networks (ANNs) 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 whether an ANN-based solution is applicable in dynamic real-time settings, such as computational grids facing trade-offs between benefits and expenses.

We perform experimental evaluation and show that the neural network predictor achieves consistent improvement over the competitors in predicting future load values. After being trained for only a couple of seconds with all the load traces collected over 10 days, our 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 work are organised as follows. Section 2 introduces background and related work. The neural network predictor is detailed in Section 3. Section 4 analyses 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 Dinda et al. [7,8]. In [7], the authors improved the understanding of how host load changes over time by collecting the traces of Digital Unix, which uses a 5-s exponential average, on over 35 different machines. By analysing the traces, they found that loads exhibit a high degree of self-similarity, with Hurst parameters ranging from 0.73 to 0.99, and that loads display epochal behaviour, with the local frequency content of the load signal remaining quite stable for long periods.

After that 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 were 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 behaviour in the host loads.

Another more accurate approach uses tendency-based prediction techniques [11,12]. Generally, these techniques 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 that give more weight to more recent measurements than to other historical data, while paying attention to different behaviours when ‘ascending’ and ‘descending’. One better strategy based on tendency with several backward steps 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 ANNs are able to overcome those limitations. They have many important advantages over the traditional statistical models, most notably nonlinear generalisation 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 modelling 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 scheduling decisions in grid environments. There are three major differences between the approach of Huang et al. and our approach. First, their method utilised a recurrent network while our method employs feed forward networks for the purpose of ensuring both high performance and low overhead. Second, and more important, their work, as well as other previous works, merely focused on performance, in particular the execution time of applications running using the proposed scheduling method. Our novelty here, which is an important difference, is that we not only improve the performance, but we also consider the cost, namely the cost for training, validating and testing, to examine if such a neural network-based solution is feasible in dynamic real-time settings. This is very important because the solution may not be applicable in real time if it takes hours or days for training. Third, their experiments were simply carried out with only one network architecture, a constant learning rate of 0.2 and 20 min of the load trace, whereas we used combinations of different architectures and learning rates with four load traces collected over tens of days.

3. Host load prediction with ANNs

3.1 An overview of ANNs

ANN, originally developed to mimic biological neural networks, is a computational model 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 form a mapping between inputs and desired outputs from the training set through learning. After the learning process has finished, it is able to understand the hidden dependencies between the inputs and outputs and generalise to data never before seen.

In this work, multilayer feedforward network [20], accompanied by backpropagation, which is the most widely used training algorithm for multilayer 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 for deployment in a real-time dynamic setting. Also, the availability of host load traces meets the needs of the multilayer feedforward network for input and output data.

3.2 Feedforward neural networks

Figure 1 depicts an example of a feedforward neural network in operation with a host load time series. The network has four network inputs where external information is received, and one output layer C with one node where the solution is obtained. The network inputs and the output layer are separated by two hidden layers: layer A with four nodes, and layer

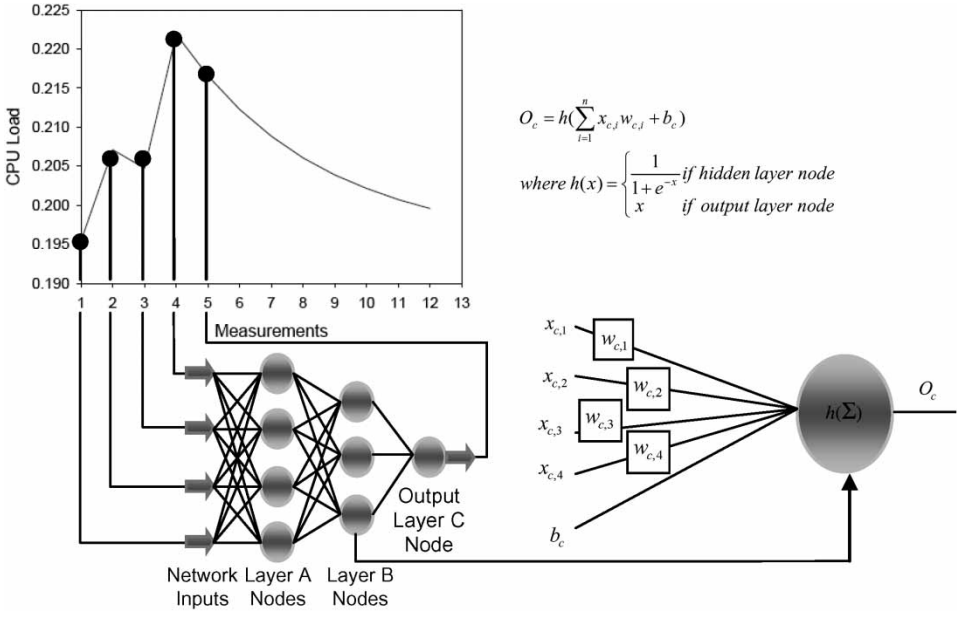

Figure 1. A three-layer feedforward network.

B with three 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 nodes in the preceding layer. Also shown in Figure 1, 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 node in the hidden layers and the output layer performs the calculation in Equation (1) on its input, and transfers the result O_c to the next layer.

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

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 training data sets 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 term 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 was trained with different learning rate values ranging from 0.01 to 0.3 in our experiments.

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 minimise a certain 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 normalised mean squared errors (NMSE) defined as follows:

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

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 customise and extend various features and parameters using Microsoft Visual Studio C++6.0. Moreover, we believe that development 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 that 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 exact numbers in neural networks. In this work, we followed the trial-and-error method for obtaining knowledge about a network architecture, which could promise both high performance and low overhead.

The trial-and-error process proceeded as follows. First, we tested the networks with several numbers of hidden layers. The more hidden layers there were, the longer time it usually took for training, which may last hours in the case of three or more hidden layers. We came to a conclusion that two hidden layers would probably meet the requirements. After deciding to stick to two-hidden-layer-architecture, we started to figure out suitable numbers of inputs. The networks were tested with a wide range of inputs, from a few to thousands of inputs. The outcome showed that hundreds of inputs would result in hours of training, too long to fit a dynamic environment. However, less than 10 inputs were deemed insufficient to produce good results. Lastly, the number of output layer nodes was the easiest, since it is necessarily one. Therefore, the networks that were tested are 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 learning rates valued at 0.01, 0.05, 0.1, 0.2 and 0.3.

Once the network parameters were determined, we were able to feed the data series to the networks for evaluation. The time series we chose here are the four most interesting load traces: axp0, axp7, sahara and themis in a number of load traces on Unix systems collected by Dinda [21]. 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 a number of previous samples to produce a load average which can be captured by a user program. These four load traces represent diversity both in capture periods and machine types, as shown in Table 1.

In addition, the load traces have to be normalised to a suitable range of values because the sigmoid activation function $h(x)$ has output of the range [0, 1]. Also, normalisation tends to make the training process better by improving the numerical condition and ensuring that various default values involved in initialisation and termination are appropriate. There are several different ways to normalise the network inputs well suited to different ranges of input values and applications. By observing the means and standard deviations (SDs) of the load traces, we confirmed 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 expected to maintain the original conditions and does not discard much information. The following normalisation formula is applied to every x in

Table 1. Descriptions of four load traces.

Name	Description	Collected load traces	Mean	SD
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

each load trace:

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

where $x_{\max}$ and $x_{\min}$ are the maximum and minimum value of the load trace, LWB is the lower bound and UPB is the upper bound of the interval [0.1, 0.9]. The four normalised load traces are presented in Figure 2.

Finally, each normalised load trace is divided into three 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 NMSE 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 backpropagation training process will be stopped and weights shall be restored to the previous values. The last set, the testing set, is then used for assessing the performance of the neural networks.

4.2 *Host load prediction results*

Figure 3 shows the first experimental results, the NMSE of the prediction errors for those four testing sets with four different network architectures and five different values for

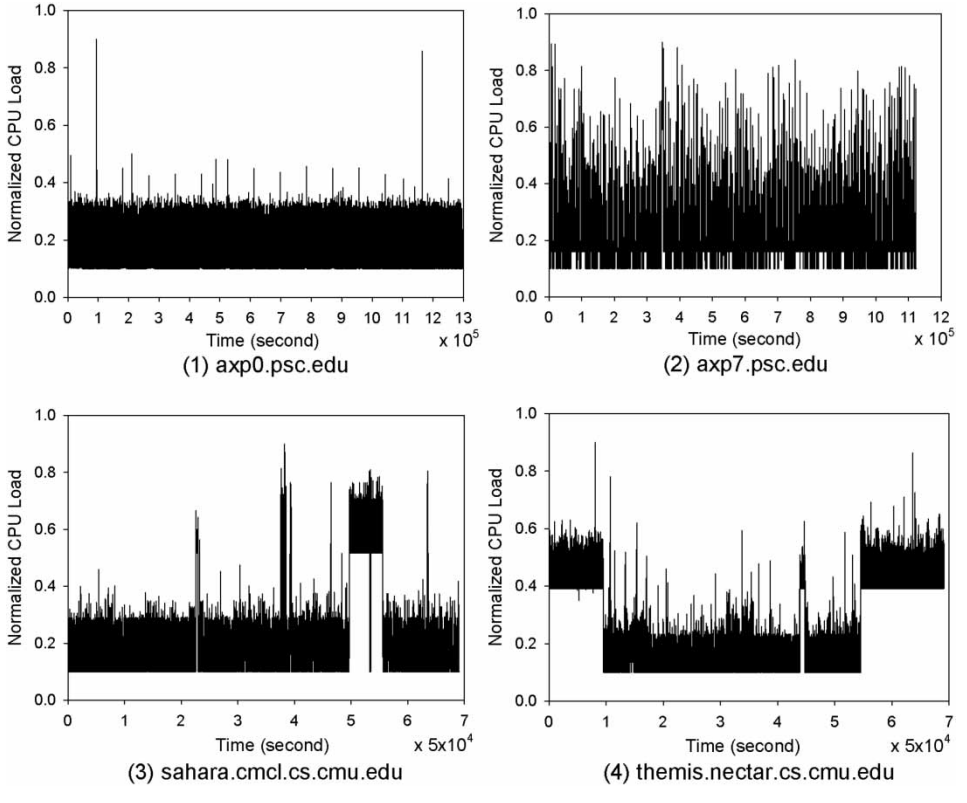

Figure 2. The normalised load traces.

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

learning rate. The prediction error for each load value in the testing set is calculated using the following equation:

$$\text{prediction error} = \frac{1}{N}\sum_{i=1}^{N}\frac{|O_i - D_i|}{D_i} \times 100\%, \tag{7}$$

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 ith trace, respectively.

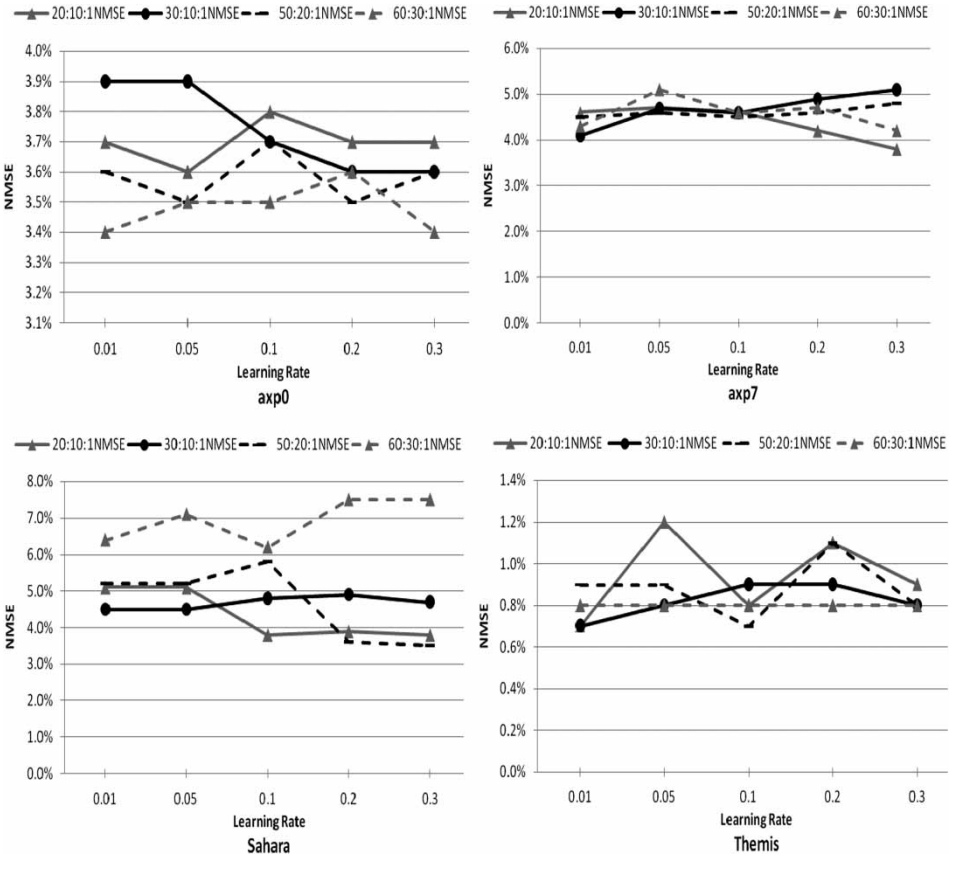

Figure 3. NMSE of the prediction errors.

As can be seen from the charts, 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. Such low values of these error terms imply that the neural network has successfully captured hidden behaviour of the host load and made highly accurate predictions. We find that no combination of learning rate and network architecture is the best for all the load traces, as values of NMSE keep changing with the changes in learning rate and network architecture, although the variation is quite small. However, the learning rate of 0.3 appears to play an important role in generating the lowest values of NMSE, in other words, the most accurate results. On three out of the four load traces, the networks of 60:30:1, 20:10:1, 50:20:1 and the learning rate of 0.3 produce the best values of NMSE on axp0, axp7 and sahara, respectively. The only exception is themis, where the network of 20:10:1 with the learning rate of 0.01 shows the best performance. As a result, the learning rate of 0.3 can be a good candidate for solutions aimed at high prediction accuracy.

Another significant factor that directly affects prediction accuracy is the shape of the load traces. The range of NMSE on themis is the smallest, while this range on sahara is the largest. In the middle range are the ranges of NMSE on axp0 and axp7. If we look back at Figure 2 and Table 2 for the shapes and partitioning of the load traces, we can easily recognise that the testing set of themis is actually repeated as one part of the training set, though they are not exactly the same. This repetition makes it easier for the neural network to predict the values which it has been trained with and results in the highest accuracy. In contrast, the testing set of sahara has some special patterns that can be found nowhere in the training set. This lack of repeating patterns causes a slightly lower degree of prediction accuracy on sahara. However, the situation can be improved considerably, providing we increase the size of the training set so that it covers non-repeating patterns of the corresponding testing set. These results once again have proved a strong relationship between the degree of self-similarity exhibited in the load traces and the degree of prediction accuracy.

Similar results are obtained with the mean and SD of the prediction errors, as displayed in Table 3. 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. The choices of learning rate and network architecture, again, 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 4. With 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 usually a connection between cases with the longest times and the largest network architecture of 60:30:1. In contrast, the smallest network architecture, 20:10:1, proves to contribute substantially to 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 learning rate of 0.3 and network architecture of 20:10:1 can be the most efficient choice for reduction in training time.

We also note that the size of training set obviously has a strong effect on the time needed to train 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 trade-off between accuracy and time; a training set which is too large takes the network a long time to complete, while a very

Table 3. Mean and SD of the prediction errors.

Mean and SD	Learning rate (%)					Mean and SD	Learning rate (%)				
	0.01	0.05	0.1	0.2	0.3		0.01	0.05	0.1	0.2	0.3
	axp0						axp7				
20:10:1mean	2.9	2.9	3.0	3.0	3.0	20:10:1mean	1.2	1.4	1.4	1.3	1.1
20:10:1SD	5.6	5.5	5.7	5.4	5.6	20:10:1SD	4.9	4.9	4.9	4.8	4.8
30:10:1mean	2.8	2.6	2.8	3.0	2.6	30:10:1mean	1.4	1.4	1.3	1.3	1.6
30:10:1SD	5.7	5.4	5.5	5.6	5.2	30:10:1SD	5.0	5.0	5.0	5.0	5.2
50:20:1mean	3.0	3.2	3.0	2.9	3.0	50:20:1mean	1.4	1.6	1.7	1.6	1.5
50:20:1SD	5.4	5.6	5.1	5.3	5.1	50:20:1SD	5.1	5.1	5.2	5.2	5.2
60:30:1mean	3.1	2.9	2.8	2.8	3.1	60:30:1mean	1.3	1.7	1.6	1.8	1.6
60:30:1SD	5.2	5.7	5.5	5.2	5.3	60:30:1SD	5.0	5.0	4.9	4.9	4.8
	sahara						themis				
20:10:1mean	7.8	8.1	6.5	5.5	5.9	20:10:1mean	1.6	3.0	2.1	2.7	2.3
20:10:1SD	8.5	8.3	8.1	8.1	8.2	20:10:1SD	3.0	3.3	3.0	3.2	3.1
30:10:1mean	7.5	7.1	7.6	7.0	6.1	30:10:1mean	1.6	2.1	2.2	2.1	2.1
30:10:1SD	8.1	8.1	8.2	8.4	8.4	30:10:1SD	3.0	3.0	3.0	3.0	3.0
50:20:1mean	8.8	8.7	9.5	6.5	6.1	50:20:1mean	2.5	2.5	1.9	3.7	2.3
50:20:1SD	8.0	8.0	8.2	7.9	7.8	50:20:1SD	3.0	3.0	2.9	2.5	2.8
60:30:1mean	11	12	10	9.4	6.6	60:30:1mean	1.8	1.9	1.9	1.9	2.1
60:30:1SD	8.2	8.8	8.1	8.1	8.0	60:30:1SD	3.0	3.0	3.0	3.0	2.9

Table 4. Training time in seconds.

Name	Learning rate				
	0.01	0.05	0.1	0.2	0.3
20:10:1axp0	725,681	68,548	49,436	26,880	18,641
30:10:1axp0	589,741	89,564	48,653	65,483	15,867
50:20:1axp0	1,356,874	75,648	85,671	20,458	21,534
60:30:1axp0	1,132,568	86,579	84,567	38,956	17,589
20:10:1axp7	163,102	9510	7402	2490	2492
30:10:1axp7	67,728	9657	5756	5363	578
50:20:1axp7	218,219	8182	1238	1199	1203
60:30:1axp7	235,868	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	16,752	56	57	6	7
30:10:1themis	21,431	95	56	14	14
50:20:1themis	1036	207	207	235	82
60:30:1themis	130,086	1027	1033	464	218

small set is inadequate to get a complete picture of the load traces. The outcome suggests that a training size of about 100,000, equivalent to the amount of load over 5 days, is reasonable to make it possible for the network to quickly respond with correct solutions in dynamic real-time grid environments.

Unlike the training time, which heavily depends on the network architecture and learning rate, the validating and testing times behave in a different way and remain fairly stable. Figure 4 shows 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 use of the same size data set. With the same set, larger network architecture causes a larger increase in the validating and

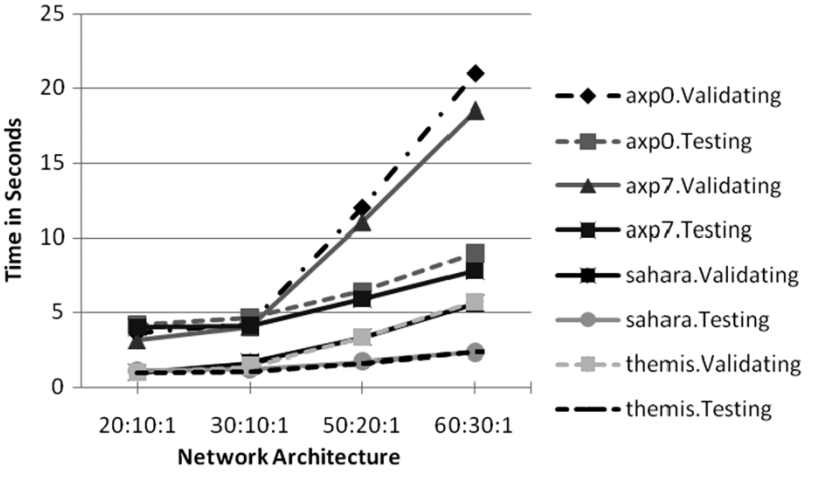

Figure 4. Validating and testing time.

testing times. However, in general this time is trivial compared with the training time. In the cases of sahara and themis, for instance, the predicted results for about 70,000 future load values can be obtained within a second.

4.3 Performance comparison

We compare the performance of the neural predictor with that of two previously proposed prediction methods in terms of mean and SD of the prediction errors. Based on our experimental results in the previous section, we selected the networks of 20:10:1 and 30:10:1, both with the same learning rate of 0.3, which provide both high degree of prediction accuracy and short training time. Their performances are compared to those taken from the 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]. This approach predicts the future load using a polynomial fitting method on several past load values and previous similar patterns. At the time of making this performance comparison, it was still the best performer in host load prediction.

Table 5 displays the results of performance comparison, with the best for each case displayed in bold face. The methods were evaluated with all the load traces, with the amount of data ranging from 1 day to the total available amount of collected traces.

We note that the neural network has achieved consistent improvement over the competitors in predicting future load values of the load traces in most cases due to its nonlinear generalisation, which is eminently suitable for capturing nonlinear behaviour in the host loads. Also, the neural predictor tends to produce more accurate results, provided there are more data in the data sets, mainly because of better training. For all the load traces, most of the best results are included in the total available amount of collected traces with both networks. This observation, however, does not appear in other methods, as they merely look ahead only one step based on the current tendency of data. They seem to usually commit a high degree of errors when the tendency changes direction.

On axp0 and sahara, both networks of 20:10:1 and 30:10:1 always outclass the other methods in all cases. The neural predictor reduces the average prediction error rate by approximately 4 to 72% on axp0 while on sahara it can predict up to 45% more accurately than the Zhang method. On axp7 where both the Yang and the Zhang methods were especially effective, except for the case of 10 days where the network of 30:10:1 cuts SD by half but is about 33% worse than the Zhang method in terms of mean, both neural networks perform better, with prediction error reduction ratio ranging from 25 to 59% for all other cases. Lastly, the 30:10:1 network, again, suffers low degrees of prediction accuracy for cases of 1 day, 5 days and 10 days on themis, but eventually improves greatly in other cases to achieve the best reduction ratio from 31 to 79%. Overall, the 20:10:1 network is superior to other methods in all cases (100%), and the 30:10:1 network is better than other methods in 18/22 cases (82%).

4.4 Performance with load traces collected on contemporary systems

So far we have used the load traces by Dinda, which are increasingly obsolete, and hence, experiments on contemporary systems are required to confirm if there exist any differences from those load traces. In this section, we validate the efficacy of our method with load traces collected on more modern systems [22]. As shown in Figure 5, abyss.cs.uchicago.edu has very low CPU load with mean and SD equal to 0.08 and 0.18,

Table 5. Performance comparison in terms of mean and SD of prediction errors, with the best for each case displayed in boldface.

axp0	1 day		5 days		10 days		15 days		20 days		75 days	
	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD
Yang et al. [11]	10.58	0.43	13.73	0.54	15.93	0.4	16.32	0.47	16.8	0.48	18.99	0.44
Zhang et al. [12]	6.54	0.20	7.86	0.22	8.31	0.33	9.33	0.28	10.14	0.39	10.57	0.37
20:10:1 Network	6.26	0.1	6.32	0.11	3.51	0.06	3.7	0.06	3.51	0.06	**3.00**	**0.05**
30:10:1 Network	**4.56**	**0.09**	**5.68**	**0.1**	**3.2**	**0.06**	**3.23**	**0.06**	**3.23**	**0.06**	3.15	0.05

axp7	1 day		5 days		10 days		15 days		20 days		65 days	
	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD
Yang et al. [11]	20.45	0.22	18.0	0.26	17.02	0.39	15.34	0.29	20.45	0.32	15.02	0.38
Zhang et al. [12]	3.14	0.16	3.06	0.23	2.92	0.13	2.83	0.07	2.81	0.08	2.73	0.08
20:10:1 Network	**1.98**	**0.08**	**1.26**	**0.05**	**1.77**	**0.07**	**1.36**	**0.05**	1.77	0.06	**1.11**	**0.05**
30:10:1 Network	2.36	0.08	1.8	0.05	3.87	0.07	1.68	0.05	**1.54**	**0.06**	1.6	0.05

sahara	1 day		5 days		10 days		15 days		20 days	
	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD
Yang et al. [11]	23.59	0.33	20.9	0.35	20.2	0.26	18.74	0.34	18.28	0.37
Zhang et al. [12]	7.54	0.25	9.86	0.31	10.31	0.12	11.33	0.21	9.2	0.23
20:10:1 Network	6.36	0.17	6.65	0.15	10.3	0.1	**6.25**	**0.08**	**5.95**	**0.08**
30:10:1 Network	**5.86**	**0.16**	**6.04**	**0.14**	**9.03**	**0.1**	6.66	0.08	6.12	0.08

themis	1 day		5 days		10 days		15 days		20 days	
	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD	Mean (%)	SD
Yang et al. [11]	3.13	0.05	20.3	0.55	25.85	0.34	27.48	0.51	27.37	0.48
Zhang et al. [12]	1.97	0.05	8.93	0.4	9.51	**0.04**	11.39	0.14	10.6	0.25
20:10:1 Network	**1.78**	**0.05**	**3.11**	**0.19**	**3.16**	0.18	**3.41**	0.21	2.34	0.03
30:10:1 Network	4.65	0.1	27.37	0.17	11.39	0.09	7.86	**0.07**	**2.18**	**0.03**

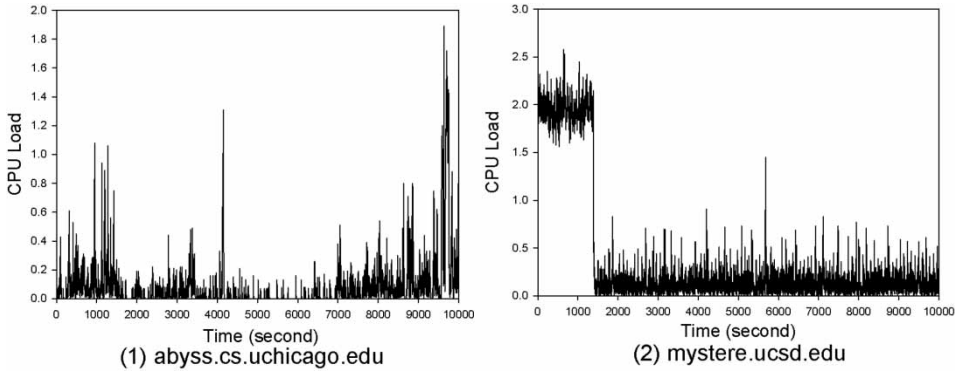

Figure 5. The load traces collected on more modern systems.

respectively. Meanwhile, mystere.ucsd.edu has a mean of 0.4, divided into two parts and SDs are 0.08 and 0.15 for each part. Both have the same size of 10,000 samples, quite small compared to those by Dinda but good enough for our validation.

We ran several experiments to evaluate the selected networks of 20:10:1 and 30:10:1 with a constant learning rate of 0.3 on these two load traces, along with axp0 as the representative of Dinda's load traces, for cases of 10,000 and 5000 samples. The experimental results are shown in Table 6, with the best for each case displayed in bold face, where abyss and mystere stand for abyss.cs.uchicago.edu and mystere.ucsd.edu, respectively. As expected, both networks have exhibited high prediction accuracy and low overhead with not much difference between the new traces and the older ones. NMSE is maintained in the range of [1.53–7.56%] and the training time is virtually only a few seconds. The longest time needed for training is 56 s for 10,000 samples on abyss. The results strongly encourage feasibility of applying the neural predictor in a wide variety of computing resources. No matter how modern the computing systems are, their load traces basically share similar characteristics which the neural networks can capitalise on to predict their future load.

5. Conclusion

In this work, 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 on the load traces in most cases. For example, the 20:10:1 network with a learning rate of 0.3 always outperforms the method which remained the best until the time of writing, by reducing the mean of the prediction error by up to 79%. 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 and clouds. 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 improve efficiency in

Table 6. NMSE (%), Mean (%), SD of the prediction errors and training time (T. Time), validating and testing time (V. Time) of the networks (in s) with the best for each case displayed in boldface.

	10,000 samples					5000 samples				
	NMSE	Mean	SD	T. Time	V. Time	NMSE	Mean	SD	T. Time	V. Time
abyss										
20:10:1	**2.52**	8.45	0.13	≈ 0	≈ 0	2.98	8.47	0.07	≈ 0	≈ 0
30:10:1	7.5	**4.95**	0.11	1	≈ 0	5.24	13.14	**0.05**	$\approx$ **0**	$\approx$ **0**
mystere										
20:10:1	5.49	8.7	0.08	4	≈ 0	**1.53**	7.7	**0.07**	13	≈ 0
30:10:1	4.23	**6.82**	0.07	56	≈ 0	3.17	11.92	0.09	**1**	$\approx$ **0**
axp0										
20:10:1	7.56	5.06	0.09	5	≈ 0	8.73	6.9	0.09	1	≈ 0
30:10:1	**6.92**	**5.05**	**0.09**	5	≈ 0	9.9	7.44	0.1	**1**	$\approx$ **0**

other areas beyond the area of host load predictions, for instance, in relation to network traffic and latency, etc.

Acknowledgements

We would like to thank Dinda and Yang for the use of their collected load traces. Thanks are also due to the anonymous reviewers for their valuable comments to help improve this work. This research has been partially supported by Ministry of Education, Culture, Sports, Science and Technology of Japan under the '21st Century COE Program'.

References

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, 1999.
[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, *Application-Level Scheduling on Distrisbuted Heterogeneous Network*, Supercomputing'96, Pittsburgh, 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*, 36th Annual Hawaii International Conference on System Sciences, Hawaii, 2003.
[5] M. Alt and S. Gorlatch, *Adapting Java RMI for Grid computing*, Parallel Comput. Technol. 21(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*, 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, 2004.
[7] P. Dinda, *The Statistical properties of host load*, Sci. Programming 7 (1999), pp. 3–4.
[8] P. Dinda and D. O'Hallaron, *Host load prediction using linear models*, Cluster Comput. 3(4) (2000), pp. 265–280.
[9] P. Dinda, *A prediction-based real-time scheduling advisor*, 16th International Parallel and Distributed Processing Symposium, Fort Lauderdale, 2002.
[10] R. Wolski, *Dynamically forecasting network performance using the network weather service*, J. Cluster Comput. 1 (1998), pp. 119–132.
[11] L. Yang, I. Foster, and J. Schopf, *Homeostatic and tendency-based CPU load prediction*, International Parallel and Distributed Processing Symposium, Nice, 2003.
[12] Y. Zhang, W. Sun, and Y. Inoguchi, *CPU load predictions on the computational grid*, IEEE International Symposium on Cluster Computing and the Grid, Singapore, 2006.
[13] G. Zhang, B.E. Patuwo, and M.Y. Hu, *Forecasting with artificial neural networks: The state of the art*, Int. J. Forecast. 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*, Int. J. Info. Technol. Decis. Mak. 5(4) (2006), pp. 639–658.
[15] B.G. Aslan and M.M. Inceoglu, *Comparative study on neural network based Soccer result prediction*, 7th International Conference on Intelligent Systems Design and Applications, Rio de Janeiro, 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*, 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*, International Joint Conference on Neural Networks, Orlando, 2007.
[18] G.P. Zhang and M. Kline, *Quarterly time-series forecasting with neural networks,* IEEE Trans, Neural Netw. 17(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*, 3rd International Conference on Natural Computation, Hainan, 2007.
[20] R.D. Reed and R.J. Marks, *Neural Smithing*, The MIT Press, Cambridge, 1999.
[21] P. Dinda, *Load Traces Archive*, Available at http://www.cs.northwestern.edu/~pdinda/LoadTraces/.
[22] L. Yang, *Load*, Available at http://people.cs.uchicago.edu/~lyang/Load/.