Evaluating the use of Artificial Neural Networks for Platform-independent Performance Prediction of Distributed Applications

Flores-Contreras, Jesus[1]; Duran-Limon, Hector[2] & Mezura-Montes, Efren[3]

1 Universidad Autónoma de Nuevo León, Facultad de Contaduría Pública y Administración,

Monterrey, Nuevo León, México, jesus.floresct@uanl.edu.mx, Av. Universidad S/N Col. Ciudad Universitaria

2 Universidad de Guadalajara, Departamento de Sistemas de Información,

Guadalajara, Jalisco, México, hduran@cucea.udg.mx, Periferico Norte No. 799, Col. Los Belenes

3 Universidad Veracruzana, Departamento de Inteligencia Artificial,

Xalapa, Veracruz, México, emezura@uv.mx, Sebastian Camacho No. 5. Zona Centro

Información del artículo arbitrado e indexado en Latindex:
Fecha de envío: Abril de 2017
Revisado por pares
Fecha de publicación en línea:Julio de 2018

Abstract

A distributed system is a set of independent computer systems interconnected by a network, which work in a cooperative way and behave as a single system creating an underlying platform for different kinds of applications. These platforms are usually used to execute long-running applications that demand a lot of computational resources e.g. CPU processing power, memory, and network bandwidth. In such kind of systems it is important to manage the available resources in an efficient way in order to improve the system's overall performance. Knowing how an application's runtime is going to behave can greatly improve performance of a system, since this information allows the efficient distribution of available resources. In this work, we present an evaluation of the suitability of artificial neural networks to achieve a platform-independent approach to execution time prediction of distributed applications running on multi-core systems. We performed our evaluation with three parallel long-running applications, namely the Weather Research and Forecasting (WRF) model, Octopus, and miniFE. Our results indicate that neural networks are capable of producing accurate results when predicting the application runtime on the same platform, but its accuracy decreases when the platform is changed.

Keywords

Distributed systems, Platform-independent, Prediction of runtime, Machine Learning technique, Long-running applications

1. Introduction

Distributed systems like Cluster and Grids are a set of autonomous computers interconnected by a network connection, which work in a cooperative way and behave as a single system creating an underlying platform for different kinds of applications. Applications running on these platforms commonly exhibit a non-linear performance behaviour [Shimizu et al., 2009].

An efficient management of the resources in the distributed system can improve the system performance. One way to improve such resource management is applying performance prediction models [Tsafrir et al., 2007]. For example in [Qian et al., 2008] a prediction model was used in a scheduling algorithm to improve the system performance. Performance prediction is usually used to improve workload management, and scheduling on distributed systems [Oliner et al., 2011]. However, a disadvantage of these models is that they are usually platform-dependent; hence it becomes necessary to redesign these models in order to apply them on different platforms. In contrast, platform-independent models [Shimizu et al., 2009] can be used on unseen hardware configurations (each configuration involving a different amount of CPU speed, number of processors, number of cores, RAM, etc.) where the same architecture instruction set (e.g. Intel) is assumed. A platform-independent model is produced as a result of applying the approach to different hardware configurations, which typically involves running a number of benchmarks on such hardware configurations. The produced model can then be used to predict the performance of an application on unseen hardware configurations where no benchmarks have been run. As an example, these approaches can be used for capacity planning where it is necessary to know whether the characteristics of a certain cluster system to be bought are enough to obtain the desired performance of specific applications.

Different methods can be used to predict the performance of an application. For example, linear regression [Sharkawi et al., 2012], non-linear regression [Carrington et al., 2006], autoregressive moving average [Dong et al., 2012], support vector machines [Hu et al., 2012], classification and regression trees [Li et al., 2009], and artificial neural networks [Oyamada et al., 2008]. The complexity of applying such methods depends on the system behaviour. For instance, the use of statistical mathematical methods in multi-core systems, which present a non-linear behaviour, can derive in computational complex models.

Other kinds of methods used for performance prediction are based on machine learning techniques. Although, there are several machine learning techniques, it is known that artificial neural networks can achieve or improve the prediction accuracy of mathematical methods [kundu et al., 2010][Sarangi and Bhattacharya, 2005][Upadhyaya et al., 2013] in different areas of study. In this paper, we establish the following statement; neural networks are suitable to define a platform-independent prediction method for long-running applications on unseen platform configurations. Hence, we focus on the execution time prediction of applications running in a free-workload environment. Our evaluation is performed by using three long-running applications, where each application demands different resources from the underlying platform.

This paper is structured as follows. Section II details the related work of our research. Section III describes the neural network design process, and defines the variables of the artificial neural network. Section IV presents the evaluation method. Section V discusses the results obtained from the evaluated structure. Finally, some concluding remarks are given in Section VI.

2. Related work

Several approaches to estimate the execution time of an application have been proposed. Each proposal makes use of different methods, such as, linear regression and machine learning techniques. For example in [Yang et al., 2005] the authors propose an architecture independent prediction approach. In order to estimate the application's runtime in different architectures, their method needs to perform a full execution on the reference system and a partial execution in the target architecture. Some disadvantages of this approach are: a) the necessity of performing a partial execution in the target system and b) the necessity of knowing the control flow of the application. In [Shimizu et al., 2009] the authors developed a performance prediction method that is agnostic and platform-independent, which considers the application's resource usage. This method measures the usage of several resources such as cache memory, and network bandwidth communication. The values measured are later on used as input parameters for a linear regression-based prediction model. However, the approach is only intended for single-core processors. Therefore, this method is not suitable for multi-core systems where the performance behaviour of an application is non-linear. In [Gianni et al., 2010] the authors proposed a method to predict the running time of a simulation system in a distributed environment. Their proposal is an iterative process that needs to perform a characterisation of the application. A disadvantage of this proposal is that the prediction method has to be implemented in the design phase of the application. In [Sanjay and Vadhiyar, 2008] the authors developed a modelling strategy to predict the runtime of parallel applications in dedicated and non-dedicated clusters. The model takes into consideration several resources in which resources are represented in an algebraic expression. This model offers several functions and the user must select one that fits better to the application behaviour. A disadvantage of this approach is that the user must perform the function selection to get better accuracy.

As we can see, there are several proposals to predict the execution time of an application. However, such attempts present some disadvantages such as them being applied during the design phase, or the burden imposed to characterise the application, and the inability to perform platform-independent predictions on multi-core systems.

Regarding the use of neural networks, in [Oyamada et al., 2008] the authors are able to predict the application performance of an application in an embedded system using an artificial neural Network. The authors claim to use neural networks because embedded systems normally have a non-linear behaviour. In [Dodonov and de Mello, 2010] the authors present a prediction method based on neural networks and chaos theory. Their method involves an off-line and on-line monitoring evaluation of the application. The off-line evaluation analyses the process execution traces, and the on-line evaluation relies on monitoring the application state while executing, a disadvantage of this approach is that the on-line evaluation requires modifying the application in order to monitor specific points of the application. In [Ipek et al., 2005] authors proposed a method to evaluate the performance prediction based on the input data. They considered to use a multilayer artificial neural network as prediction method. Their proposal is designed for a specific application, the semicoarseni game multi grid solver (SMG2000). The neural network consists of a feed forward architecture with an activation sigmoid function. Results showed that artificial neural networks did not have good performance on predicting the application execution time. This low performance is because the activity of the system, activities as sharing resources between processes, this generates variations in the performance, which are considered as noise. In [Dauwe et al., 2016] proposed two prediction models, based on linear regression, and artificial neural networks. These models focused on predicting the performance of the application in co-located environments. Results determined that neural networks have a better performance than linear regression method. Although, some of the prediction methods use neural networks, none of these approaches has been unseen platform configurations. Because of that, we establish the following neural networks are suitable to define a platform-independent prediction method for unseen platform configurations.

3. Artificial Neural network structure design

Several attributes can be considered in order to predict the application runtime. For example in [Carrington et al., 2006] estates that only two resources are necessary to estimate the application runtime on systems with single-core processors, CPU and network bandwidth. However, we are dealing with a multi-core environment, and we considered that there are several characteristics that should be taken into account in order to predict the application runtime. We assumed that the application runtime could be estimated by equation 1.

$$\tau_{app} = f(\mu, \upsilon, \omega, \varphi, \beta) \quad \textbf{(1)}$$

Where :

- τ_{app} represents the application runtime in the distributed system
- μ is the number of Cores used to execute the application
- υ defines the number of nodes used
- ω indicates the operation frequency of the CPU Core
- φ is the memory available on each node
- β is the network bandwidth in the system

Therefore, for our evaluation we considered these parameters to estimate the runtime of an application.

A) Design of the Neural Network Structure

In order to determine the most optimal neural network configuration for each application, we evaluated several structures of neural networks for each one of our tested applications. Each network structure was executed thirty times and for each structure evaluation we used a random sampling validation method [Raychaudhuri, 2008].

We evaluated several structures by varying the following characteristics of the neural network: activation functions (logsig,tansig), training algorithm (Levenberg-Marquart -LM-, Bayesian Regularization -BR-, and Scaled Conjugated Gradient -SCG-), and the number of neurons in the hidden layer. Table I contains a summary of the results obtained for each structure. The accuracy of each structure was determined considering the determination coefficient[1] R^2 and the root medium squared error[2] (RMSE) of each evaluated structure. The best values are shown in bold in Table I.

Table I Summary of the accuracy of different neural network structures

WRF								
	10 Neuros				15 Neuros			
Training	R^2		MSE		R^2		MSE	
Algorithm	logsig	tansig	logsig	tansig	logsig	tansig	logsig	tansig
LM	0.9980	0.9981	23.5988	23.4795	0.9981	0.9981	23.4257	22.9985
BY	0.9984	0.9984	21.5448	21.5567	**0.9984**	0.9984	**21.3765**	21.4326
SCG	0.9619	0.9815	94.4068	64.2824	0.9556	0.9799	102.6666	69.8754

Octopus								
	10 Neuros				15 Neuros			
Training	R^2		MSE		R^2		MSE	
Algorithm	logsig	tansig	logsig	tansig	logsig	tansig	logsig	tansig
LM	0.9691	0.9679	190.5737	194.4141	0.9709	0.9709	185.2404	185.2850
BY	0.9751	0.9743	170.9223	173.6385	**0.9765**	0.9757	**165.9382**	168.5926
SCG	0.9347	0.9504	271.5035	239.0471	0.9258	0.9490	289.3011	241.8977

miniFE								
	10 Neuros				15 Neuros			

[1] Determination coefficient is the variation proportion between the predicted values and the real values. The higher the value, the better the precision of the model.

[2] RMSE measures how much error exists between two datasets, i.e. it is used to compare a predicted value against an observed value. It ranges from 0 to ∞ and has negative oriented scores. A RMSE value close to zero is better.

Training	R^2		MSE		R^2		MSE	
Algorithm	logsig	tansig	logsig	tansig	logsig	tansig	logsig	tansig
LM	0.9926	0.9926	15.4763	15.5539	0.9928	0.9927	15.2698	15.3684
BY	0.9931	0.9931	14.9572	14.9538	**0.9932**	0.9932	**14.8246**	14.8699
SCG	0.9500	0.9736	37.4262	27.4687	0.9535	0.9734	36.9939	27.8267

From the evaluated structures, we selected the most suitable neural network with the higher R^2 and lower RMSE values. As we can see in Table I, the neural network structure that satisfies this requirements is the network structure with 15 neurons on the hidden layer, Bayesian Regularization training algorithm, and logsig activation function.

4. Evaluation

This section describes the cluster platform, tools, and applications that we used to carry out our experiments. We also describe the experimental settings and experiment results obtained for platform-dependent and platform-independent scenarios.

A. Platform, tools, and applications

The underlying platform of our experiments consisted of a cluster with eight nodes, each node has two Intel Quad-core Xeon processors at 2.13GHz, 16 GBytes of Memory RAM, and they are interconnected through an Ethernet connection 10/100/1000 BaseT. We used two tools to emulate different platform configurations in this system: cpuupower and wondershaper. cpupower allowed us to manipulate the clock rate frequency on each core processor whereas wondershaper allowed us to increment or decrement the amount of network bandwidth.

As we mentioned in section I, three different applications were used to evaluate the performance of the neural network. WRF model [Mesoscale and of NCAR, 2007], Octopus [Developers, 2000], and miniFE [Michael A. Heroux, 2013]. WRF is a cpu- and communication-intensive application, it forecasts the weather of a specific geographical region. Octopus is also a cpu- and communication-intensive application, which performs differential finite operations. The main purpose of Octopus is to simulate the electron-ion dynamics of one-, two-, and three-dimensional finite systems and it is based on time-dependent density-functional theory (TDDFT). Finally, miniFE is a memory-intensive application. miniFe is a mini application that implements a couple of kernels to solve linear system operations using a conjugate-gradient algorithm.

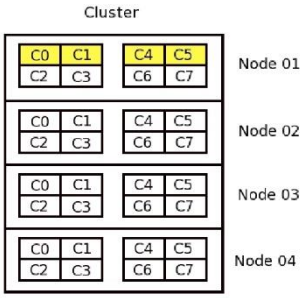
The input data of each application remained constant in all the experiments. The input data for the WRF model was fixed to 24 hours of land surface of \$5625 Km^2 of the data obtained on March 1st, 2012. The miniFE input data defines the dimension domain of the finite element; dimension is defined by three parameters nx (335), ny (347), and nz (347). Finally the Octopus performs a simulation of the Benzene molecule. The energy units was defined in electronvolts, length unit in Armstrongs, the radius of the box shape is 8, and the space between points in the mesh was 0.10.

B. Experiment description

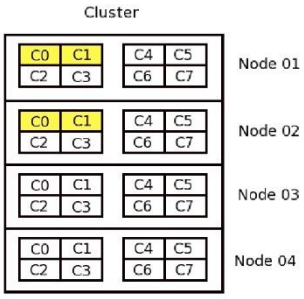
The experiments were carried out under different resource configuration schemes and platform scenarios, where the combination of both gives a particular platform configuration. A resource configuration scheme consists of a combination of a defined number of cores. For example, Figure 1 depicts three configuration schemes where the application was executed using four cores. We evaluated the ratio of variation of the execution time for each configuration scheme. We executed several configuration schemes in the system at a different time and different date. We found that the coefficient of variation[3] of the execution time had a value less than 0.26%, in every configuration. This value indicates that the execution time of a configuration has a low variation and the execution time can be considered similar for each execution of the same configuration. Therefore, we considered executing each configuration twice in the experiments was enough. A platform scenario emulates a different platform in the same system by modifying the system's characteristics. In our case we modified the CPU frequency and network bandwidth.

[3] Coefficient of variation determines the ratio variation of the standard devi- ation respect to the media value. A higher value represent higher heterogeneity in the data, a lower value indicates higher homogeneity

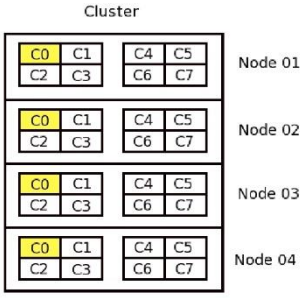
Several platform scenarios were emulated by setting the clock rate frequency at 1.60GHz, 1.73GHz, 2.00GHz, and 2.13GHz, and network bandwidth at 60Mbps, 80Mbps, and 100Mbps. Each resource configuration scheme was combined with the different platform scenarios to obtain a particular platform configuration.



a) First configuration scheme



b) Second configuration scheme



c) Third configuration scheme

Figure 1: Example of three configuration schemes using four Cores to execute the applications

We generated two datasets, which were mutually exclusive. The reference dataset (R-ds) was used for training, validating, and testing the neural network and contained 765 different platform configurations. Therefore, we had 1530 execution instances for each application (given that each application was run twice). The dataset E-ds was used to evaluate the prediction accuracy of the neural network in a platform-independent scenario. The dataset (E-ds) contained 550 instances of unseen platform configurations. Each record of both datasets, R-ds and E-ds, contained the execution time of an application as well as the platform configuration used in terms of number of cores, number of nodes, amount of bandwidth, amount of memory, and CPU frequency. Table II depicts the different scenarios contained in each dataset. For example, the dataset R-ds has the execution time of an application run on the different configuration schemas combining 1 core, within 1 node, a CPU frequency of 1.60GHz, and a network bandwidth of 100Mbps. The memory remained constant in all the scenarios with a value of 16GBytes.

Table II Data of the scenarios on each dataset

Frequency	Dataset	Cores	Nodes	Bandwidth	Number of instances
1.60 GHz	R-ds	1-8	1	100 Mbps	70
		2-16	2	100 Mbps	310
	E-ds	1-8	1	40,60 Mbps	70
		3-24	3	100 Mbps	112
1.73 GHz	R-ds	1-8	1	100 Mbps	70
		2-16	2	100 Mbps	290
	E-ds	1-8	1	60 Mpbs	32
		3-24	3	100 Mbps	112
2.00 GHz	R-ds	1-8	1	60, 80, 100 Mbps	140
		2-16	2	100 Mbps	290
	E-ds	3-24	3	100 Mbps	112
2.13 GHz	R-ds	1-8	1	100 Mbps	70
		2-16	2	100 Mbps	290
	E-ds	3-24	3	100 Mbps	112

We observed that the execution time of an application on each single core in the same processor had a coefficient of variation of 0.15%, which means that the execution time of an application is similar for the cores contained within the same processor. However, the execution time of an application in our platform can vary significantly (e.g. 5.30% in the case of WRF) when it is run on cores of different processors. Therefore, for the dataset R-ds we only considered configurations of cores of different processors. For instance, in the case of one core and one node there are two platform configurations e.g. core 0 (processor 1) and core 4 (processor 2). In case of two cores and one node, there are three platform configurations e.g. [core 0, core 1], [core 0, core 4], and [core 4, core 5]; and so on. Finally, all our experiments assume a free-workload environment.

C. Evaluation accuracy for a platform-dependent scenario

We evaluated the prediction accuracy of the neural network for seen platform configurations, where the applications had previously been run. In this case we used the data in R-ds. The dataset was divided randomly in three parts namely as: training, validating, and testing. Training has 70% of the data, and validation 15%, both were focused on training the neural network and on minimising the overfitting. The testing part had the remaining 15% of the data and it was focused on measuring the real prediction accuracy of the neural network. We evaluated 10 times the accuracy of the neural network. Table III presents the statistical results of the prediction accuracy of the neural network for the scenarios described in the dataset R-ds.

Table III Data of the scenarios on each dataset

	miniFE		Octopus		WRF	
	R^2	RMSE	R^2	RMSE	R^2	RMSE
Min	0.9930	14.6835	0.9696	155.2306	0.9983	21.1403
Media	0.9932	14.8115	0.9770	164.4985	0.9984	21.4811
Median	0.9933	14.7641	0.9791	157.4070	0.9984	21.3364
Max	0.9934	15.0892	0.9797	189.2693	0.9984	21.9540
Desv. Std.	0.0001	0.1347	0.0041	13.8607	0.0000	0.3286

Table III illustrates the R^2 and RMSE of the neural network for each application. We can see that the R^2 of the WRF model, and miniFE value remained above of 99%, and their standard deviation is 0.00, which means that for each evaluated iteration the prediction accuracy was close to each other. On the other hand, the accuracy of the predictions of Octopus decreases to 97%, and its standard deviation is also close to zero. Thus, each evaluated iteration has the same accuracy value.

The RMSE value for miniFE and WRF were low. In the case of Octopus the RMSE was a little bit high. This is because the accuracy of Octopus was 2% lower than the accuracy of WRF and miniFE.

D. Evaluation accuracy for a platform-independent scenario

In this case, we used the same neural network that we generated with the dataset R-ds in the platform-dependent scenario. We evaluated the accuracy of the neural network for unseen platforms (i.e. the platform-independent scenario), where the dataset E-ds (detailed in Section IV.B) contains the data used to evaluate the prediction accuracy for unseen platform configurations. The accuracy was measured by considering the R^2 and the RMSE. Table IV shows the statistical results of the neural network method for each application.

Table IV Neural network model accuracy on unseen platforms

	miniFE		Octopus		WRF	
	R^2	MSE	R^2	MSE	R^2	MSE
Min	0.1789	86.8003	-0.7694	758.1247	0.1095	447.6400
Average	0.4611	312.7932	0.1214	2860.0835	0.4504	878.4788
Median	**0.4377**	**292.1792**	**0.2017**	**2641.9448**	**0.4765**	**776.8977**
Max	0.7860	601.6973	0.9126	6561.3607	0.7686	1914.8131
Desv. Std.	0.1996	146.5209	0.5221	1765.8757	0.2631	442.0952

As we see in Table IV, the R^2 of the neural network decreases when it estimates the application runtime on an unseen platform. The R^2 average accuracy drops down under 50%, however, the most significant decrement in accuracy is observed with Octopus, where the R^2 accuracy achieves an average of 12%. We also can see that the RMSE value significantly increases and clearly overpasses the values acquired by the neural network that estimates the application execution time based on the data obtained from the reference dataset R-ds.

5. Discussion

Our results show that the prediction accuracy of an application runtime on the same platform is very accurate since the neural network R^2on all the applications was above of 97%, and the RMSE value was low. But when the neural network was evaluated with the data of the emulated platform (i.e. in the case of the platform-independent scenario), its R^2 decreased below 50% for all the applications. Besides, the RMSE significantly increased compared to the RMSE obtained with the platform-dependent model.

These results could indicate that the selected neural network structure is overfitted, and the number of neurons in the hidden layer needs to be larger. However, we increased the number of neurons to 25 and 30 and we found that the prediction accuracy in the same platform remained above of 97%, but when a platform-independent scenario was used, the prediction accuracy for 25 neurons was 41%, and for 30 was 37%. We did not continue increasing the number of neurons because we could create an under fitted structure.

Although neural networks have been successfully employed to represent the non-linear behaviour of a system in different application domain areas, we have showed that neural networks are not suitable in for platform-independent performance prediction models. The reason of this can be that the neural network is affected by the input data in the learning process [Raza and Baharudin, 2012]. This is because the training algorithm adjusts the weights of the neurons according to the input data, hence, when the information changes abruptly the neural network is not able to estimate the application runtime. Although the datasets were randomly divided, this technique may not be suitable. Therefore, it could be necessary to implement another selection technique in order to reduce the sensitivity of the neural network to the input data in the learning process [Hayashida et al., 2014].

We believe further research is required to explore the suitability for defining platform-independent prediction models based on other machine learning techniques such as classification trees and classification and regression trees. This, given that such techniques have been used as prediction methods in different areas such as the medical area [AL-Dlaeen and Alashqur, 2014], hardware detection failure [Li et al., 2014], and for predicting the resource usage and the execution time of applications exhibiting non-linear behaviour and running on multi-core systems [Li et al., 2009][Matsunaga and Fortes, 2010].

6. Conclusion

Our goal was to evaluate the suitability of artificial neural networks for defining a platform-independent model able to predict the runtime of long-running applications in multi-core systems. We evaluated artificial neural networks because they have shown successful to accurately model the behaviour of systems, including non-linear systems, in different application domains.

In our study we defined an optimal neural network structure by evaluating different essential attributes of the neural networks. Our results showed that the structure should have 15 neurons in the hidden layer, the Bayesian Regularization training algorithm, and the logsig activation function. The neural network successfully modelled the performance of the tested applications in a platform-dependent scenario. However, the neural network structure was unsuccessful to provide accurate performance prediction values, for all the applications we tested, in a platform-independent scenario. We found that the accuracy significantly decreased below 50% for each tested application.

As future work, we will continue evaluating other machine-learning techniques in order to achieve a platform-independent approach to runtime prediction of long-running applications in a multi-core distributed environment.

References

[AL-Dlaeen and Alashqur, 2014] AL-Dlaeen, D. and Alashqur, A. (2014). Using decision tree classification to assist in the prediction of alzheimer's disease. In Computer Science and Information Technology (CSIT), 2014 6th International Conference on, pages 122–126.

[Carrington et al., 2006] Carrington, L., Snavely, A., and Wolter, N. (2006). A performance prediction framework for scientific applications. Future Gener. Comput. Syst., 22(3):336–346.

[Developers, 2000] Developers, O. T. (2000). Octopus. Accesed : 27/11/2014.

[Dodonov and de Mello, 2010] Dodonov, E. and de Mello, R. F. (2010). A novel approach for distributed application scheduling based on prediction of communication events. Future Generation Computer Systems, 26(5):740 – 752.

[Dong et al., 2012] Dong, F., Luo, J., Song, A., Cao, J., and Shen, J. (2012). An effective data aggregation based adaptive long term cpu load prediction mechanism on computational grid. Future Gener. Comput. Syst., 28(7):1030–1044.

[Gianni et al., 2010] Gianni, D., Iazeolla, G., and D'Ambrogio, A. (2010). A methodology to predict the performance of distributed simulations. In Principles of Advanced and Distributed Simulation (PADS), 2010 IEEE Workshop on, pages 1 –9.

[Hayashida et al., 2014] Hayashida, T., Nishizaki, I., Sekizaki, S., and Nishida, M. (2014). Structural optimization of neural networks and training data selection method for prediction. In Computational Intelligence and Applications (IWCIA), 2014 IEEE 7th International Workshop on, pages 171–176.

[Hu et al., 2012] Hu, L., Che, X.-L., and Zheng, S.-Q. (2012). Online system for grid resource monitoring and machine learning-based prediction. Parallel and Distributed Systems, IEEE Transactions on, 23(1):134 –145.

[Ipek et al., 2005] Ipek, Engin, de Supinski, Brunis R., Schulz, Martin, and McKee, Sally A. An Approach to Performance Prediction for Parallel Applications. Cunha Jose, and Medeiro Pedros, editors, Euro-Par, pages 196-205. Springer 2005.

[Kundu et al., 2010] Kundu, S., Rangaswami, R., Dutta, K., and Zhao, M. (2010). Application performance modeling in a virtualized environment. In High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on, pages 1 –10.

[Li et al., 2009] Li, B., Peng, L., and Ramadass, B. (2009). Accurate and efficient processor performance prediction via regression tree based modeling. J. Syst. Archit., 55:457–467.

[Li et al., 2014] Li, J., Ji, X., Jia, Y., Zhu, B., Wang, G., Li, Z., and Liu, X. (2014). Hard drive failure prediction using classification and regression trees. In Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on, pages 383–394.

[Matsunaga and Fortes, 2010] Matsunaga, A. and Fortes, J. (2010). On the use of machine learning to predict the time and resources consumed by applications. In Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, pages 495–504.

[Mesoscale and of NCAR, 2007] Mesoscale and of NCAR, M. M. M. D. (2007). Weather research and forecasting model. Accesed : 27/11/2014.

[Michael A. Heroux, 2013] Michael A. Heroux, S. N. L. (2013). Finite element mini application minife. Accesed : 27/11/2014.

[Oliner et al., 2011] Oliner, A., Ganapathi, A., and Xu, W. (2011). Advances and challenges in log analysis. Queue, 9(12):30:30–30:40.

[Oyamada et al., 2008] Oyamada, M. S., Zschornack, F., and Wagner, F. R. (2008). Applying neural networks to performance estimation of embedded software. Journal of Systems Architecture, 54(12):224 – 240.

[Qian et al., 2008] Qian, Z., Zeng, M., Qi, D., and Xu, K. (2008). A dynamic scheduling algorithm for distributed kahn process networks in a cluster environment. In Computational Intelligence and Industrial Application, 2008. PACIIA '08. Pacific-Asia Workshop on, volume 2, pages 36 –42.

[Raychaudhuri, 2008] Raychaudhuri, S. (2008). Introduction to monte carlo simulation. In Simulation Conference, 2008. WSC 2008. Winter, pages 91–100.

[Raza and Baharudin, 2012] Raza, M. and Baharudin, Z. (2012). A review on short term load forecasting using hybrid neural network techniques. In Power and Energy (PECon), 2012 IEEE International Conference on, pages 846–851.

[Sanjay and Vadhiyar, 2008] Sanjay, H. A. and Vadhiyar, S. (2008). Perfor- mance modeling of parallel applications for grid scheduling. J. Parallel Distrib. Comput., 68(8):1135–1145.

[Sarangi and Bhattacharya, 2005] Sarangi, A. and Bhattacharya, A. (2005). Comparison of artificial neural network and regression models for sedi- ment loss prediction from banha watershed in india. Agricultural Water Management, 78(3):195 – 208.

[Sharkawi et al., 2012] Sharkawi, S., DeSota, D., Panda, R., Stevens, S., Taylor, V., and Wu, X. (2012). Swapp: A framework for performance projections of hpc applications using benchmarks. In Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International, pages 1722–1731.

[Shimizu et al., 2009] Shimizu, S., Rangaswami, R., Duran-Limon, H. A., and Corona-Perez, M. (2009). Platform-independent modeling and prediction of application resource usage characteristics. Journal of Systems and Software, 82(12):2117 – 2127.

[Tsafrir et al., 2007] Tsafrir, D., Etsion, Y., and Feitelson, D. G. (2007). Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distrib. Syst., 18(6):789–803.

[Upadhyaya et al., 2013] Upadhyaya, S., Farahmand, K., and Baker- Demaray, T. (2013). Comparison of NN and LR classifiers in the context of screening native american elders with diabetes. Expert Systems with Applications, 40(15):5830 – 5838.

[Yang et al., 2005] Yang, L. T., Ma, X., and Mueller, F. (2005). Cross- platform performance prediction of parallel applications using partial execution. In Proceedings of the 2005 ACM/IEEE conference on Super- computing, SC '05, pages 40–, Washington, DC, USA. IEEE Computer Society.