A Neural Networks Approach for Intelligent Fault Prediction in HPC Environments

Kulathep Charoenpornwattana, Chokchai Leangsuksun
{kch020, box}@latech.edu

Computer Science, College of Engineering & Science
Louisiana Tech University, Ruston LA, 71272, USA

Geoffroy Vallée, Anand Tikotekar, Stephen L. Scott
{valleegr, tikotekaraa, scottsl}@ornl.gov

Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge TN, 37831, USA

Abstract

Reliability is a well-known issue in today's HPC environments and is expected to become even more challenging in the next generation peta-scale systems. Because current fault tolerance approaches (e.g., checkpoint/restart mechanisms) are considered to be inefficient due to performance and scalability issues, improved fault tolerance approaches such as Proactive Fault Avoidance (PFA) are today under investigation. The PFA approach is based on fault prediction and migration in order to reduce both the impact of failures on applications and the recovery time. In this document, we explore the usage of Artificial Neural Networks (ANNs) techniques for fault prediction improvement in a PFA context. By initially training the feed-forward network with a supervised back propagation learning algorithm, this network is then fed with historical IPMI sensor data collected from our cluster. Results show a prediction performance improvement over the previous "thresholds trigger" approach.

1. Introduction

Reliability is one of the major issues in the HPC systems today. It is expected to become even greater challenge in the next generation peta-scale systems. The Checkpoint/Restart (C/R) mechanism is the current most popular fault tolerance mechanism for failure recovery, which has been widely deployed in many production systems. The major concern with C/R mechanisms is that it may be inefficient in large scale systems due to the scalability and performance issues [18].

Previously, we have developed a mechanism for Proactive Fault Avoidance (PFA) as part of the Unified Fault Tolerance (UFT) Framework [14]. The mechanism is based on fault prediction, and live migration of virtual machines. The system's sensor and thresholds are gathered and analyzed online via the Intelligent Platform Management Interface (IPMI) for faults prediction. In addition, we take advantage of the fault tolerant features of virtualization to migrate the computing entity away from the system when a fault is observed or predicted, before causing system failure.

A common approach for fault prediction in the existing framework [13][14] is the usage of threshold triggers. Generally, the manufacturers set the default thresholds to suggest the conditions where the system should be operated. If the system doesn't operate under such conditions (*e.g.*, any critical attributes rise beyond or drop below the thresholds), it could indicate an anomaly. The threshold trigger approach is developed based on this fact. When the critical attributes reach a certain point near the threshold, an event *fault alarm* is generated and actions might be taken. The drawback of the threshold trigger approach is that false positive alarms may occur when the attributes are influenced by the environmental factors (*e.g.*, power oscillation, temporary changing of surrounding temperature). The environmental factors could cause fluctuation of sensor attributes beyond the thresholds but unlikely to cause system failures. Furthermore, once the attributes reach a certain point, it doesn't guarantee that the attribute will continue raising or dropping to the thresholds. If an alarm is generated in such situation and PFA attempts to migrate tasks away, valuable resources will be exhausted, which is intolerable for large scale systems.

Artificial Neural Network (ANN) has been long studied and applied in wide range across problem domains especially in business and finance (*e.g.*, sale forecasting, stock/financial prediction). It also has been successfully applied in engineering-related problems such as machinery condition monitoring and electric load forecasting [10]. However, to the best of our knowledge, ANN for hardware based fault prediction in HPC environments has not been done in any literatures. The principal strength of the ANN is the ability to find the relationship pattern between *input* and *output*, then to recognize the pattern forming a model, which later could be used to predict the output when inputs are given.

The purpose of this paper is to study an applicability of ANNs for fault prediction in HPC environments and to demonstrate the possibility of adapting such methods for fault prediction purpose, which could be important for further research in this area. The rest of this paper is organized as following: Section 2 discusses the related work. In Section 3, we explain the methodology, and show how we apply an ANN for component fault prediction. We validate our method and present the experimentation results in Section 4. We conclude our work in Section 5.

2. Related research

ANNs have been studied for over 50 years. The concept of the ANNs was introduced with an attempt to create a model, which could simulate the human brain functions. The model was expected to be able to make logical decisions based on the past experience. The first publication toward research in neural networks was published by McCulloch and Pitts in 1943, which studies and describes human brain functions using mathematical models. In 1949, a psychologist, Hebb, discovered that *synaptics*, connections between neurons inside the human brain, are changing as a person learns. Today, we have seen ANNs applied in the real world to solve different types of problem domains and it could produce fairly accurate results in forecasting problems. W. Tian in [11] used ANN to predict web contents that are likely to be re-accessed. The predicted contents could be cached in order to decrease future access time.

Many authors reported successes applying ANNs to their research. The majority of these studies are done in the area of business and financial. The following are examples of ANNs in business and financial applications: Kryzanowski [2] and Wong [5] use ANNs to help in stock selection; Refenes [3] uses ANNs to evaluate stock performance; in [6] authors use ANNs in Financial Asset Management; and Moody [7] explores ANNs to predict economy. ANNs do not only emerge into financial and business fields, but there are also several publications in medical and engineering fields which reporte the uses of ANNs. For example, Atienza and et al. employs ANNs to stratify risk of heart failure [8]. ANNs is also applied for face recognition application [12] and for hand written recognition [9].

A number of fault tolerance frameworks for PFA [13][14][15] have been designed and developed for a deployment in large scale systems. Studies have only focused on fault tolerance methods and policies. The fault prediction mechanism in the framework relies on the simple threshold trigger method. Data collected from hardware monitors is assumed to be accurate and does not lead to false prediction [13]

3. Fault Prediction with Artificial Neural Network

Artificial Neural Network (ANN) has been reported to successfully resolve complex problems in several different disciplines. In addition, several studies have shown that ANN is able to use in time-series prediction with fair accuracy results. In this section, we present a methodology for fault prediction in HPC environments based on ANNs. We use ANN for time-series prediction to anticipate future sensor data. The ultimate goal is to apply ANN in our proactive fault avoidance framework [13][14] to create an intelligent fault predictor to minimize the number of fault alarms.

We apply Java Object Oriented Neural Engine (Joone) [16] to build and train the network. Joone is a free java based ANN framework for building and training ANN. It supports several learning algorithms and network topologies. In addition, it provides Java APIs for developers to build and train networks in their applications. We build a feed-forward multi-layer perceptron network with 29 sigmoid neurons in input layer, 45 sigmoid neurons in a single hidden layer, and 1 sigmoid neuron in output layer. The neurons are fully connected with synapses to the neuron in the next layer.

The network is built with the Joone GUI editor as shown in Figure 3.1.

The network is trained with a supervised back-propagation learning algorithm, and then the network parameters are chosen with trial and error approach. The network's parameters that could produce the output closest to the desired output and low Root Mean Square Error (RMSE) rate during the training phase are used for testing and prediction.

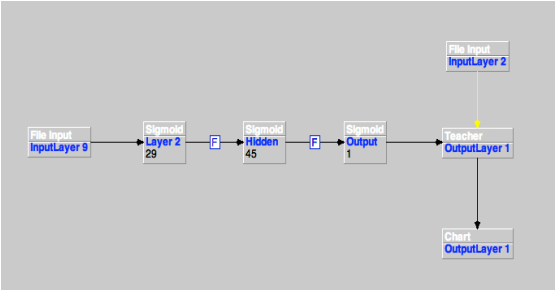

Figure 3.1: Design of the ANN in Joone's GUI

In general, sample data must be collected and processed for the training. The extraneous information is removed and the remaining data is adjusted in the proper format. In the training process, the network is fed with the input vector and desired output several times, allowing the network to recognize the relationship between input and the desired output. Each phase is described in subsections below.

3.1 Data Collection and Preprocessing

Typically, data must be collected from one of nodes in a cluster. To demonstrate how data is collected and preprocessed, we illustrate the proof-of-concept process on a node in our HPC cluster. The configuration of the node is dual 3GHz Intel Xeon processors with 4 GB of memory. The average CPU usage during data collection is above 90%. Data is retrieved periodically with a 10 seconds interval using OpenIPMITool [17]. There are a total of 29 sensors available on the node, which could be classified into 3 categories: 14 voltage sensors, 10 temperature sensors, and 5 fan sensors. The raw data format gathered from OpenIPMITool is *{Sensor name, Sensor value, Unit, Status, lnr, lcr, lnc, unc, ucr, unr}*, as shown in Figure 3.2.

Where: lnr : lower non-recoverable thresholds
lcr : lower critical thresholds
lnc : lower non-critical thresholds
unc : upper non-critical thresholds
ucr : upper critical thresholds
unr : upper non-recoverable thresholds

Baseboard 5VSB	5.076	Volts	ok	na	4.442	4.583	5.452	5.617	na
Baseboard 12V	12.028	Volts	ok	na	10.788	11.160	12.710	13.082	na
Baseboard 12VRM	12.028	Volts	ok	na	10.602	10.912	12.958	13.392	na
Baseboard -12V	-12.040	Volts	ok	na	-13.040	-13.408	-10.744	-10.456	na
Baseboard VBAT	3.180	Volts	ok	na	2.682	2.759	3.518	3.627	na
Baseboard Temp	28.000	degrees C	ok	na	5.000	10.000	60.000	65.000	na
FntPnl Amb Temp	23.000	degrees C	ok	na	5.000	10.000	40.000	45.000	na
Baseard FanBoost	28.000	degrees C	ok	na	na	na	60.000	na	na
ATA BP FanBoost	26.000	degrees C	ok	na	na	na	60.000	na	na
ATA BckPln Temp	26.000	degrees C	ok	na	-10.000	0.000	60.000	65.000	na
FP Amb FanBoost	23.000	degrees C	ok	na	na	na	40.000	na	na
Baseboard Fan 1	11373.000	RPM	cr	na	4080.000	4692.000	na	na	na
Baseboard Fan 2	12291.000	RPM	cr	na	4080.000	4692.000	na	na	na
Baseboard Fan 3	12852.000	RPM	cr	na	4080.000	4692.000	na	na	na
Baseboard Fan 4	12852.000	RPM	cr	na	4080.000	4692.000	na	na	na
Baseboard Fan 5	12291.000	RPM	cr	na	4080.000	4692.000	na	na	na
Processor1 Temp	28.000	degrees C	ok	na	-10.000	0.000	75.000	80.000	na
Processor2 Temp	27.000	degrees C	ok	na	-10.000	0.000	75.000	80.000	na
Proc1 FanBoost	28.000	degrees C	ok	na	na	na	75.000	na	na
Proc2 FanBoost	27.000	degrees C	ok	na	na	na	75.000	na	na
Processor Vccp	1.498	Volts	ok	na	1.030	1.088	1.884	1.942	na

Figure 3.2: Raw data from OpenIPMITool

While collecting data, faults are injected into the system to simulate failure scenarios. The air vents are blocked and some of the fans are stopped in order to block the air circulation and increase the temperature on some specific components of the system.

We then filter raw data to both remove irrelevant information and reformat into a format compliant with Joone. The final data is the following:

$\{x_1 ; x_2 ;\ldots; x_m ; y_1 ; y_2 ;\ldots; y_n ; z_1 ; z_2 ;\ldots; z_o ; \alpha\}$

Where:

X_m is value of voltages sensors (in volts)

y_n is value of temperature sensors (in degree Celcius)

z_o is value of fan sensors (in Round Per Minute(RPM))

α is current state of the system

Currently, it is challenging to determine when an actual failure occurs by simply determining the sensor values and thresholds. Even though the manufacturers pre-set the default thresholds to suggest the conditions where the system should operate. If the system doesn't operate under these conditions, it may behave irregularly or fail. However, some systems may be working normally beyond thresholds. For example, if the critical thresholds temperature is 80 degree Celsius, some systems may fail when the system temperature reaches 95 degree Celsius, some other may fail at 80 degree Celsius. For sake of simplicity, we assume that a system fails when any of the crucial sensors violate the critical thresholds which existing FT policies are normally deployed [13][14].

We define α as the current state of the system, which is enumerated by determining the system status. It could only be two possibilities: "1" for failed system, and "0" for working system. We eliminate data from the fan sensors since the fan fault is unlikely to cause directly a system failure, so if this leads to a failure, it may be triggered by other sensors (*e.g.*, temperature sensor). In other words, if any of the sensors goes beyond or drop below the critical thresholds, the state of the system is set to 1, otherwise 0:

$$\alpha = \begin{cases} 1 \text{ ; } x_n, y_n > \text{ucr} \\ \text{ ; } x_n, y_n < \text{lcr} \\ 0 \text{ ; otherwise} \end{cases}$$

Since the ANNs expect input and desired output to be in range 0 and 1, all data must be normalized before being processed for network training. We use Equation (1) for data normalization technique.

$$N(X_i) = (X_i - X_{min}) / X_{max} \qquad (1)$$

It is also important to note that we only collect sensor data via IPMI for demonstration purposes. However, the data collected from similar mechanisms (e.g. SMART for disk failure prediction) could also be processed and fed the network.

3.2 Training

The network is trained with back-propagation algorithm. The back propagation is the most widely used algorithm to train neural networks. The training is based on a simple concept: if the network gives a wrong answer, then the weights linked between neurons are corrected so that the error is lessened and as a result, future responses of the network are more likely to be correct [1].

In this step, the network is trained with the normalized x_n, y_n and z_n as input vectors and the future system state (α_{t+1}) as the desired output as illustrated in Figure 3.3.

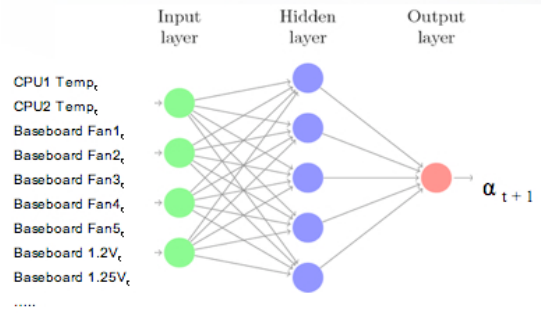

Figure 3.3: Training network with input and desired output

4. Evaluation and Discussion

In our experiment, we start collecting IPMI data with OpenIPMITool with 10 seconds interval. The initial training data has total of 360 entries. During data collection, we inject faults into the system by stopping the system's fans and block CPUs air vent to increase the temperature of CPU1 and CPU2 respectively (we use dual processor node). Furthermore, we include simulated fault positive data of CPU temperature to the training data to train the network to recognize the fault positive events. Data is normalized and transformed into the Joone readable format. During the training, we choose learning rate = 0.2, momentum = 0.4, epochs = 1000, and training pattern = 360.

After the network is trained, we use the network to predict larger data sets. The testing data set contains 3000 records, which consist of normal, actual faults and fault positive events. We use the same method to inject faults on CPU1 and CPU2.

We expect output to be 2 possible values; fail (1), or not fail (0). The output from the ANN, however, is range from 0 to 1, therefore, if the output is greater than 0.5, we consider the system is going to fail (1) and if the output is less than 0.5, the system is not going to fail (0)

In Figure 4.1, we show the results of ANN prediction once we increase the temperature of CPU1 beyond the critical threshold (at 0.8).

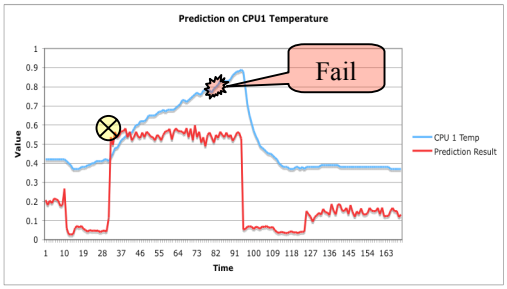

Figure 4.1: CPU1 temperature rising above the threshold

Similarly, with CPU2 temperature, the result from the ANN could raise up before the actual temperature reach the failing point (see Figure 4.2).

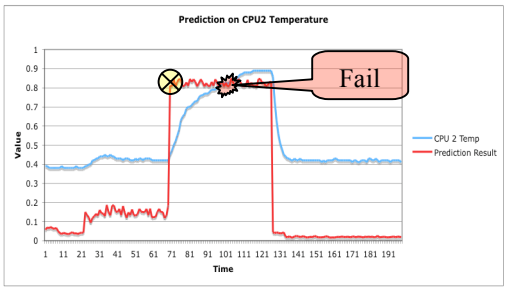

Figure 4.2: CPU2 temperature rising above the threshold

Interestingly, we have observed from previous 2 experiments, that the network determines the output from the combination of available sensor values. For example, in both Figure 4.1 and 4.2, the output from the network raised above 0.5, when more than one fans are stopped.

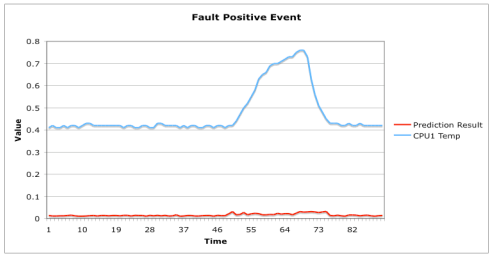

Figure 4.3 False positive event

In Figure 4.3, we simulate false positive event of temperature from CPU1. The CPU1 temperature is raised very near the critical threshold. Other components are in the normal operation (*e.g.*, no fan is stopped). The output from the network slightly increases from the normal when the fault positive event occurs but not high enough to pass 0.5. In contrast, the same event with the threshold trigger approach, once the CPU1 temperature increase over the certain point near the critical threshold (typically, 0.7 - 0.75), the alarm is generated and actions might be taken to evade the computing entity.

5. Conclusion and Future work

In this paper, we present our on-going research on fault prediction in HPC environments with artificial neural network. We built a feed-forward network and trained it based on a back-propagation algorithm with historical IPMI data collected from our live system. The network shows promising prediction results. Based on the study in this paper, an intelligent fault predictor could be implemented in the proactive fault avoidance framework to improve fault prediction accuracy and reduce the number of fault alarms.

The major concern of fault prediction with ANN approach is performance. The network must be periodically trained with new data sets. We are continuing to study ANN with different techniques to reduce the overhead. One possibility would be using the ANN as fault recognition; the network is trained with fault data set to recognize the fault patterns. The network is then expected to identify whether the given input matches the previous fault patterns or not. Once the network fails to recognize the failure, it is retrained with the new fault data set. Furthermore, the irrelevant data in the training data should be removed in order to reduce number of inputs and number of neurons in the hidden layer for faster network training.

Additionally, we study fault prediction using ANN based on a uni-variable model. In the future research, we could verify the performance based on multi-variables with actual data or simulation.

Acknowledgement

We thank Dr.Vir V. Phoha, a computer science professor at Louisiana Tech University for his help and valuable discussions during this research.

References

[1] J. E. Dayhoff, Neural Network architectures. An introduction, International Thompson Publishing 1990.

[2] Kryzanowski L., Galler M., Wright D.W., "Using Artificial Network to Pick Stocks", Financial Analyst Journal, August 1993, pp. 21-27

[3] Refenes, A.N., Zapranis, A., Francis, G., "Stock Performance Modeling Using Neural Networks: A Comparative Study with Regression Models", Neural Networks, vol. 7 No. 2, 1994, pp 375-388

[4] Schoeneburg, E., "Stock Price Prediction Using Neural Networks: A Project Report", Neurocomputing, vol.2, 1990, pp. 17-27

[5] Wong, F.S., Wang, P.Z., Goh, T.H., Quek, B.K., "Fuzzy Neural Systems for Stock Selection", Financial Analyst Journal, January-February 1992, pp. 47-53

[6] A. Refenes and M. Azema-Barac, "Neural Network Applications in Financial Asset Management" Neural Computing Application, vol.2, pp.13-29, 1994

[7] Moody J., "Forecasting the Economy with Neural Nets: A Survey of Challenges and Solutions". In Orr G.B and, Muller K.-R. (Eds.) Neural Networks: Tricks of the Trade, pp. 347-371, Springer, Berlin, 1998

[8] Atienza F. et al. "Risk Stratification in Heart Failure Using Artificial Neural Networks", Research paper, Cardiology Department, University General Hospital, Valencia, 2000. Available at http://www.amia.org/pubs/symposia/D200367.PDF

[9] Palacios R. and Gupta A. "Training Neural Networks for Reading Hand Written Amount on Check", Working paper 4365-02, MIT Sloan School of Management, Cambridge, Massachusetts, 2002. Available at http://ssm.com/abstract_id=314779

[10] D. Park, M. El-Sharkawi, R. Marks, L. Atlas, and M. Damborg, "Electric load forecasting using an Artificial Neural Network", IEEE Trans. Power Syst. Vol. 6, pp 442-449, May 1991

[11] Wen Tian, Ben Choi, and Vir Phoha, "An Adaptive Web Cache Access Predictor Using Neural Network," Developments in Applied Artificial Intelligence, IEA/AIE 2002, Lecture Notes in Artificial Intelligence, Vol. 2358, pp. 450-459, 2002.

[12] Lawrence, S. Giles, C.L., Tsoi, A.C., Back, A.D., "Face Recognition: A Convolutional Neural Network Approach", IEEE Transaction on Neural Network, 8, 98-113.

[13] G. Vallee, K. Charoenpornwattana, C. Engelmann, A. Tikotekar, C. Leangsuksun, T. Naughton, S.L. Scott, "A Framework For Proactive Fault tolerance", In proceedings of the Third International Conference on Availability, Reliability and Security (ARES 2008), Barcelona, Spain,

[14] K. Charoenpornwattana, C. Leangsuksun, G. Vallee, A. Tilotekar, S.L. Scott, "A Scalable Unified Fault Tolerance Framework", submitted to International Conference on Distributed computing System (ICDCS 2008), Beijing China.

[15] A. Nagarajan, F. Mueller, C. Engelmann, S. L. Scott, "Proactive Fault Tolerance for HPC with Xen", in ICS '07: of the 21st annual international conference on Supercomputing. New York, NY, USA: ACM Press, 2007, pp. 23–32.

[16] Joone : http://www.joone.org

[17] OpenIPMITool : http://ipmitool.sourceforge.net

[18] R. Oldfield, "Investigating lightweight storage and overlay network, for fault tolerance," in HAPCW'06: High Availability and Performance Computing Workshop. Santa Fe, New Mexico, USA: Held in conjunction with LACSI 2006, OCT 2006.