FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 6 No. 6, December, 2022, pp 289 - 295
DOI: https://doi.org/10.33003/fjs-2022-0606-1094

A FRAMEWORK FOR PROCESS-RESOURCE REQUIREMENTS PREDICTION IN A FACTORED OPERATING SYSTEMS USING MACHINE LEARNING

[*1]Habila Mikailu, [2]Agaji Iorshase, [3]Blamah Nachamada and [4]Ishaya Gambo

[1]Department of Computer Science, Nigeria Army University Biu, Nigeria
[2]Department of Computer Science, Federal University of Agriculture Makurdi, Nigeria
[3]Department of Computer Science, University of Jos, Jos, Nigeria
[4]Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria

*Corresponding authors' email: mikailu.habila@naub.edu.ng

ABSTRACT

The evolution of multicore technology has come with new challenges to process scheduling. An intelligent process resource requirement prediction framework in a factored operating system (FOS) was developed using machine learning. To simulate this system, an array of positive integer numbers in the interval of 1 to 10, with a maximum length of 20, was used to represent process. The properties of this array (array length and sum of the array elements) were used as the parameters of the application processes. These served as input into the machine learning network to predict the process' size and hence, the resource requirements for each process. The framework was simulated using Java 2EE. Experimental results of the framework showed that prediction of processes' sizes with enhanced resource requirement allocation, and hence, required faster and efficient scheduling of processes to multiple resource cores, with an increased system throughout.

Keywords: Machine Learning, Multicore, Resource Prediction, Factored Operating System, process scheduling

INTRODUCTION

Job processes and threads in an operating system (OS) gain access to system resources by scheduling algorithms. The requirement to execute multiple processes at a time with the present-day systems and concurrent execution is the propelling force behind scheduling algorithms. Distinct scheduling methods have emerged, among which several have been used to overcome many problems in scheduling, such as which process to give processor time. Among these methods applied in tackling scheduling problems are the branch and bound, critical path methods, priority rules, dynamic programming, shortage job first, etc., to determine which process gets the processor time (Silberschatz et al., 2013).

According to Koya (2017), the traditional scheduling algorithms, in theory, are First-Come-First-Serve (FCFS) Scheduling, Round Robin (RR) Scheduling, Priority Scheduling, Shortest Job First (SJF) Scheduling, Shortest Remaining Time First (SRTF) Scheduling, Multilevel Queue Scheduling (MQS) and multilevel Feedback Queue (MFQ) Scheduling. These scheduling algorithms are either preemptive or non-preemptive.

According to Amur et al. (2008), the time expended by a process executing on a processor is quantified as burst length. Estimating the next CPU burst of a process is complex and requires much effort and calculations. The lengths of the next CPU burst can mathematically be predicted using a traditional method known as "Exponential Averaging," which uses the process history to estimate the next processor time (Silberschatz et al., 2013).

In Grids Computing, where multi-level platforms from multiple locations provide a wide range of services to reach a common goal, scheduling tasks is considered a critical component of a Grids Computing infrastructure. On this platform, administrative domains are searched to allocate a task to a machine or, on the contrary, schedule tasks to several resources at one or more sites (Shah et al., 2010).

Scheduling in the grid consists of three main phases: discovering available resources phase, apportioning and attributing tasks to executable resources phase, and finally, the tasks executing phase. Dedicated machines are usually used to apportion tasks to executable resources in Grids Computing due to their addiction to the duration of processor time. Consequently, a dedicated machine must be able to estimate the succeeding processor time duration in Grids Computing scheduling techniques (David, 1998).

Helmy et al. (2015) proposed a Machine Learning (ML) based approach to estimate the length of the CPU bursts for processes. The proposed approach aimed to select the most significant attributes of the process using feature selection techniques and then predict the CPU burst for the process in the grid. ML techniques such as Support Vector Machine (SVM) and K-Nearest Neighbors (K-NN), Artificial Neural Networks (ANN), and Decision Trees (DT) were used to test and evaluate the proposed approach using a grid workload dataset named "GWA-T-4 AuverGrid". The experimental results show a strong linear relationship between the process attributes and the burst CPU time. Moreover, K-NN performs better in nearly all approaches regarding CC and RAE. Furthermore, applying attribute selection techniques improved the performance in terms of space, time, and estimation.

As the scalability of the processor core slows, ascribable to Moore's law which speeds up research in modern computing architecture, a demand exists to consider new OS models. Wentzlaff et al. (2011) described FOS as an OS for 1000 cores, executing each task on a separate core. Their idea was concerning the future abundant processor cores, such that devoting a core to a single process will maximize the benefits of multicore technology. According to the authors, the scheduling method may take a different approach from the current traditional method. They affirmed that scheduling in FOS becomes a space-sharing approach as opposed to the time multiplexing of the conventional OS.

According to Wang et al. (2019), space sharing is when multiple tasks run on physical resources simultaneously. Still, the resources are physically divided into multiple partitions

that the tasks execute concurrently, as opposed to time-sharing, where processes run simultaneously. Still, only one process runs on the shared resources at a time. Laplante & Milojicic (2016) also envision that by 2030 OS will be created using a new technology paradigm that would be nearly unrecognizable today. Formulating better scheduling techniques is a central and most profound knowledge of process execution behavior that is very useful. Negi & Kishore (2004) studied the approaches to the process characterization problem. They also proposed a novel method to characterize and categorize process execution behaviors using machine learning techniques that learn from previous execution instances of programs.

To predict the time the programs were executed,24 attributes were selected and used. The output of the estimation range of 91.4% to 99.7% was attained. Portable batch system scheduling was improved with these results, where the requirements of resources were predicted based on the developed knowledge base that kept track of the former program executions. Negi & Kishore (2005) presented a viable idea that included machine learning methods to improve process scheduling. This was achieved by allocating varying timeshares to distinct processes to decrease the overhead cost of context-switching. A special timeshare in the form of an integer filed was linked to processes that assisted in showing the preferred estimation of processor time to be apportioned to reduce their turnaround time. Processes were arranged into classes of distinct special timeshares, each giving a time interval of 50ticks. 5 sets of standard programs were selected, having distinct instances to function with distinct special timeshare numbers to ascertain the littlest time for each execution. Each example of data incorporated process parameters combined into a single data set. Their result showed a noticeable reduction in the various processes' execution time. However, the problem with this method is that whenever a new program arrives, there is an overhead of classifying it separately and feeding that data to the Kernel. Notwithstanding, the work is a stepping stone in a modern OS.

Ojha et al. (2015) applied the use of Linux scheduler history to estimate the parameters of processes. These include execution time and, context switch counts. Performance preemption was made better by the scheduler as the processor's overhead time due to preemption and context-switching was reduced. The parameters used include StrTab, RoData, Hash, DynSyn, etc., that produced a Decision Tree for classifying instances into 20 class time rates. Their WEKA tool simulated the classifier and applied a self-learning module based on Reinforcement Learning. The result obtained from the classifier served as input to the module. As the Decision Tree was busy classifying new processes, the module, on the other hand, was examining new classes from the background. The scheduler's decision in allocating timepieces was modeled using Markov Chain. Their study confirms the suggestion that reducing the context switch count can significantly cut down the execution time as relevant time slices were assigned to each process in a class. This self-learning module empowered the system always to make the classification of processes better.

Siddha et al. (2007) examined several scheduling techniques for multicore platforms under distinct load settings and related tradeoffs. They asserted that the challenges facing process schedulers are identifying and predicting the resource requirements of each job and scheduling them in a manner that reduces shared resource contentions. In their testing and analysis, a dual-package symmetric multi-processor platform was considered. Each package was assigned two cores sharing a 4MB last-level cache. The experiment result demonstrated a uniform distribution of loads among the packages. The core speed was increased, resulting from dynamic speed up, and optimal performance was achieved.

Schartl (2016) identified locks, poor locality for sharing processor cores, and cache coherent shared memory as OS design challenges for multicore architectures. The author proposed lock avoidance to mitigate the lack of the system's scalability. The application and OS were split up to avoid sharing of cores. Each core was dedicated to every job on a system. Also, the author proposed using message passing for communication instead of cache-coherent shared memory. This message passing was incorporated so that system performance could be enhanced. However, designing an OS with locks that proffers good structural scalability and load balancing remains a challenge because of the technological advancement of multicourse.

Hardware technology is advancing, and in the near future, scheduling will not be based on processes competing for scarce resources but rather on selecting a suitable resource from the abundant system resources. Thus, this work aimed at building an intelligent system that predicts the processes' sizes based on resource requirements and selects the most appropriate resource(s) using machine learning techniques.

This research work proposed a multilayer perceptron machine learning algorithm that could be deployed in a factored operating system environment to evaluate application processes and predict their sizes with respect to resource requirements. The design of an intelligent architectural framework for the proposed system is beneficial in ensuring that the abundant processor cores provided by the advancement in multicore technology would be adequately harvested

METHODOLOGY

A multilayer perceptron neural networks method is used to develop the framework. The aim is to develop an intelligent prediction system that can predict process size and resource (CPU) requirements in a multicore environment. Figure 1 depicts the architecture of the proposed framework. In building this framework, a multi-agent software development idea was used as a distributed system representing processor cores and a machine learning model as autonomous intelligent agents.

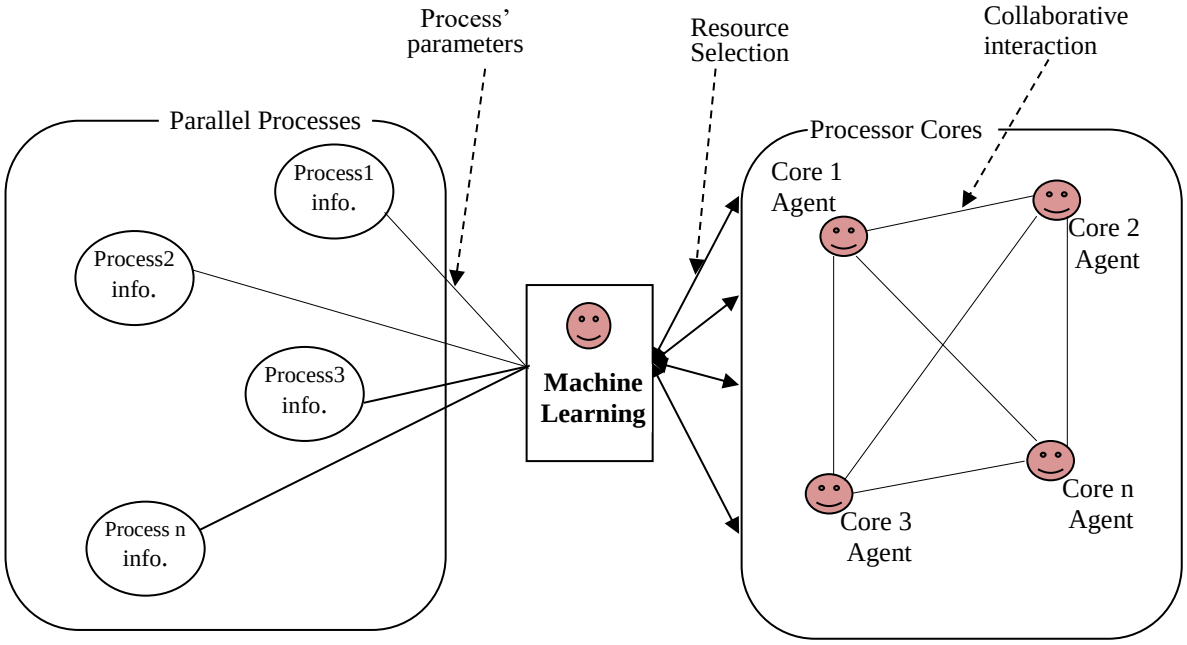

Figure 1: The Proposed Architecture of the Process-Resource Prediction

The parameters of the processes were retrieved, and the processes' sizes concerning the degree of resource requirements were predicted using a machine learning algorithm. The significant components of this architecture are the parallel application processes 1 to n, (n>=1) requesting for spatially distributed resources 1 to m (m>=1) represented by agents that are allocated using machine learning. Machine learning determines the size of each process using its parameters. It then collaborates with resource agents to know their various capacities.

Thus, on acquiring the needed information from both process jobs and the resource cores, the agent predicts the required or suitable resource core for each process job. In simulating this framework, an array of positive integer numbers in the interval of 1 to 10, with a maximum length of 20, was used to represent processes. Two attributes of the array (array length and sum of the array elements) were used to describe the process resource requirements.

Machine learning using a backpropagation algorithm is used for process size predictions. Figure 2 depicts a detailed back propagation model of the proposed system. It shows an explicit model of a full-scale network with $\{p_1, p_2\}$ as input categories. Also, two hidden layers with three and two numbers of neurons, respectively, as well as one output neuron, as depicted in the neural network architecture of figure 2. The input data is normalized using Equation 1.

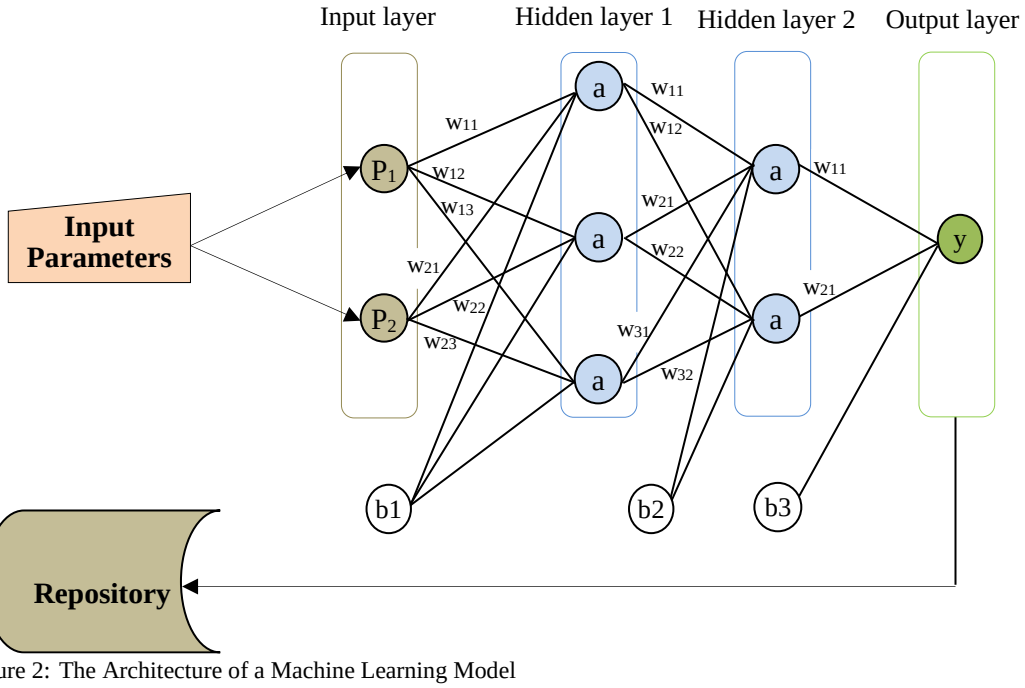

Figure 2: The Architecture of a Machine Learning Model

$$p = \frac{(p-min1)(p-min2)}{(max1-min1)} + min2 \quad (1)$$

where p is the input parameter of the processes
On the other hand, the output is scaled within the range of 0 and 1 using equation 2.

$$y = \frac{p-minp}{maxp-minp} \quad (2)$$

Where y is the output value of the network
Equation 3 is the Tan Activation Function used in the forward pass training of the network.

$$\tanh(p) = \frac{e^{p}-e^{-p}}{e^{p}+e^{-p}} \quad (3)$$

Equation 4 is the Mean Square Error (MSE) used to compute the difference between the estimated and what is estimated in training the network to improve prediction accuracy.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\bar{y}i - yi)^2 \quad (4)$$

Where $\bar{y}i$ is the predicted value, and yi is the targeted value of the machine learning network.
A partial differential equation to the cost function (square error) of equation 5 is used in the backward propagation algorithm network to measure the distance between the predicted value and the actual value.

$$\text{Cost Function (CF)} = j(w) = \frac{1}{2}\sum_{i=1}^{n}(\bar{y}i - yi)^2 \quad (5)$$

Equation 6 is used in updating the weights in the Back Propagation Algorithm using Stochastic gradient descent:

$$Wl = W - \alpha\frac{1}{n}\frac{\partial j(w)}{\partial wl} \quad (6)$$

Where wl is a new weight, w is the weight, α is the learning rate, and j is the cost function.
The backpropagation algorithm model employed as the basis for the learning rule is defined as:

i. Small random values between -1 and 1 were initialized for the weights.
ii. Input parameters of processes are used as the training vector input.
iii. The input signal is propagated forward through the network to get the first output and thus compare the machine learning result with the target result.
iv. The learning rate value used for the network training is 0.1and a bias value of 1.0.
v. Errors are computed using Mean Square Error (MSE) method by taking the square errors from the output neurons and running them back through the weights to obtain the hidden layer errors.
vi. Repeat steps iii to v; by this method, our network of three layers will be trained.

The Unified Modeling Language Sequence Diagram for Process-Resource Prediction

Figure 3 shows the UML sequence diagram of a process-resource requirements prediction framework. The diagram shows the interactions between the proposed framework's various agents (objects), that is, the machine learning agent, process jobs, and the processor core agents.

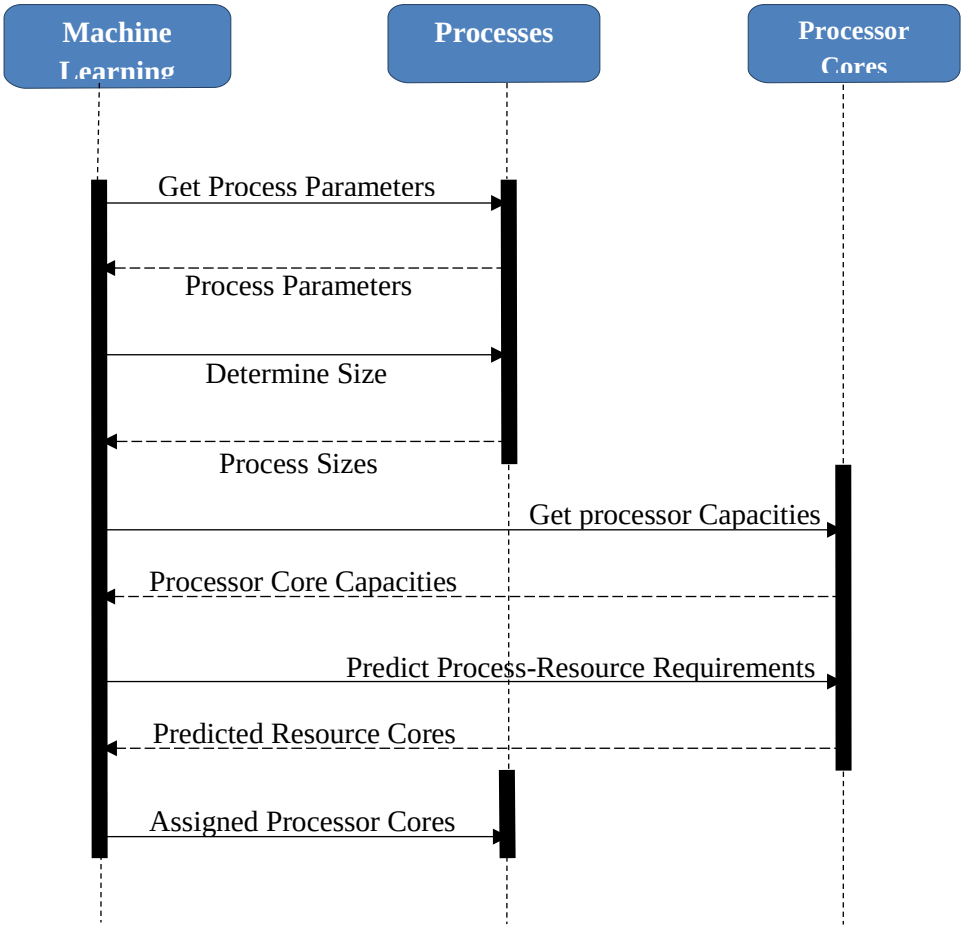

Figure 3: Process-Resource Requirements Prediction

The machine learning algorithm was used to extract the parameters of the process jobs, which in turn predetermined the size of each process in terms of resource requirements. Moreover, it then delves into the available Processor core resources to get the knowledge of their various capacities. On reaching the needed information from both the process jobs and the processor cores, the machine learning algorithm predicted the most efficient resource cores for the various process jobs seeking processor time. The framework was simulated using NetBeans IDE 7.2.1, MySQL Query Browser Development, and JBOSS 4.2.2.GA Server, and a Macromedia Dreamweaver 8.0. In simulating this framework, 2000 parallel process jobs were used to test the efficiency of the proposed system on 5000 processor cores.

RESULT AND DISCUSSION

Result

Figure 4: The Training and Prediction Page

Table 1: The Un-normalized Training Case

ANN TRAINING CASE			
S/NO	**Array Length**	**Array Element Sum**	**Targeted Output**
1	15	85	20
2	18	115	24
3	5	16	6
4	14	63	18
5	13	77	17
6	12	68	16
7	5	24	7
8	12	57	15
9	13	65	17
10	4	22	6

Table 2: The Normalized Machine Learning Training Case

NORMALIZED TRAINING CASE			
S/NO	**Array Length**	**Array Element Sum**	**Targeted Output**
1	4.5	2.55	0.666667
2	5.4	3.45	0.8
3	1.5	0.48	0.2
4	4.2	1.89	0.6
5	3.9	2.31	0.566667
6	3.6	2.04	0.533333
7	1.5	0.72	0.233333
8	3.6	1.71	0.5
9	3.9	1.95	0.566667
10	1.2	0.66	0.2

Table 3: Artificial Neural Network Training Results

S/NO	Array Length	Array Element Sum	Targeted Output	ANN Result
1	4.5	2.55	0.666667	0.5317926812
2	5.4	3.45	0.8	0.5325738518
3	1.5	0.48	0.2	0.3700710488
4	4.2	1.89	0.6	0.5304526304
5	3.9	2.31	0.566667	0.5305195703
6	3.6	2.04	0.533333	0.5288742535
7	1.5	0.72	0.233333	0.4022394741
8	3.6	1.71	0.5	0.5276272429
9	3.9	1.95	0.566667	0.5297299907
10	1.2	0.66	0.2	0.3346436349

11	2.4	1.11	0.333333	0.5004117889
12	1.5	0.99	0.233333	0.4307023676
13	2.1	1.47	0.333333	0.4995388084
14	0.9	0.6	0.166667	0.2320646768
15	2.4	1.23	0.366667	0.5036166943
16	1.8	1.02	0.266667	0.4629974038

Table 4: The Results of the Proposed Process-Resource Requirement Prediction

S/N	Job Id	Job Size	Performance Function.
1	I92B3	21	29.0
2	I92B3	29	36.0
3	I92B3	26	32.0
4	I92B3	28	34.0
5	I92B3	28	34.0
6	I92B3	16	28.0
7	I92B3	30	38.0
8	I92B3	29	36.0
9	I92B3	19	29.0
10	I92B3	6	25.0
11	I92B3	32	40.0
12	I92B3	30	38.0
13	I92B3	30	38.0
14	I92B3	28	34.0
15	I92B3	31	40.0
16	I92B3	24	31.0
17	I92B3	31	40.0
18	I92B3	25	31.0
19	I92B3	30	38.0
20	I92B3	31	40.0

Discussion

The ANN machine learning shows the inputs (the array length and sum of array elements) to the network and the targeted output. The ANN operation button is where the input to the network is made, as shown in Figure 4. As Figure 4 reflects, the add button is used to get the input data into the network before it is trained. The training command enables the network to be trained, and the prediction test command predicts the application process size. The core operation button allows input of the core's parameters which are then computed to determine the capacity of the available processor cores. The core capacity button displays the available processor cores in the system and their respective performance function. The result button displays the result of the proposed process-resource requirements predictions. The training case button gives the training data sets for both the normalized and the un-normalized data sets.

Each normalized value representing the array length and sum of array elements was obtained using the expression in equations 1 and 2. The activation function of equation 3 is then applied to the normalized data during the training process to segregate the relevant data from the noisy ones. On getting the output, the difference between the target and predicted values was computed using the MSE of equation 4, which improved the accuracy of the prediction. The cost function of equation 5 measured the distance between the predicted and the actual values. Hence, the gradient descent of equation 6 was applied to gain a sense of direction to the model to reduce errors and achieve convergence.

On getting the input data to the network, the raw data was normalized before the training and the prediction activities. Observing the un-normalized training data in Table 1, the array length of 15, the sum of an array element of 85, and target output of 20 were normalized to 4.5, 2.55, and 0.666667, respectively, using equations 1 and 2 as shown in Table 2.

The neural network was successfully trained, and the framework was used with data comprising jobs of different sizes and processor cores of various capacities. Network testing was necessary to enhance the validation accuracy of machine learning for the actual prediction of process-resource requirements. The network was tested using 200 data items, as depicted in Table 3. The input fields contain the normalized values of the process parameters of the test data. The target output constitutes the thought-over output for each observation. The ANN machine learning result column comprises the system output for each test run of data.

Table 4 shows the result of the first 20 process-resource requirements prediction. It is observed from the sampled result and, by extension, the entire result that process job sizes range from a maximum of 32KB to a minimum of 6KB. On the other hand, the processor cores range from a maximum of 40GHz to a minimum of 25GHz in capacity. The result of the simulated framework showed that higher processor cores were selected and allocated to process jobs with higher resource requirements. It is seen that a process job size of 32KB was allocated to the largest processor core of 40GHz in capacity, also 31KB to 40GHz, 30KB to 38GHz down to the least 6KB to 25GHz. Process jobs of 32KB and 31KB were allocated the same 40GHz because there were still 40GHz processor cores; after that, the higher process jobs of 32KB were exhausted. This was continued until the entire process jobs were allocated in the most suitable way.

CONCLUSION

Thus, the framework is efficient and accurate in predetermining the process job resource requirement and predicting the most optimized processor core in a multicore

factored OS environment. This framework is more adaptable because of the advancement in multicore technology as opposed to the work of Helmy et al. (2015), which predicted the CPU burst but was limited to the traditional scheduling approach and blind to the growth in multicore technology. This intelligent framework is scalable, adaptable, and positioned to enhance the overall system performance throughput.

The framework is recommended for use because of its ability to predict the process-resource requirements in an optimized fashion. Moreover, with the current growth in multicore technology and the user demands on system resources, the framework could adequately adapt and utilize the abundant processor cores and improve the overall system performance. In future work, the efficiency of the proposed framework will be tested on scheduling in a factored OS, where process jobs execution on processor cores is timed, to validate the improvement in system throughput since this framework limited to demonstrate.

REFERENCES

Amur, H. R., Gautham, R. S., Dipankar, S. & Srivatsa, V. (2008). Plimsoll: a DVS Algorithm Hierarchy. https://www.academia.edu/7901429/Plimsoll_a_DVS_Algorithm_Hierarchy.

Courtial, F. (2017). Deep Neural Networks from Scratch. Retrieved on 7th July, 2019, from https://matrices.io/deep-neural-network-from-scratch/

David, F. (1989). Allocating modules to processors in a distributed system.*IEEE Transactions, 15*(11): 1427-1436.

Helmy, T., Al-Azani, S. & Bin-Obaidellah, O. (2015). A Machine Learning-Based Approach to Estimate the CPU-Burst Time for Processes in the Computational Grids.*In Third International Conference on Artificial Intelligence, Modelling and Simulation*. Retrieved on 19th July, 2019, fromhttp://uksim.info/aims2015/CD/data/8675a003.pdf

Koya, B. K. (2017). *An Interactive Tutoring System to Teach CPU Scheduling Concepts in an Operating System Course*. MSc. Thesis. Computer Science and Engineering. Wright State University, India). Retrieved from https://corescholar.libraries.wright.edu/cgi/iewcontent.cgi?article=2885&context=etd_all.

Laplante, P. & Milojicic, D. (2016). Rethinking Operating Systems for Rebooted Computing.*IEEE International Conference on Rebooting Computing (ICRC)*. Retrieved from https://ieeexplore.ieee.org/abstract/document/7738695/authors

Negi, A. & Kishore, K. P. (2005). Applying machine learning techniques to improve Linux process scheduling. *In TENCON IEEE*, Region 10, pages 16. Retrieved on 19th May, 2019, from http://alumni.cs.ucr.edu/~kishore/papers/tencon.pdf

Negi, A. & Kishore, K. P. (2004). Characterizing Process Execution Behaviour Using Machine Learning Techniques. *In DpROMWorkShop Proceedings, HiPC International Conference.* Retrieved on 12th July, 2019, from http://alumni.cs.ucr.edu/~kishore/papers/hipc.pdf

Ojha, P., Siddhartha, R. T., Vani, M. & Mohit, P. T. (2015). Learning Scheduler Parameters for Adaptive preemption. *Journal of Computer Science & Information Technology*. Retrieved on 18th, February, 2019, from DOI: 10.5121/csit.2015.51513

Schartl, A. (2016). Design Challenges of Scalable Operating Systems for Many-Core Architectures. Retrieved on 7th April, 2018, from http://www4.cs.fau.de/Lehre/WS16/PS_KVBK/slides/slides-schaertl.pdf

Shah, M., Nasir, S., Mahmood, A. K. & Oxley, A. (2010). Analysis and evaluation of grid scheduling algorithms using real workload traces. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, pp. 234-239.ACM.

Siddha, S., Pallipadi, V. & Mallick, A. (2007). Process Scheduling Challenges in the Era of Multi-core Processors. *Intel Technology Journal*, 11(4): 360-369.Retrieved on 14th October, 2018, from DOI: 10.1535/itj.1104.09

Silberschatz, A., Galvin, P. B. & Gagne, G. (2013). *Operating System Concepts*. John Wiley and sons, Inc. USA.

Wang, Y., Li, L., Wu, Y., Yu, J., Yu, Z. & Qian, X. (2019). TPShare: A Time-Space Sharing Scheduling Abstraction for Shared Cloud via Vertical Labels. *ISCA '19: ACM Symposium on Computer Architecture, Phoenix, AZ.*ACM, New York, NY, USA. Retrieved on 10th September, 2019, from https://doi.org/10.1145/1122445.1122456.

Wentzlaff, D., Gruenwald, C., Beckmann, N., Modzelewski, K., Belay, A., Kasture, H., Youseff, L., Miller, J. & Agarwal, A. (2011). Fleets: Scalable services in a factored operation system. *Computer Science and Artificial Intelligence Laboratory Technical Report, Massachusetts Institute of Technology, Cambridge, Ma 01239 USA.*