Received December 15, 2019, accepted January 1, 2020, date of publication January 15, 2020, date of current version January 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2966678

Cloud Infrastructure Estimation and Auto-Scaling Using Recurrent Cartesian Genetic Programming-Based ANN

QAZI ZIA ULLAH[1,2], **GUL MUHAMMAD KHAN**[3], **AND SHAHZAD HASSAN**[1]
[1]Computer Engineering Department, Bahria University, Islamabad 44000, Pakistan
[2]Electrical and Computer Engineering Department, COMSATS University Islamabad, Attock 43600, Pakistan
[3]NCAI, Electrical Engineering Department, Peshawar University of Engineering and Technology, Peshawar 25120, Pakistan

Corresponding author: Qazi Zia Ullah (zia_comsian@yahoo.com)

This work was supported by HEC through Bahria University Islamabad.

ABSTRACT Use of cloud resources has increased with the increasing trend of organizations and governments towards cloud adaptation. This increase in cloud resource usage, leads to enormous amounts of energy consumption by cloud data center servers. Energy can be conserved in a cloud server by demand-based scaling of resources. But reactive scaling may lead to excessive scaling. That, in turn, results in enormous energy consumption by useless scale up and scale down. The scaling granularity can also result in excessive scaling of the resource. Without a proper mechanism for estimating cloud resource usage may lead to significant scaling overheads. To overcome, such inefficiencies, we present Cartesian genetic programming based neural network for resource estimation and a rule-based scaling system for IaaS cloud server. Our system consists of a resource monitor, a resource estimator and a scaling mechanism. The resource monitor takes resource utilizations and feeds to the estimator for efficient estimation of resources. The scaling system uses the resource estimator's output for scaling the resource with the granularity of a CPU core. The proposed method has been trained and tested with real traces of Bitbrains data center, producing promising results in real-time. It has shown better prediction accuracy and energy efficiency than predictive scaling systems from literature.

INDEX TERMS Artificial neural networks, auto-scaling, cartesian genetic programming, energy efficiency, evolutionary computation, green computing, infrastructure as service, workload prediction, cloud server.

I. INTRODUCTION

Cloud computing is a paradigm for providing access to a shared pool of computing resources (e.g. servers, networks, storage, and applications). Cloud delivers resources with the least possible service provider management effort and intervention [1], [2]. Since the inception of cloud computing, the use of cloud computing services increases as reported by Gartner's chief information officers (CIOs) survey.[1] 2018 (i.e. conducted from 3,160 CIOs across 98 countries). According to a survey.[2] in 2018 (by 451 Research), sixty per cent of enterprise workloads will run on cloud and infrastructure as a service (IaaS) demand will have the highest growth. A survey.[3] (released by North Bridge Venture Partners on 19th June 2014), states that fifty-six per cent of businesses are using IaaS technologies to harness elastic computing resources. The survey further reports that over eleven thousand cloud services and Application Program Interfaces (APIs) are currently in use by the cloud customers, and the tendency is towards everything as a service in future. According to International Data Corporation (IDC) report.[4] published on 18th January 2018, the global spending on public cloud services and infrastructure is estimated to increase from $160 billion in 2018 to $277 billion in 2021 that shows a compound annual growth rate (CAGR) of 21.9%. It is expected that by 2020, internet traffic generated per second will be 51,974 GigaByte [3]. The tendency towards

The associate editor coordinating the review of this manuscript and approving it for publication was Vivek Kumar Sehgal.

[1]https://www.gartner.com/newsroom/id/3847965
[2]https://451research.com/blog/764-enterprise-it-executives-expect
[3]http://www.northbridge.com/cloud-computing
[4]https://www.idc.com/getdoc.jsp?containerId=prUS43511618

everything as a service, increase in demand for IaaS services, the growing inclination of governments and enterprises towards clouds, and an enormous increase in internet traffic will result in the generation of gigantic volumes of data being processed by data centers' servers (hosts) [4].

This increase in the use of cloud services leads to increase the energy consumption by cloud data centers [5]. The worldwide increase in energy consumption by data centers was reported 1.1-1.5 per cent, while for the US, it was from 1.7-2.2 per cent in 2010 [6]. A significant increase in the average power (kW) and utilization of data center servers has been reported in [7], from 2006 to 2020. With an increase in the usage of a data center, it requires the cooling system to consume more electricity, i.e. up to 30 per cent [8]. Research has shown that the power consumed by a data center is directly proportional to its resource utilization [9]. Energy consumption is one of the main issues of cloud data centers [10]. With the increase in energy consumption, the cost of cloud services increases for both cloud providers (in terms of electricity bills) and customers (service purchase) [11]. Also, with an increase in energy consumption, more heat is dissipated, and more carbon is emitted in the environment [12]. Compensating heat dissipation requires extra cooling equipment installations that further increases the cost of cloud services [13]. As cloud services, demand increases the cloud owners/providers business increases, but the quality of service (QoS) to users degrades [14]. Because with the same limited infrastructure, a large number of workloads cannot run properly. In such situations, the resource management problem will be the main issue to be adequately solved. Without optimized resource management, QoS cannot be provided to users [9]. A large number of workloads on clouds will have a challenging impact on the performance of the cloud data center infrastructure [15].

In a data center, about 40% of total energy is consumed by information technology (IT) equipment that includes communication links, switching, and aggregation elements and servers [16]. The energy consumed by data center servers is about 27% of the total energy consumed by the data center that becomes two-third of the energy consumed by IT equipment [17]. A cloud data center server operates only at about 20-30%, while 70-80% remains idle doing no useful job [18].

Energy consumption at the cloud data center can be optimized by estimating future cloud resource demand and then scale the resources accordingly [19]. Data center's resource usage prediction is a challenging task due to diversity of workloads as well as irregular arrival of client requests (i.e. arriving at a different time and requesting for varying amounts of resources) [19].

The CPU utilization is one of the critical metrics for evaluating the cloud data center server's performance and is used by researchers for estimating server's future performance [20]–[22]. In cloud data center's servers CPU is the highly demanding resource and so the primary cause of resource unavailability [4]. Several techniques have been used for CPU usage prediction in the cloud. These are multi-layer perceptron (MLP), auto-regressive moving average (ARIMA), extreme learning machine (ELM), auto-regression (AR), k-nearest neighbor (KNN), feed-forward artificial neural network (FNN), autoregressive neural network (AR-NN), and recurrent artificial neural network (RNN) [4], [23]–[28], [36]. The CPU usage can be seen as a function of time so it can be characterized as time series problem. Thus, it becomes a regression problem that can be solved by either conventional time series analysis methods or neural networks. In traditional time series analysis, a mathematical formula defines the dynamics in the series and instructions and rules are the core of the study. In contrast, when a neural network is applied to data, it trains, gains experience, learns the regularities of the past and sets its own rules [29]. ANN has been proved better performance than prediction methods like average, moving average, last value, autoregressive moving average and exponential smoothing when tested for resource usage prediction [25], [30], [31]. As ANN has shown better results than prediction methods like average, moving average, last value, autoregressive moving average and exponential smoothing when tested for resource usage prediction [25], [30], [31]. But conventional ANNs trains only their weights associated with their neuronal inputs and don't evolve their number of neurons and number of synaptic connections to a neuron [4], [32].

Modern multi-core CPUs can scale (up/down) frequency by using dynamic voltage and frequency scaling (DVFS) [33]. DVFS mechanisms are applied at the micro-architecture level in most of today's servers [34]. The energy-saving efficiency of DVFS is decreasing because of the following reasons in modern virtualized data centers. First, the supply voltages are currently low for processor chips and hence less space for further reduction. Second, advanced processor chips are multi-cores and must operate at the same supply voltage level and thus the same frequency (e.g. Intel processors). Finally, in a virtualized system (server), information collection (necessary for optimal clock frequency and voltage level for the CPUs) about running applications is challenging because DVFS resides in a privileged domain (hypervisor) while the application in user domain (virtual machine) [35]. Thus, we use a rule-based scaling approach that scale individual core(s) instead of DVFS that varies voltage levels to operate CPU at different frequencies. Therefore, we use a rule-based scaling mechanism for enabling/disabling CPU logical core(s) according to predicted CPU usage.

Authors in [19] report that energy consumption at the cloud data center can be optimized by estimating future cloud resource demand and then scale the resources accordingly. Our methodology consists of resource estimation and then scaling of resources. As CPU utilization is one of crucial metrics for evaluating the cloud data center server's performance and is used by researchers for estimating server's future performance [20]–[22], thus we predict CPU usage of a cloud server and then scale CPU cores (logical cores) accordingly. ANN is a right candidate for resource estimation as discussed in [25], [30], [31], but conventional ANNs have limited learning scope that is they learn only their weights [4]. To enhance the learning capability of ANNs, we use the recurrent Cartesian genetic programming-based artificial neural network (RCGPANN). RCGPANN can learn synaptic weights, the number of synaptic connections per neuron and number of neurons during evolution.

The findings of this research will lead to the benefit of society because cloud computing plays a vital role in science and technology. The massive share of global electric power consumed by cloud data centers justifies the need for more effective techniques for energy efficiency in cloud servers. Thus, cloud infrastructure owners that apply the recommended approach derived from the results of this study will be able to minimize their data center's energy consumption and heat dissipation. That will reduce the volume of carbon gases emitted by data centers in the environment. Cloud computing can be used by scientists to fulfil their large scale computational needs. Minimizing the power consumed by the cloud infrastructure lowers the cloud services cost. Thus, scientists can have economic cloud services from an energy-efficient cloud. Carbon gases have a terrible impact on human life and the environment by changing water supplies, weather patterns, the growing season for food crops and vulnerable coastal communities with increasing sea levels. As mentioned earlier that with an increase in energy consumption by data centers more heat is dissipated, and more carbon is emitted in the environment [12]. Therefore the recommended approach, when adopted by cloud data centers, will have a significant reduction in energy consumption and carbon gases emission. Besides cloud data centers, our designed system can be used in cell phones, laptops, and computers used in unmanned aerial vehicles (UAVs) where energy saving is more necessary and critical. Pakistan is experiencing the worst energy crises that have severely affected its economy. Our developed system can reduce the power consumed by the country's workstations, servers, data centers, UAVs, laptops and cell phones. Thus, it will have a significant impact on minimizing the energy crisis and boosting the economy of the country.

Our technical contributions are in resource monitoring, resource utilization prediction, energy-efficient resource management and autonomous resource provisioning in cloud systems. We highlight our technical contributions below:

- We propose a system information gatherer and reporter (SIGAR) application programming interface (API) based resource monitor that has the capability to monitor resources both physical and virtual at any frequency set by the cloud provider.
- We propose a recurrent, CGP artificial neural network (RCGPANN) based prediction system.
- We propose a rule-based scaling system for elastic scaling of cloud resources.

II. RELATED WORK

Resource usage prediction of IaaS cloud is one of the primary technique used for predictive scaling that ensures energy efficiency in the cloud. Prevost, John J., et al. in [23] used multilayer perceptron and auto-regression for predicting future resource requests. They used back-propagation for training their MLP model that consisted of three inputs, one output and two hidden neurons [23]. Their work would be more productive if they used actual resource usage instead of the number of requests for a resource. Ismaeel, S. and Miri, A. used ELM for predicting virtual machine (VM) usage demand in [24]. ELM is a neural network where weights are trained randomly in a single step. This property makes it faster than other neural networks and SVMs, but single step assignments of weights make it pronto significant prediction errors. Calheiros, Rodrigo N., et al. in [25] used ARIMA model for predicting the number of requests for a cloud resource in future. Farahnakian, Fahimeh, et al. in [26] used the K-NN algorithm for predicting the status of host CPU, whether it is under-loaded or over-loaded. Thus, to achieve energy efficiency and maintain service level agreement (SLA) through VM consolidation. Ali Yadavar Nikravesh, Samuel A Ajila, and Chung-Horng Lung in [27] proposed a self-adaptive prediction suite that selects between ANN and SVM algorithms by watching the workload patterns. They categorized workload into periodic, growing and unpredicted patterns. Their self-adaptive system selects SVM in case of periodic and growing patterned workload while ANN is chosen in case of unpredicted workloads. Sladescu, Matthew, et al. in [28] used the event aware prediction method for predicting auction-based workload burst. Their approach outperformed ARIMA and SVR for workload bursts prediction because bursts are innately uneven [28]. In our previous work, we used an adaptive prediction system that selects between AR-NN and ARIMA for predicting future CPU usage of an IaaS cloud [36]. The adaptive system first checks the normality of the workload; if it passes the normality test, then the prediction is made by ARIMA. Otherwise, AR-NN is used [36]. The AR-NN uses back-propagation for training its weights so it can be improved by training through evolutionary methods, i.e. through crossover and/or mutation [4]. Mason, Karl, et al. used RNN with two input neurons, three hidden neurons, one output neuron and 18 weights for predicting CPU usage demand. They trained their network with particle swarm optimization (PSO), covariance matrix adaptation evolutionary strategy (CMA-ES) and differential evolution (DE), respectively [4]. Their results showed that CMA-ES has better prediction accuracy than random walk (RW), moving average (MA), linear regression (LR), back-propagation (BP), PSO and DE. The RNN used in [4] had the number of neurons, and the number of connections is fixed during evolution. To make RNN more evolvable, we propose RCGPANN based CPU usage prediction system. That has an evolvable number of neurons, evolvable synaptic connections and evolvable synaptic weights.

Elasticity is one of the core features of cloud computing. That ensures the ability to add/remove structures with minimal cloud provider/user intervention and with minimum time, as per demand [37]. Researchers have classified elastic scaling solutions based on the scope, policy, purpose, and method as given in [38]. In this work, our scaling scope is IaaS cloud and policy is automatic. While the goal is energy efficiency and way is vertical scaling. Cloud providers have partially infused elasticity in their clouds VMs, applications and servers. Cloud providers like Amazon EC2, Rackspace and GoGrid have horizontal auto-scaling mechanisms [39]–[41]. Dawoud et al. in [42] proposed an elastic VM for ensuring service level objectives (SLOs) automatically. Toffetti, Giovanni, et al. in [43] proposed elastic applications for running on a VM. Arabnejad et al. used a rule-based system for horizontal scaling of cloud resources. They used five scaling states for a VM. They used fuzzy, reinforcement learning strategies for the prediction of VM usage [44]. Ilyushkin, Alexey, et al. in [45] used several auto-scaling policies for scaling resources used by workflow-based workloads in cloud environments. Zhang, Xinwen, et al. in [46] introduced cloud and mobile device web lets for providing transparency in the use of cloud resources by mobile devices. Shahin, Ashraf A. proposed threshold-based auto-scaling policies for enabling/disabling VMs of a cloud. The scaling decisions were made on RNN predicted CPU usage [47]. The most dynamic and vital resource that got more importance by researchers is CPU [4].

Modern CPUs consist of several cores while individual cores can be turned on/off when demanded by host load. CPU also can follow the DVFS mechanism [33]. But the DVFS mechanism has limitations in increasing/decreasing frequency at core-level by increasing/reducing supply voltages [35]. Thus we propose a rule-based auto-scaling mechanism for scaling CPU through enabling/disabling logical core(s).

III. RESOURCE MONITOR

Our resource monitor uses SIGAR API for collecting CPU usage of the host computer. Whereas SIGAR is an API that offers an interface for collecting system information conveniently. It has the capability to collect CPU, memory, network, and disk usage information. Our resource monitor computes CPU percentage usage by using the expression: $CPU_{percentage} = CPU_{total\,percentage} - CPU_{idle\,percentage}$. Where $CPU_{total\,percentage}$ has a fixed value of 100, and $CPU_{idle\,percentage}$ varies with the variations of workload on CPU. Our resource monitor stores CPU usage in a buffer and the prediction system reads CPU usage from that buffer for prediction purpose.

In the case of memory, it can collect percentages and actual values of free memory and used memory. While in case of hard disk, it can obtain the number of disk reads and writes along with the amount of disk read and write bytes at any timestamp in milliseconds. On the other hand, our resource monitor can collect the amount of received and transmitted bytes and the number of received and transmitted packets. It can obtain information about the number of packets dropped in transmission and reception. It can report the number of errors during transmission and reception.

Our resource monitor can be used for monitoring resources at any timestamp in milliseconds. Besides monitoring, the collected resource usage can be used for the estimation of each resource. The resource monitor data can be used to estimate the host computer usage by applying a multi-objective function. In the multi-objective function, each resource will have a weight assigned based on its usage history.

We have validated our resource monitor for collecting CPU usage data of a multicore server in sections VII.C and IX.

IV. CGP BASED RECURRENT NEURAL NETWORK

Cartesian genetic programming (CGP), is a graph-based form of genetic programming developed by Julian Miller in 1999 and 2000 with some contribution from join work of Julian Miller and Peter Thomson in 1997 [48]. In CGP, the genetic code of the program represents a network of nodes placed in the Cartesian plane. RCGPANN is a neuroevolutionary technique based on CGP for representation and evolution of recurrent neural networks [49]. The parameters of CGP are number of nodes in the chromosome, the maximum number of inputs per node (i.e. arity), number of columns, number of rows and levels-back. Different arrangements of these parameters create different graphs (networks). That is a fixed-length array of integers represents the CGP genotype. The position of integers represents the type of genes. These genes include input connections, weights, switch controls, node function (activation function) and output genes. In case of RCGPANN besides the above genes, some outputs are fed-back as inputs which are called recurrent inputs. When genotypes of CGP chromosomes are decoded to phenotypes, all nodes are not connected in the path from inputs to outputs. The un-connected nodes are called non-coding nodes. Through mutation operator, the status of a node or nodes is/are randomly chosen between coding and non-coding. This status change of nodes can result in an offspring with different characteristics than the parent. The CGP nodes are replaced by artificial neurons to form RCGPANN. During evolution RCGPANN topology, weights and activation functions experience changes through mutation [50]–[52].

A. RCGPANN NEURON

CGPANN neuron is similar to that of conventional ANN, but in case of RCGPANN, it has recurrent inputs also along with forwarding inputs. Forward inputs come from system inputs, or outputs of other neurons belong to previous columns in the Cartesian plane. Recurrent inputs are the system outputs fed-back as input to the system. RCGPANN neuron is shown in (1).

$$\alpha = \left(\sum_{i=1}^{n} \omega_i \times x_i \times \tau_i\right) + \left(\sum_{i=n+1}^{p} \omega_i \times r_i \times \tau_i\right) \quad (1)$$

where $\mathrm{x_1, x_2, x_3, \cdots, x_n}$ represent feed-forward inputs, $\omega_1, \omega_2, \omega_3, \cdots, \omega_n$ represent respective weights of

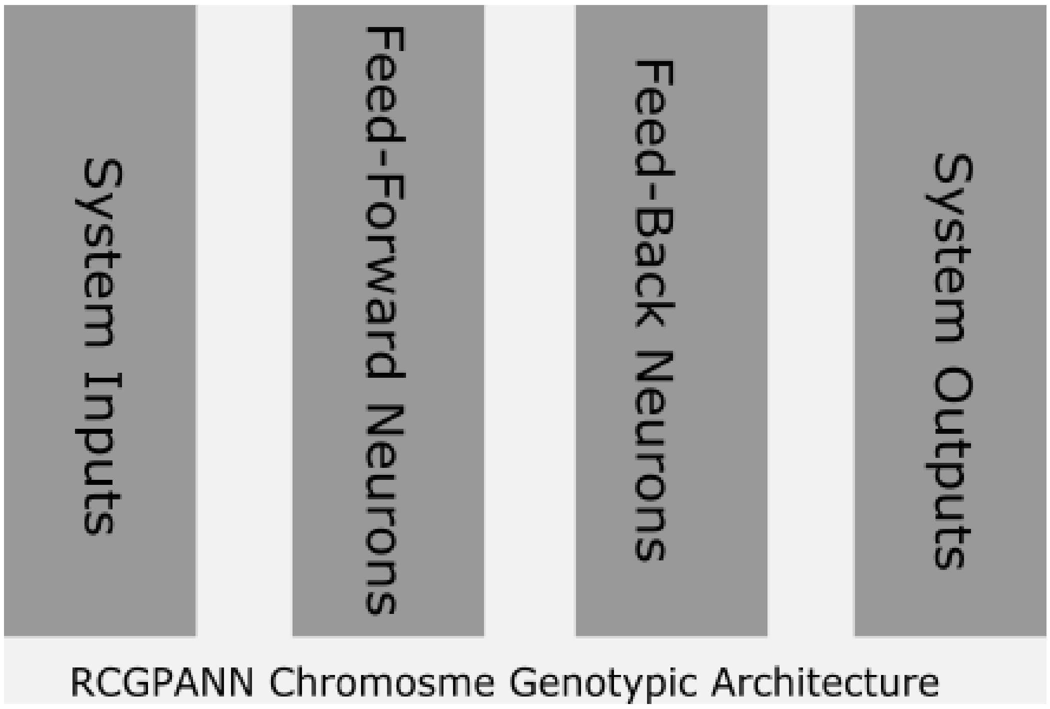

FIGURE 1. RCGPANN chromosome architecture. inputs, outputs, feed-forward network genes and feed-back network genes of neurons placement in the cartesian plane.

feed-forward inputs, $\tau_1, \tau_2, \tau_3, \cdots, \tau_n$ represent forward pass transistor switch enable/gate/controller, $r_{n+1}, r_{n+2}, r_{n+3}, \cdots, r_p$ represent recurrent inputs, $\omega_{n+1}, \omega_{n+2}, \omega_{n+3}, \cdots, \omega_p$ represent recurrent inputs respective weights and $\tau_{n+1}, \tau_{n+2}, \tau_{n+3}, \cdots, \tau_p$ represent recurrent pass transistor switch enable/gate/controller. The switch controller/enable is used to enable or disable an input connection to neuron during evolution.

Our RCGPANN neuron use sigmoid function as an activation function as shown in (2).

$$\sigma(\alpha) = \frac{1}{1+e^{-\alpha}} \tag{2}$$

B. RCGPANN CHROMOSOME

Genes of a neuron are placed in a specific order. The first input is placed in the first position, followed by its weight and a switch (enable), the second input followed by respective weight and enable switch, and so on. At the last position, function gene is placed. Both the feed-forward and feed-back neurons of the RCGPANN follow the same mechanism. Figure.1 presents chromosome architecture of RCGPANN genotype. In RCGPANN chromosome, first system inputs (neurons) are placed followed by feed-forward neurons. Then feed-back neurons are placed, and at the end, system outputs are placed.

C. TRAINING OF RCGPANN

Neurons of RCGPANN are placed in the order shown in figure 1. Figure 2 presents the neuroevolutionary training process for RCGPANN. All genes are mutated with random values in the initialization stage of evolution to make parent chromosome. Parent chromosome inherits further 9 (λ is 9) chromosomes in the first stage of evolution. Error calculation metric used here is mean absolute error (MAE) that is given by (3). The chromosome with the lowest MAE of the nine chromosomes is selected for the next stage of evolution. This process continues further until either the lowest possible MAE is reached or the highest number of generations/evolutions is reached.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{3}$$

where y_i represents the i^{th} output generated by RCGPANN, while x_i represents the i^{th} sample in the data set. Whereas, n represents the number of samples in the data set.

D. TESTING OF RCGPANN

In the testing process, the trained RCGPANN is subjected to testing data, and MAEs are calculated. The testing procedure is that let the network has a total of eleven inputs (five forward inputs and six feed-back inputs) and six system outputs. The testing data set's first five values are fed to the network that generates six outputs; these six outputs, along with five data set's primary inputs are fed to the network to get six outputs. These six outputs are compared with values in data set at respective positions. The first output is compared with data set value at sixth position, the second output is compared with the value at seventh position and so on to get absolute errors between data set values and RCGPANN outputs.

Algorithm 1 Predictive Scaling Algorithm

```
1 :   Begin
2 :   $C_p \leftarrow$ predictied resource usage
3 :   $C_{max} \leftarrow$ maximum resource capacity
4 :   $C \leftarrow$ current resource capacity
5 :   $c_1 \leftarrow$ capacity of first core
6 :   $\wedge$ if $C_p > C\ (|C_p - C| \geq c_1)\ C_p < C_{max}$ then
7 :   Call Up Scaling Algorithm 2
8 :   $\wedge$ elseif $C_p < C\ |C_p - C| \geq c_1$ then
9 :   Call Down Scaling Algorithm 3
10 :    else
11 :    Do Nothing
12 :    endif
13 :    End
```

After finding all absolute errors throughout the length of the data set, then the whole MAE is calculated by accumulating all MAEs between each point of test data and predicted data.

V. THE SCALING SYSTEM

The scaling system scales the resources as per prediction made by RCGPANN. In present work, our resource understudy is CPU. Our resource monitor extracts CPU percentage values, preprocess these values, apply to the RCGPANN and then feed the averaged output value to the scaling system. The scaling procedure is performed in the following manner.

An IaaS server CPU consists of n cores with capacities $c_1 = c_2 = \cdots = c_n$. The percentage contribution of each core in total CPU is $c_1 = c_2 = \cdots = c_n = \frac{100}{n}$. With m active cores the current CPU usage C will be $C = \frac{100}{n} \times m$. In a special case, let there are 4 cores in CPU and 3 of them are active then current CPU percentage usage is $\frac{100}{4} \times 3$.

The predicted CPU percentage usage is C_p and maximum CPU percentage capacity is C_{max}. The scaling decisions are made according to algorithm 1, algorithm 2 and algorithm 3.

According to algorithm 1, when C_p is greater than C, $|C_p - C|$ is greater than or equal to c_1 and $C_p < C_{max}$ then up scaling is called. If C_p is less than C by equal or greater capacity value than c_1 then down scaling is provoked. In case the difference between C_p and C is less than c_1 then No-Scaling is done.

Algorithm 2 Up Scaling Algorithm

```
1 :   Begin
2 :   $i \leftarrow m$
3 :   while $(|C_p - C| \geq c_1)\ C_p \leq C_{max}$ do
4 :   ON core $c_{i+1}$
5 :   $C = C + c_{i+1}$
6 :   $i = i + 1$
7 :   endwhile
8 :   Go to Algorithm 1
12 :   End
```

Algorithm 3 Down Scaling Algorithm

```
1 :   Begin
2 :   $i \leftarrow m$
3 :   while $|C_p - C| \geq c_1$ do
4 :   OFF core $c_i$
5 :   $C = C - c_i$
6 :   $i = i - 1$
7 :   endwhile
8 :   Go to Algorithm 1
12 :   End
```

Algorithm 2, lists the steps required for up scaling. The up-scaling mechanism iteratively scales resource (CPU in current case) when predicted resource capacity is greater than current capacity by capacity of a single core (in homogeneous CPU) and of the smallest core (in case of heterogeneous CPU). In each iteration of the **while** loop a new core is enabled and added to the current capacity of the CPU. Similarly, the value of i is incremented in each iteration of **while** loop for next iteration. The loop goes to next iteration when the conditions of the **while** loop are satisfied otherwise the loop terminates.

In algorithm 2, i represents number of cores, it is initialized with current cores m. When the condition $|C_p - C| \geq c_1$ of while loop is satisfied, the loop statements are executed, until the condition gets false. In each iteration of the while loop a new core is added to the active ON cores.

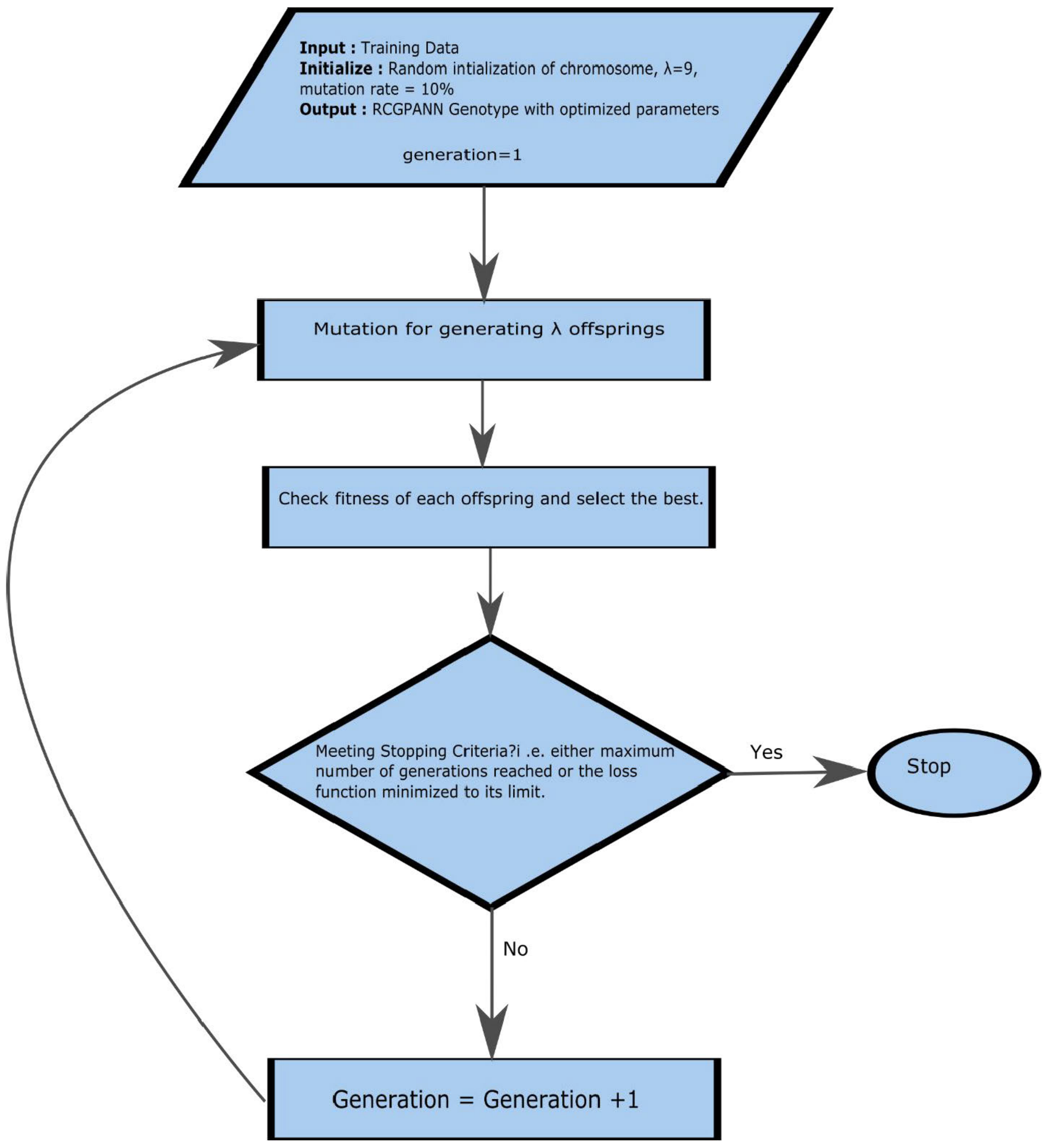
Input : Training Data
Initialize : Random intialization of chromosome, λ=9, mutation rate = 10%
Output : RCGPANN Genotype with optimized parameters
generation=1
Mutation for generating λ offsprings
Check fitness of each offspring and select the best.
Meeting Stopping Criteria?i .e. either maximum number of generations reached or the loss function minimized to its limit.
Yes
Stop
No
Generation = Generation +1

FIGURE 2. Evolutionary training of RCGPANN chromosome.

When predicted usage is less than current CPU usage then down-scaling algorithm is called from algorithm 1. Down-scaling mechanism is presented in algorithm 3.

In contrast to up-scaling, in down-scaling on each iteration of the while loop a core is made OFF, and current active ON cores are updated.

Scaling with granularity of cores limits the scaling system from wasting of time in up and down scaling of CPU. Such scaling granularity reduces the time complexity of the scaling algorithm. In case of 1000 cores the worst case time complexity of up scaling will be $O(999)$. Similarly, the worst case time complexity for down scaling will also be $O(999)$. When both up scaling and down scaling are in worst case then the predictive scaling algorithm will have worst case complexity of $O(1998)$. Let k represent the number of cores in CPU then the worst case time complexity of the predictive scaling system can be $O(2k - 2)$.

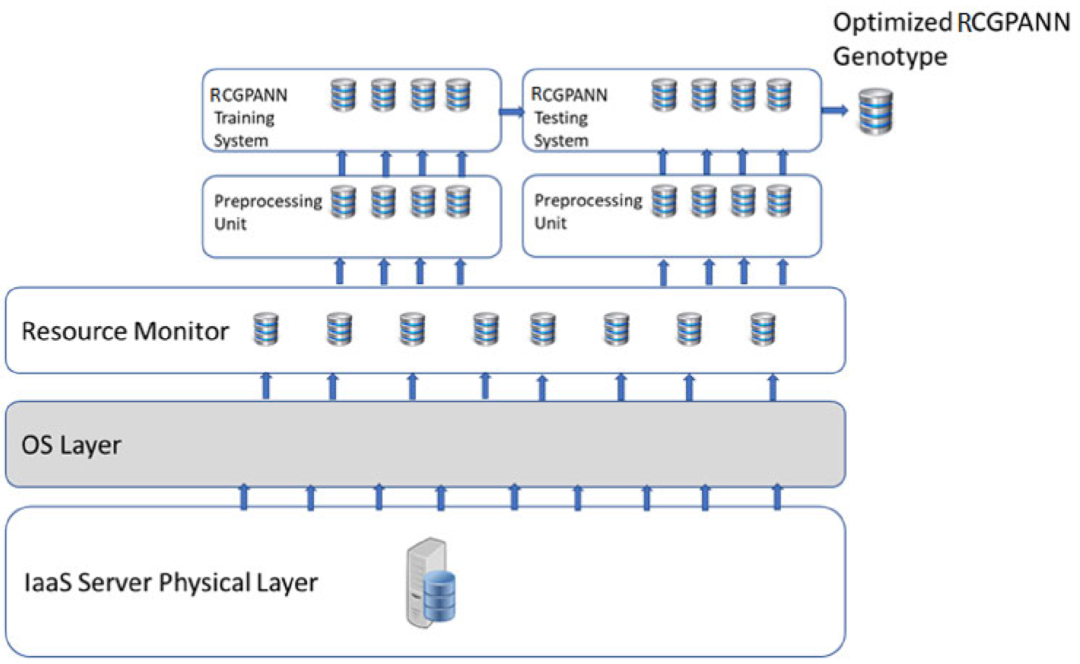

FIGURE 3. Training and testing system of RCGPANN.

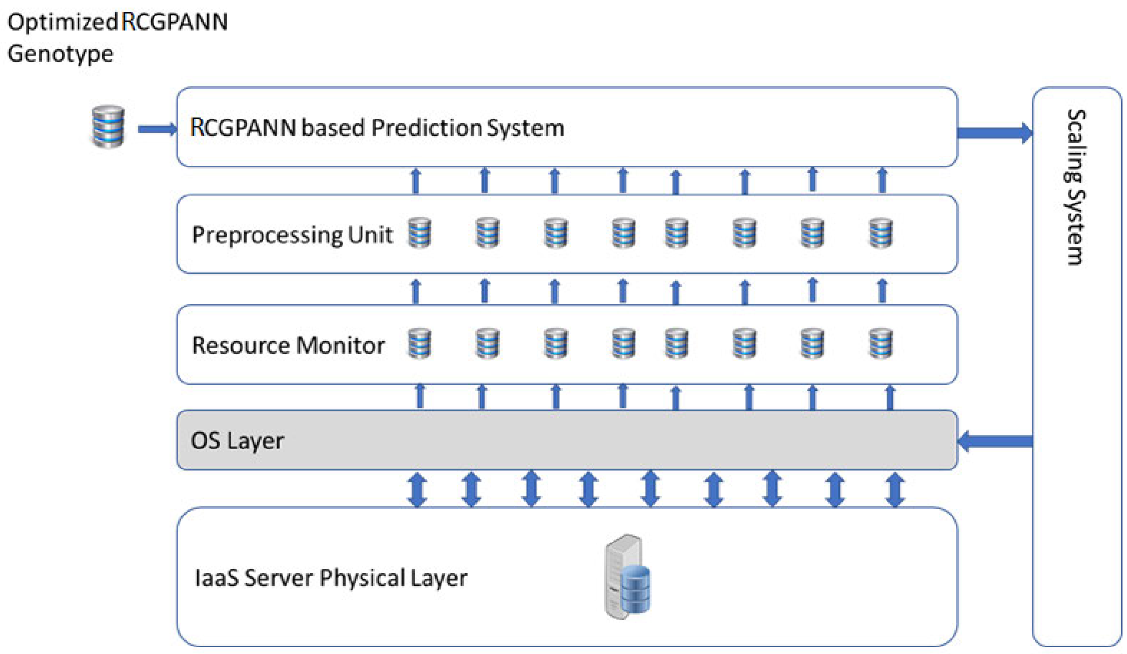
Optimized RCGPANN Genotype
RCGPANN based Prediction System
Preprocessing Unit
Resource Monitor
OS Layer
IaaS Server Physical Layer
Scaling System

FIGURE 4. Architecture of real-time implementation and validation system of RCGPANN based predictive scaling system.

VI. SYSTEM ARCHITECTURE AND EXPERIMENTAL PLATFORM

The system architecture of the experimental platform is divided into two parts. First is for training and testing of RCGPANN and second is for real-time implementation and validation of RCGPANN based prediction and scaling system. The training and testing system consists of a physical layer, operating system layer, resource monitor layer, preprocessing, and training/testing layers, as shown in figure 3. The resource monitor extracts resource usage values from the physical layer with the help of the operating system layer. The resource usage values are stored in buffers and preprocessed with 0-1 normalization, as shown in (4).

$$N(i) = x(i) - min/max - min \tag{4}$$

where $x\ (i)$ and N (i) represent i^{th} samples actual and normalized values of resource usage respectively. The minimum and maximum values of the buffered resource usage are *min*, and *max* respectively. The normalized values are used for training and testing of RCGPANN.

The real-time implementation system architecture is composed of a physical layer, operating system layer, resource monitor layer, preprocessing, optimized RCGPANN layer and the scaling system, as shown in figure 4. Resource monitor collects resource usage that is normalized in preprocessing block and passed through optimal RCGPANN for prediction. The predicted usage is de-normalized by the formula given in (5) and averaged to get a single value of prediction. The scaling decision is taken based on this prediction value.

$$x(i) = N(i)(max - min) + min \tag{5}$$

The scaling system through operating system commands instructs the physical layer to scale its resource (i.e. CPU) according to predicted values.

VII. RESULTS AND DISCUSSION

The training and testing of RCGPNN were conducted on a system with Intel® Core^{TM} i7-6700 CPU 3.40GHz processor and 8GB RAM. The data set used was 29 days CPU usage of a Bitbrains[5] data center server. Training and testing experiments were conducted on fifty-fifty per cent of data set samples. For the real-time implementation, we used a system with CPU @ 1.7 GHz, 4GB RAM and Ubuntu OS.

We trained RCGPANN using neuroevolutionary algorithm shown in figure 2. Input for the evolutionary method is the training data set, with random initialization of chromosome genes. The mutation rate is set to 10% and followed the 1+9 evolutionary strategy. We conducted training experiments for predicting one instance (next one sample), two instances, three instances, four instances, five instances and six instances. The resultant model of each experiment is tested with test data.

A. ACCURACY AND COMPLEXITY TRADE-OFF WITH NUMBER OF PREDICTION INSTANCES

Figure 5 presents the space and time complexities of RCGPANN models achieved in experiments for predicting 1-6 instances (samples). We have computed the space complexity of a model by counting the number of neurons in the model. While for the time complexity, we have calculated the number of logistic sigmoid functions and the number of multiplier units of the critical path (the longest track of the parallel network architecture) of the model. The space complexities of the RCGPANN models trained for one instance (sample) prediction, two instances prediction, three instances prediction, four instances prediction, five instances prediction, and six instances prediction are O(5), O(11), O(18), O(15), O(14), and O(12), respectively. The time complexities of the model achieved for one instance (sample) prediction, two instances prediction, three instances prediction, four instances prediction, five instances prediction, and six instances prediction are O(5), O(10), O(18), O(20), O(18), and O(14), respectively. Results show that space and time complexities have a linearly increasing trend for 1-3 instances prediction while from 3-6 instances, there is a gradually decreasing trend in complexities. The most critical parameter of the prediction model is its prediction accuracy. Figure 6 presents prediction errors (MAE) of 1-6 instances predictions. Prediction MAE increases with an increase in the number of prediction instances approximately in a linear fashion. Thus a prediction model with few prediction instances may have better accuracy than a model with more prediction instances.

B. ACCURACY AND COMPLEXITY TRADE-OFF WITH NUMBER OF NEURONS

Further, we discuss prediction results for one instance prediction in detail by conducting fifteen different experiments. In first, fourth, seventh, tenth and thirteenth experiments, we initialize chromosome with fifty neurons. In second, fifth, eighth, eleventh and fourteenth experiments, we initialize chromosome with hundred neurons. In third, sixth, ninth, twelfth and fifteenth experiments, the initial chromosome size is five hundred neurons. Figure 7 presents space and time complexities of RCGPANN models obtained in experiments for one instance prediction. We have computed the space complexity of a model by counting the number of neurons in the model. While for the time complexity, we have calculated the number of logistic sigmoid functions and the number of multiplier units of the critical path (the longest track of the parallel network architecture) of the model. The model achieved in the first experiment has space complexity of O(9) and the time complexity of O(8). The RCGPANN model of the second experiment has space complexity of O(10) and the time complexity of O(12). Space and time complexities of the model in the third experiment are O(9) and O(10), respectively. In the fourth experiment, the model has space complexity of O(8) and time complexity of O(12). In fifth experiment, the model has space complexity of O(5) and time complexity of O(8). The models of sixth, seventh, eighth, ninth, and tenth experiments have time complexities of O(16), O(14), O(10), O(48), and O(8), respectively. The models of sixth, seventh, eighth, ninth, and tenth experiments have their respective space complexities of O(11), O(10), O(10), O(40), and O(8). Similarly, the RCGPANN models achieved in eleventh, twelfth, thirteenth, fourteenth, and fifteenth experiments have space complexities of O(9), O(22), O(7), O(9), and O(5), respectively. The models of eleventh, twelfth, thirteenth, fourteenth, and fifteenth experiments have their respective time complexities of O(12), O(28), O(8), O(10), and O(6). Results show that space and time complexities have nonlinear behavior from first experiment to fifteenth experiment. Figure 8 presents the MAE of each model obtained in experiments. From experiment two to five, MAEs have increasing trend while from five to six, MAE

[5]http://gwa.ewi.tudelf.nl/datasets/gwa-t-12-bitbrains

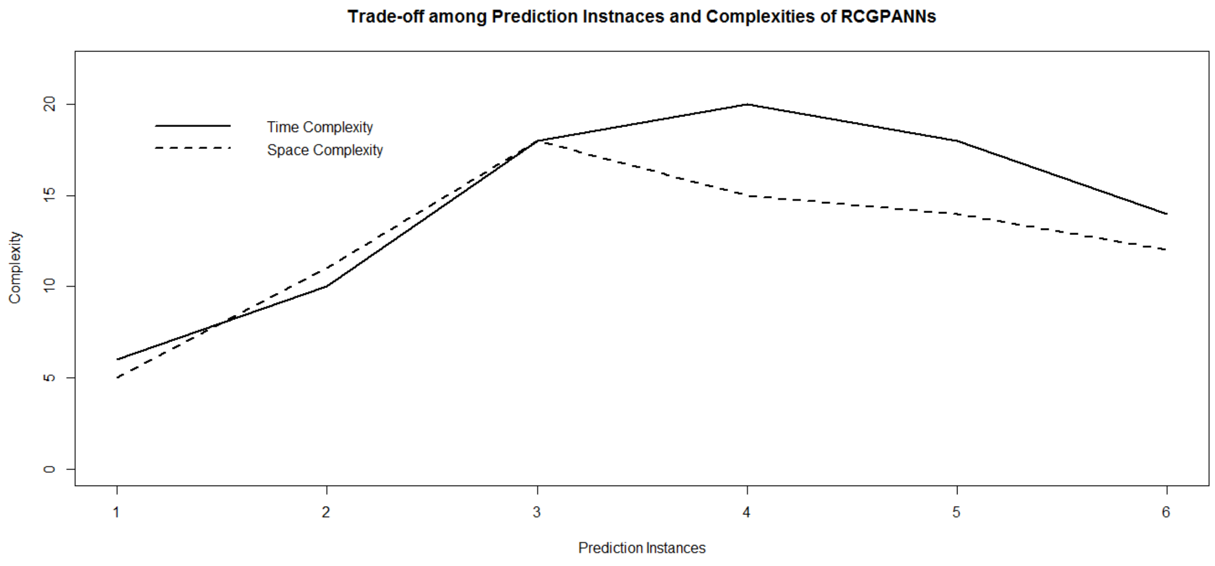

FIGURE 5. Space and time complexities of RCGPANNs with different instances prediction.

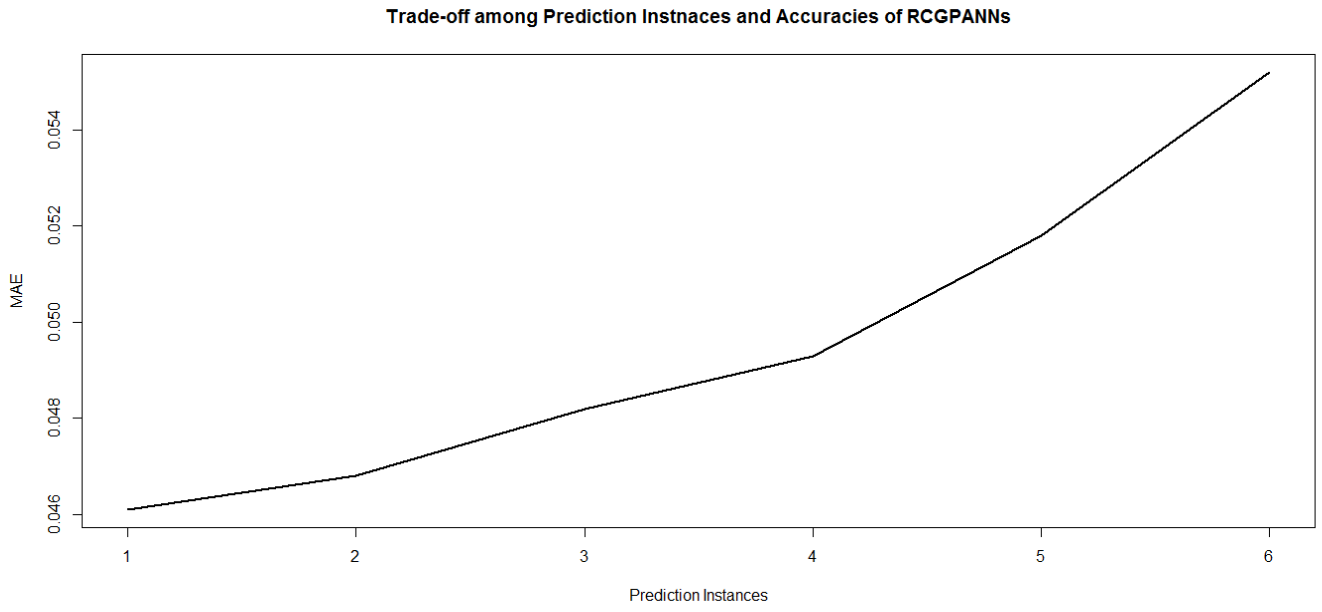

FIGURE 6. Accuracy of RCGPANNs with different instances prediction.

decreases. Throughout the experiments, MAE has a zigzag form; thus, a generalized rule cannot be formed. Although, a model with least MAE is selected for further investigation of predictive scaling system.

Further, we discuss the real-time implementation of the RCGPANN model that has the best prediction accuracy along with the scaling system.

C. REAL-TIME IMPLEMENTATION AND VALIDATION ON MULTI-CORE SERVER

We implemented our real-time system on a computer with CPU @ 1.7 GHz, 4GB RAM and Ubuntu OS. Our system includes a Java-based API for resource monitoring, java-based preprocessor, java-based optimized RCGPANN, java-based de-normalizer and a mean finder of six RCGPANN outputs for getting single output. The scaling system includes nested if-else statements and while loops. Within the while loops, Ubuntu shell commands[6] are called for enabling and disabling of CPU logical core(s) [54]. The instantaneous power consumption profile of CPU with one core, two cores, three cores and four cores is shown in figure 9. The power

[6]https://github.com/spotify/linux/blob/master/Documentation/cpu-hotplug.txt

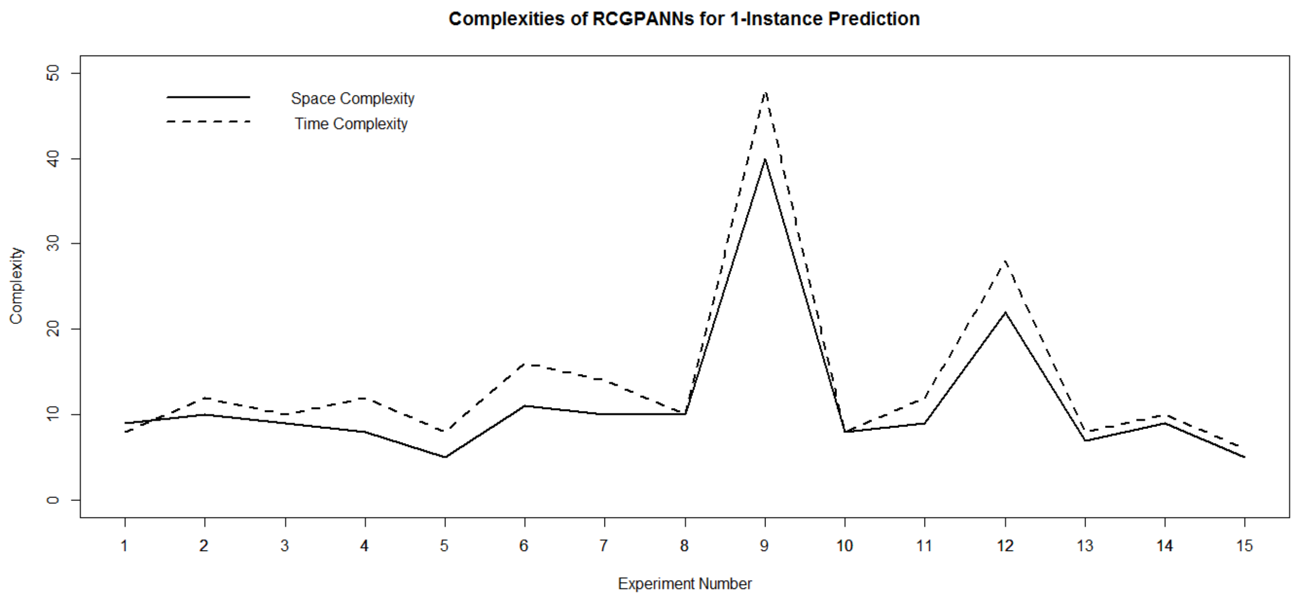

FIGURE 7. Space and time complexities of RCGPANN models.

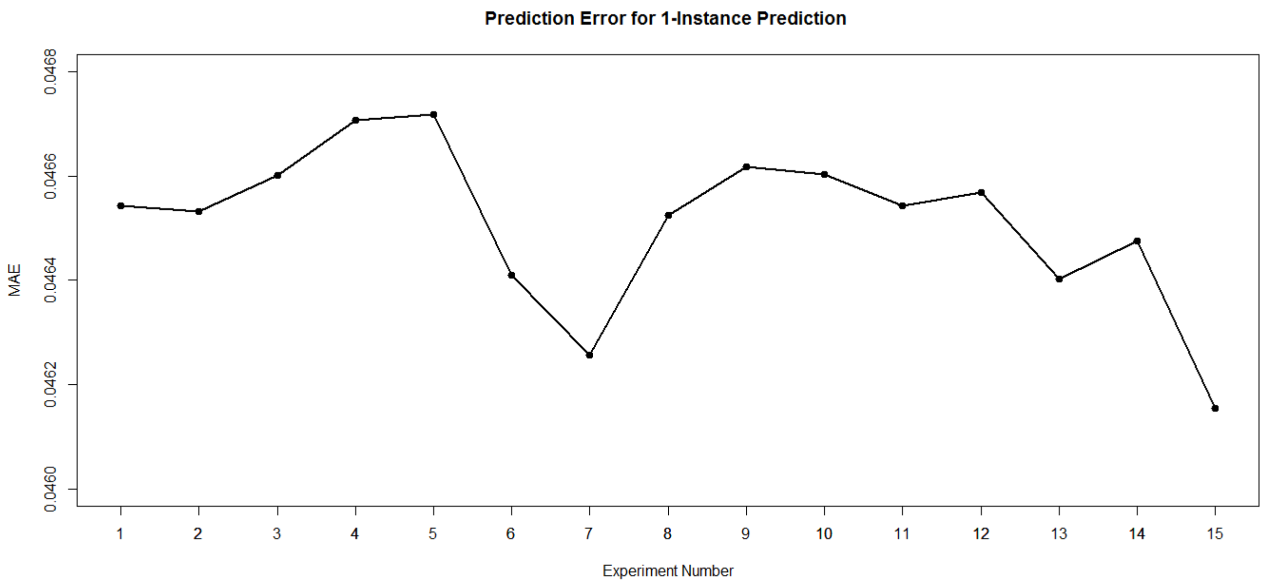

FIGURE 8. Prediction errors of RCGPANN models.

consumption for four cores reaches 16 watts at maximum and 14 watts at minimum, for three cores the power consumption floats between 12 watts and 15 watts, for two cores it is between 13 watts and 14 watts, and for one core it is between 12 watts and 13 watts. Thus CPU with more cores in ON state has more power consumption than with few cores, as shown in figure 10. The average power consumed by 4-cores is 15.22 watts, by 3-cores is 14.16 watts, by 2-cores is 13.28 watts and by 1-core is 12.74 watts. Thus by increasing and decreasing the CPU cores, energy consumption increases and decreases respectively.

Figure 11 shows the logical cores usage. In figure 11, the core-1 (CPU1) is 48.5% used, core-2 (CPU2) is 57.4%, core-3 (CPU3) is 40.8% and core-4 (CPU4) is 59.8% in use. Thus it will reduce energy consumption if suitable scaling is made with few numbers of cores. At this stage, our prediction system predicted future CPU percentage usage as shown in figure 12, as 71.38% so this percentage usage requires only 3-cores, so our scaling system scales the CPU cores to 3 and thus saves extra power consumption (that was in case of 4-cores). The core-wise usage of CPU cores shown in figure 13, validates our prediction and scaling system.

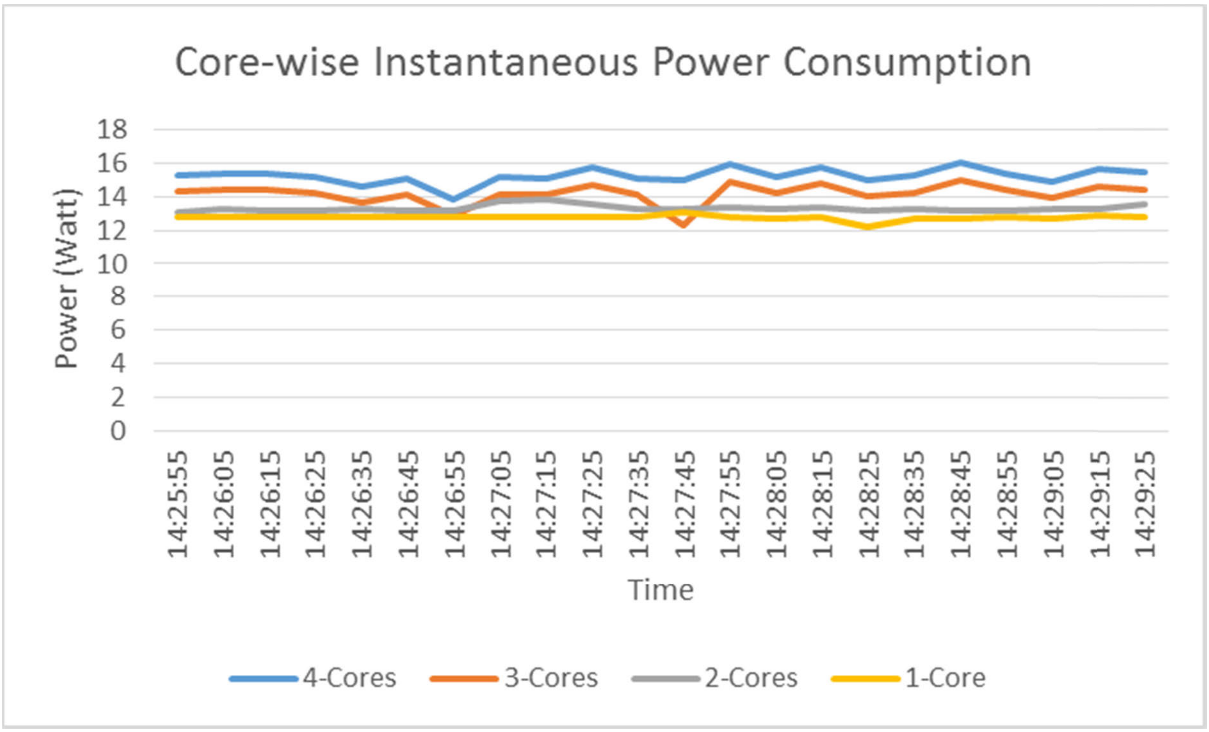

FIGURE 9. Core-wise instantaneous power consumption of Intel R CoreTM i3-4010U CPU @ 1.7 GHz, 4GB RAM and Ubuntu OS.

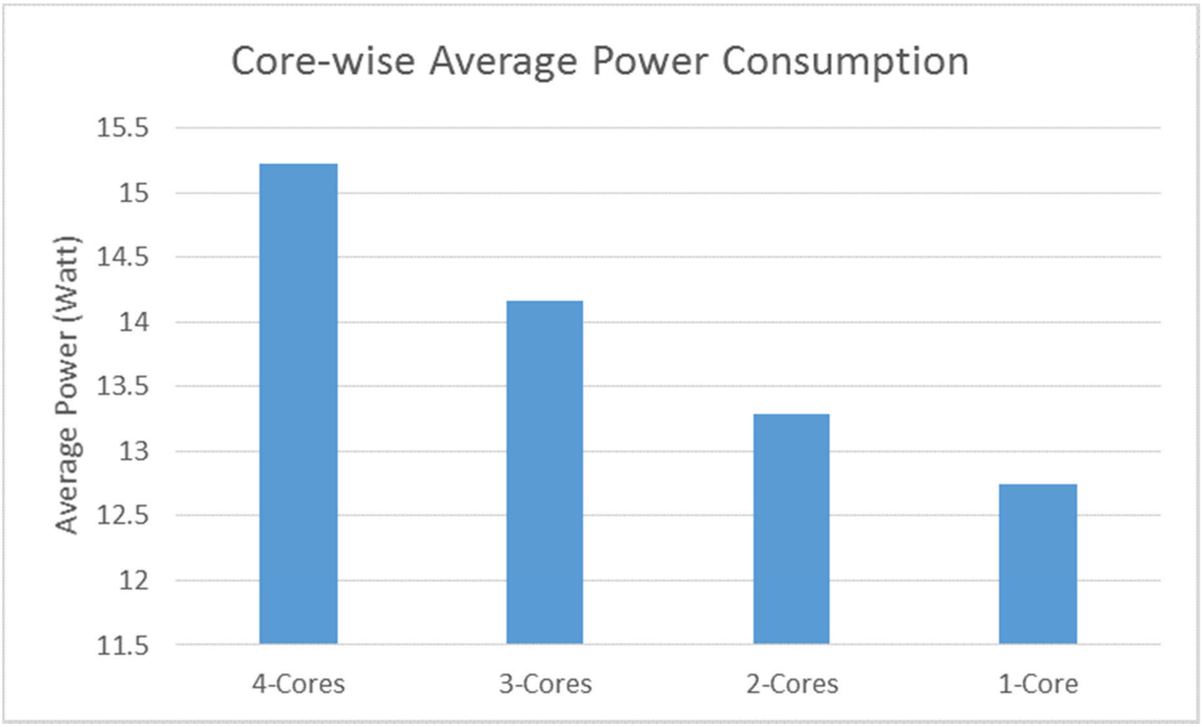

FIGURE 10. Average power consumption of Intel®Core^{TM}i3-4010U CPU @ 1.7 GHz, 4GB RAM and Ubuntu OS.

Now the core-1 is 72.4% in use, core-2 is 74.7%, core-3 is 89.2% used and core-4 is 0.0% (i.e. OFF). Thus now CPU is using 3-cores for the current load that ensures minimization of power consumption, heat dissipation and indirectly carbon dioxide emission (in case of carbon fuel usage).

TABLE 1. Comparison of prediction models on test data.

Method	MAE	Time Complexity	Evolvable parameters
MLP [45]	0.0761	$O(4)$	Weights
ELM [46]	0.14041	$O(4)$	No
ARIMA [47]	0.13931	$O(6)$	No
AR [45]	0.14031	$O(1)$	No
AR-NN [48]	0.06112	$O(4)$	Weights
KNN [56]	0.08421	$O(n \times \sqrt{n})$	N/A
SVM [57]	0.0571	$O(n^2)$	N/A
RNN[4]	0.0515	$O(6)$	Weights
RCGPANN	0.0461	$O(6)$	Weights, Topology, Synaptic connections

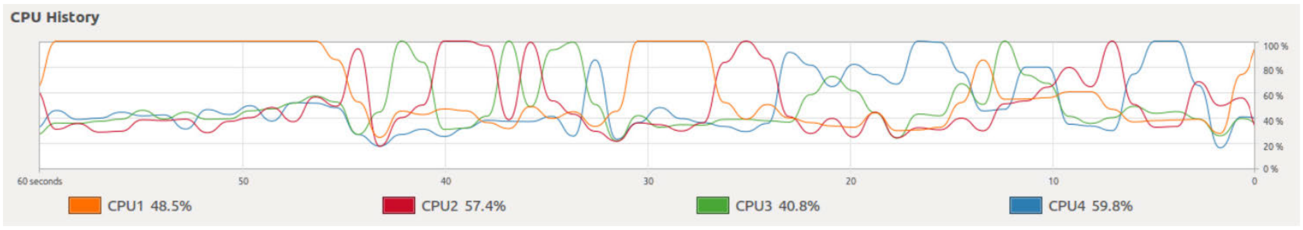

FIGURE 11. CPU usage of each core before prediction and scaling (4 cores active).

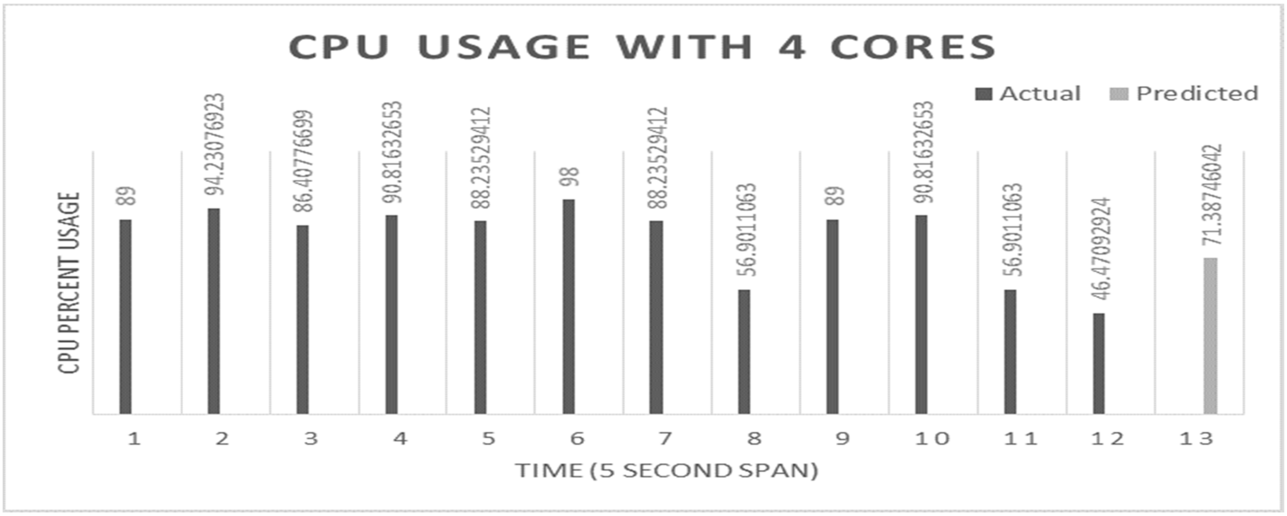

FIGURE 12. CPU usage is sampled at each 5 second time span. Samples taken at 5 seconds to 60 seconds are the actual CPU % usage values, while the sample shown at time 65 seconds is the RCGPANN network based predicted usage.

In the same way, the prediction system predicts future CPU demand with 3-core in ON state, as shown in figure 14. The predicted value is 45.62%, so this load can be handled efficiently by 2-cores. Thus our scaling system scales CPU to 2-cores, as shown in figure 15. CPU core-1 and core-2 are in ON state while core-3 and core-4 are in OFF state, this further validates our prediction and scaling system while minimizing energy consumption.

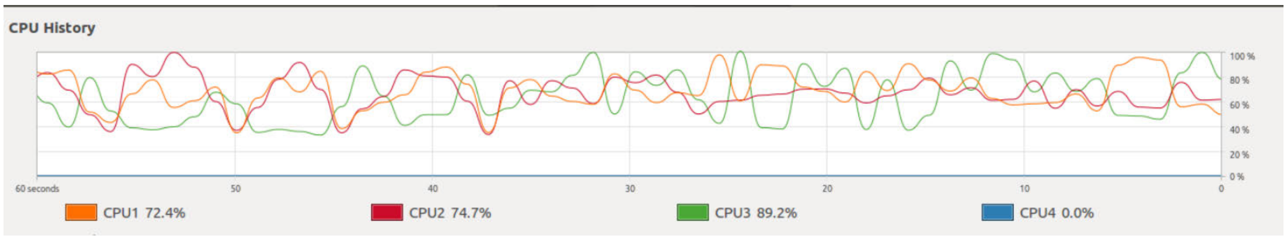

FIGURE 13. CPU usage of each core after prediction and scaling (CPU scaled to 3 cores).

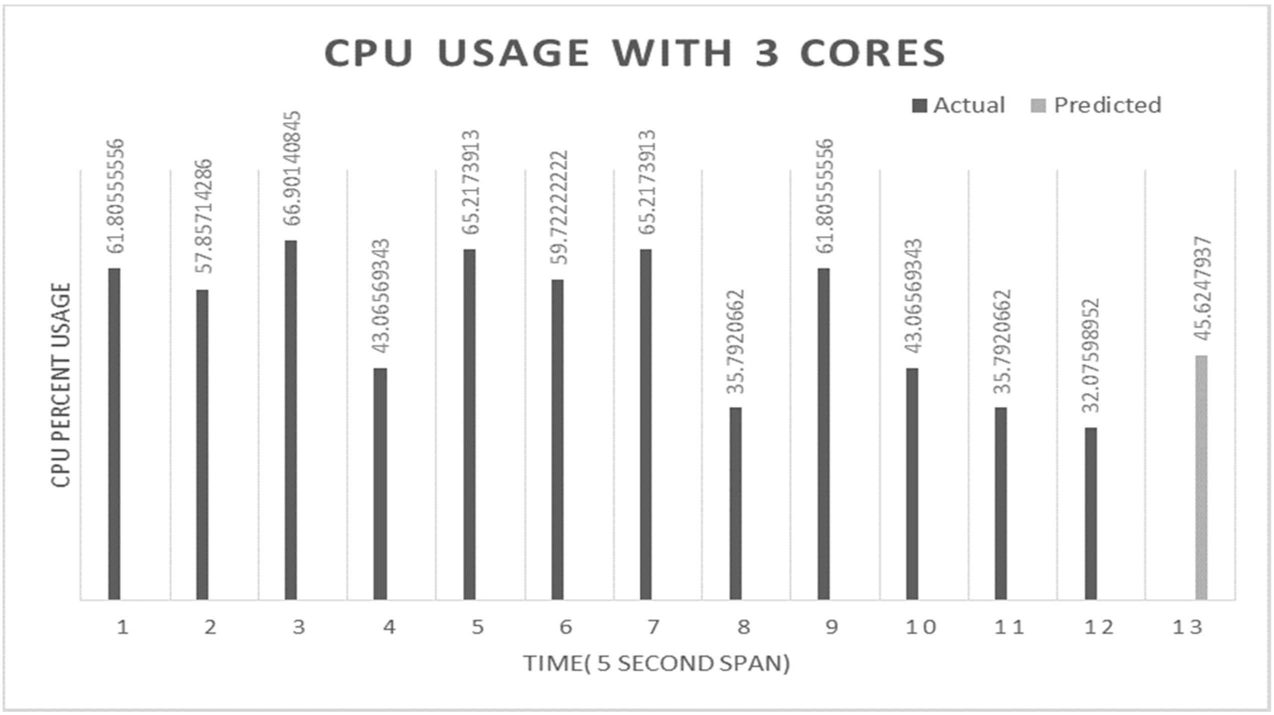

FIGURE 14. CPU usage of 3-cores is sampled at each 5 second time span. Samples taken at 5 seconds to 60 seconds are the actual CPU % usage values, while the sample shown at time 65 seconds is the RCGPANN network based predicted usage.

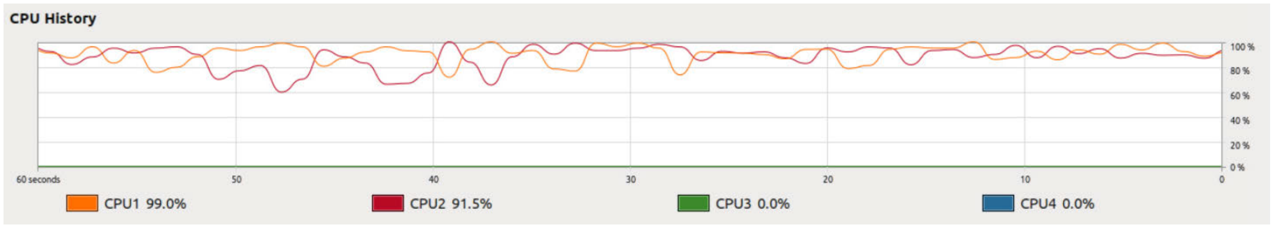

FIGURE 15. CPU usage of each core after prediction and scaling (CPU scaled to 2 cores).

As shown in figure 10, the average power consumption in a duration of 5 minutes by 4-cores is 15.22 watts, by 3-cores is 14.16 watts, by 2-cores is 13.28 watts and by one core is 12.74 watts. Thus, the average power varies with a varying number of cores. Further, the energy consumption in kilo-watt-hour can be calculated by the formula given in (5)

$$\text{Energy(kWh)} = \frac{\text{power(W)} \times \text{time(hour)}}{1000} \tag{6}$$

The energy consumed by a CPU with 4-cores ON for a day, a month and a year will be 0.36528 kWh, 10.9584 kWh and 131.5008 kWh respectively. If there are a hundred CPUs (each with 4-cores ON) in an IaaS server, then yearly energy consumption will be 13150.08 kWh. If the same server has 1-core ON, then yearly energy consumption will be 11016 kWh. Thus yearly maximum energy saving will be 2134.08 kWh if CPUs are scaled from 4-cores to 1-core at low loads.

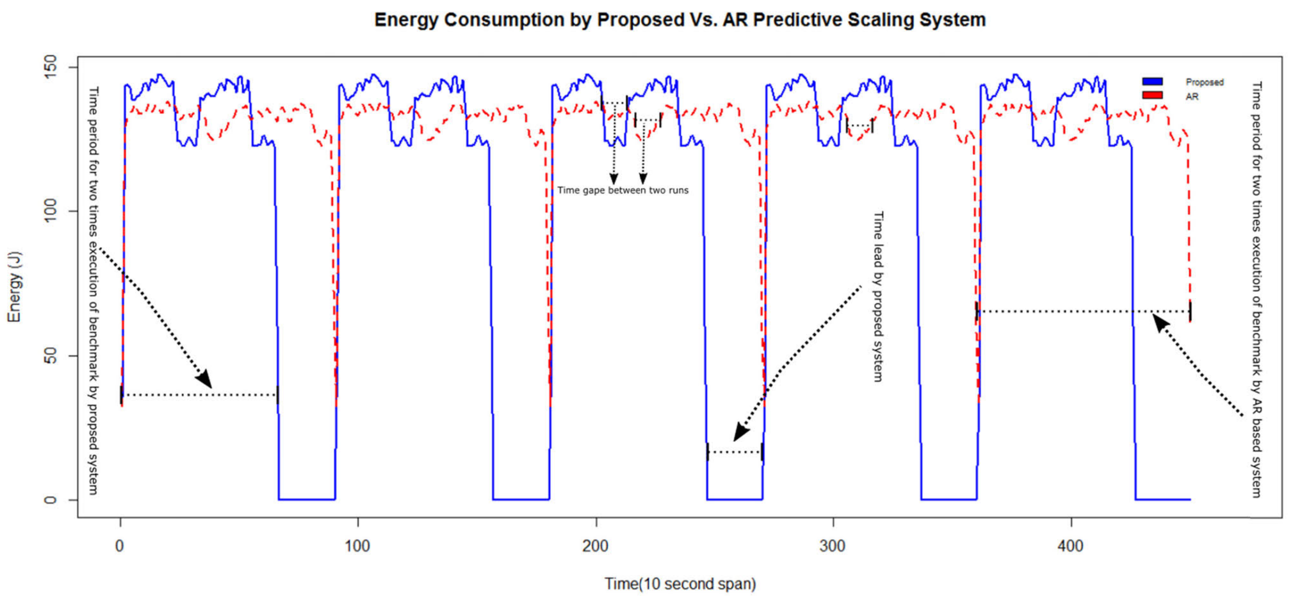

FIGURE 16. Comparison of proposed system with AR based predictive scaling system.

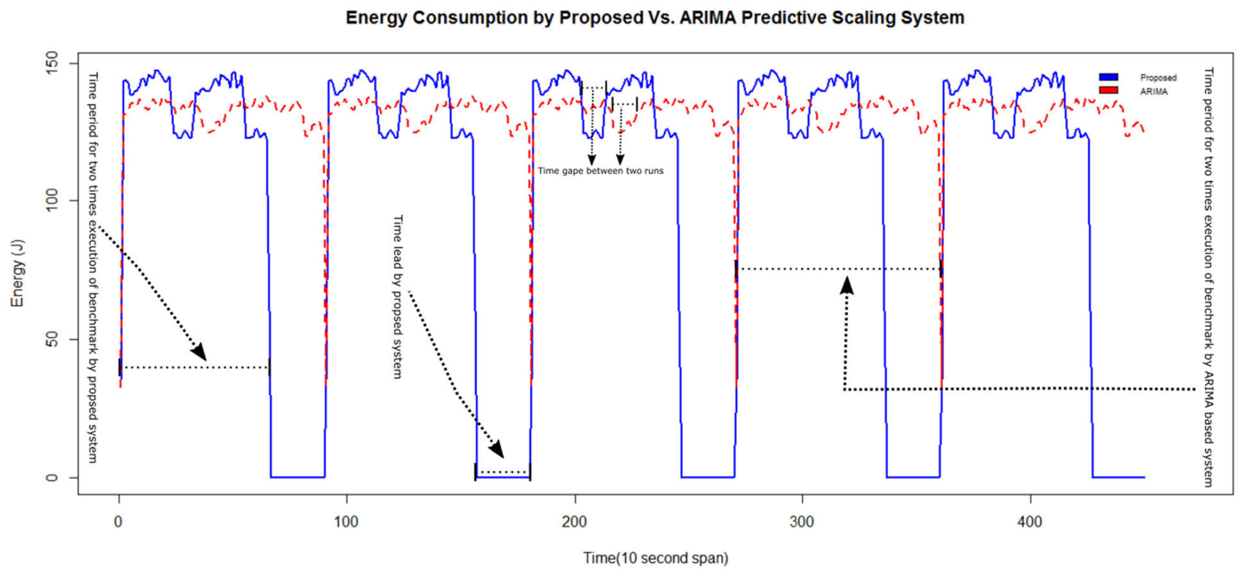

FIGURE 17. Comparison of proposed system with ARIMA based predictive scaling system.

VIII. COMPARISON WITH PREDICTION MODELS FROM LITERATURE

In the literature, several methods have been used for IaaS resource estimation. The notable methods are MLP, ARIMA, ELM, AR, KNN, SVM, AR-NN and RNN [23]–[28] [4]. We have trained and tested prediction models from literature with the same procedure that we used for RCGPANN models training and testing. Table 1 presents a comparison of prediction models under study. Mean absolute error (MAE) of MLP [45] is 0.0761. While for ELM [46], MAE is 0.14041. For ARIMA [47], MAE is 0.13931. Similarly, for AR [45], the prediction error is 0.14031. AR-NN [48] based prediction has MAE value of 0.06112. Prediction error for KNN [57] is 0.08421. SVM [58] has test error value 0.0571. For RNN [4], MAE value is 0.0515. Our proposed RCGPANN has test error value of 0.0461. It is clear from the table that time complexities of KNN and SVM are dependent on the size of data set while other models have constant time complexities. Thus, due to large time complexities, for further study, we consider prediction models other than KNN and SVM.

Further, we discuss the comparison of predictive scaling systems based on MLP, ARIMA, ELM, AR, AR-NN, RNN and No-Scaling system with our proposed predictive scaling system.

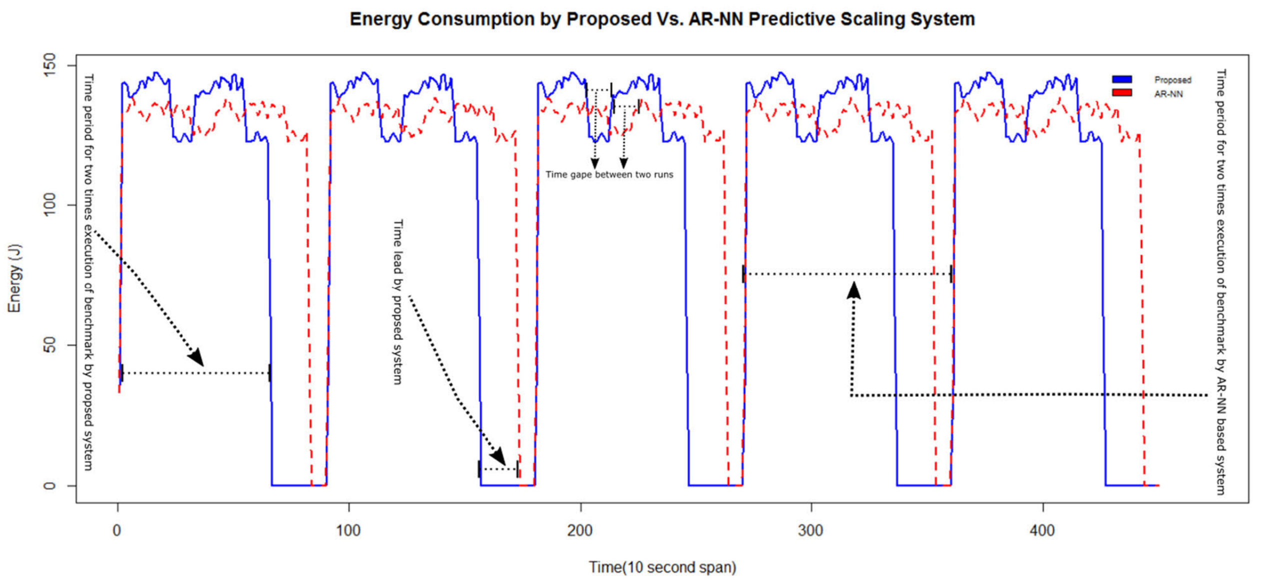

FIGURE 18. Comparison of proposed system with AR-NN based predictive scaling system.

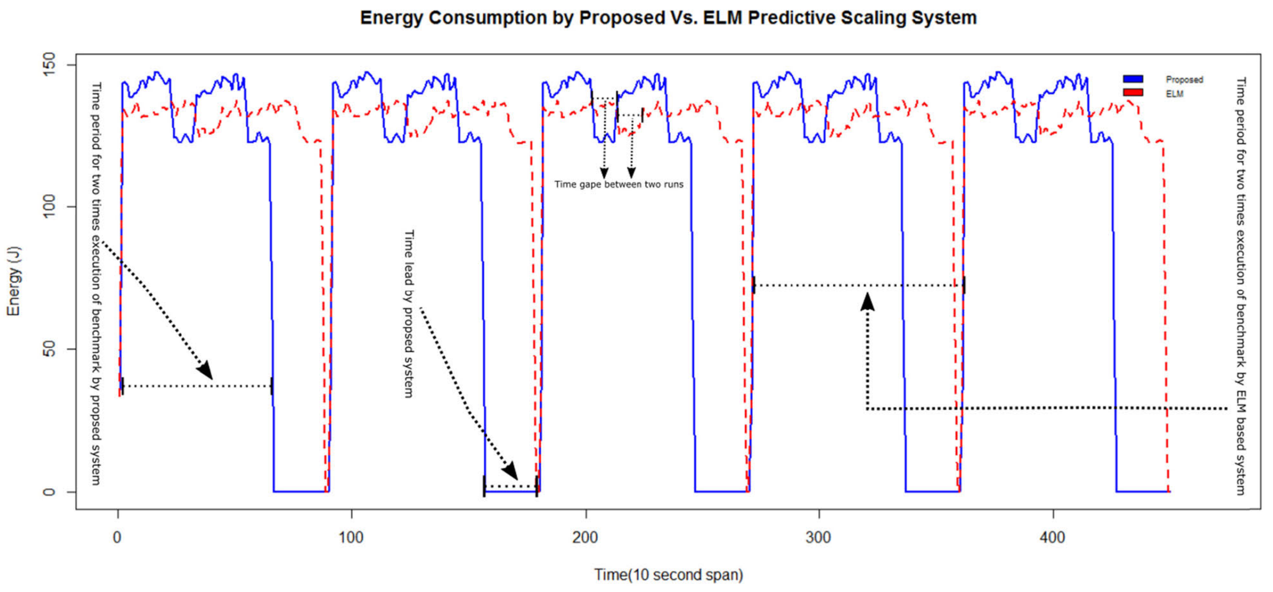

FIGURE 19. Comparison of proposed system with ELM based predictive scaling system.

IX. COMPARISON WITH PREDICTIVE SCALING BASED ON MODELS FROM LITERATURE

In the previous section, we conducted real-time experiments for validation of our predictive scaling system. After validation in real-time, we compare our predictive scaling system with other predictive scaling methods and the no-scaling by executing the same workloads. For performance comparison with the same computation load, a benchmark is required. For this purpose, we used the CPU benchmark of Geekbench.[7] In this section, we first discuss the CPU benchmark of Geekbench, second, we discuss experimental setup, and third, we discuss results of our comparison with related work.

[7]https://www.geekbench.com/

A. GEEKBENCH

We used CPU benchmark of Geekbench for energy comparison of our predictive system with methods from related work and the baseline No-Scaling method. CPU benchmark has four types of workloads (i.e. Cryptography Workloads, Integer Workloads, Floating-Point Workloads and Memory Workloads). Every kind of workload consists of different numbers of workloads each model a real-world task/application. During running the benchmark, each

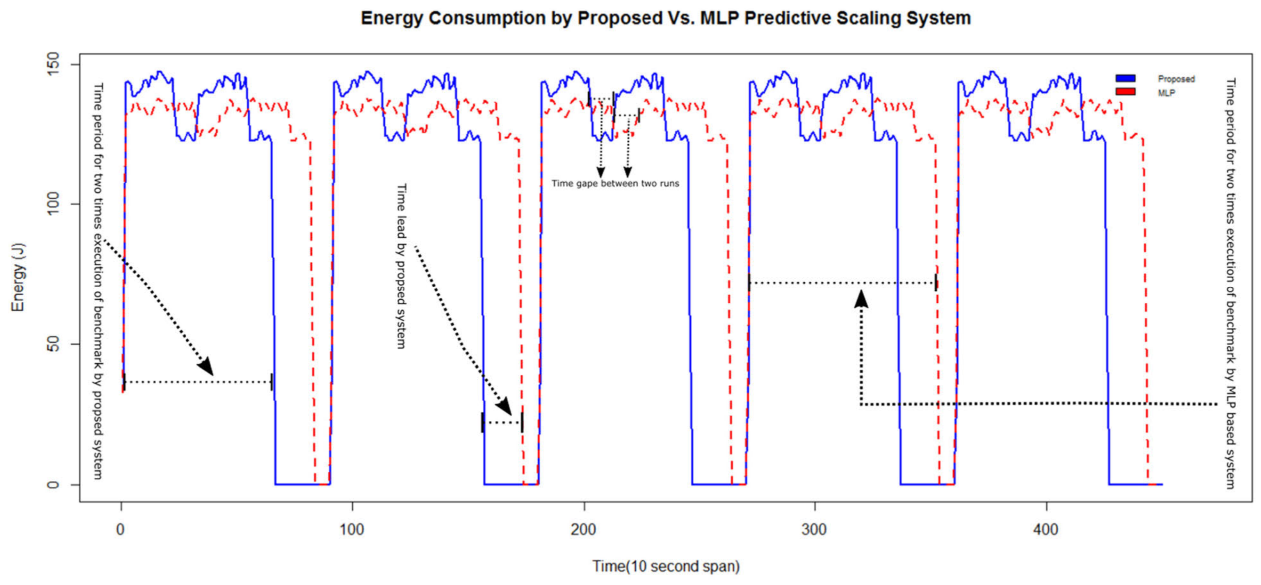

FIGURE 20. Comparison of proposed system with MLP based predictive scaling system.

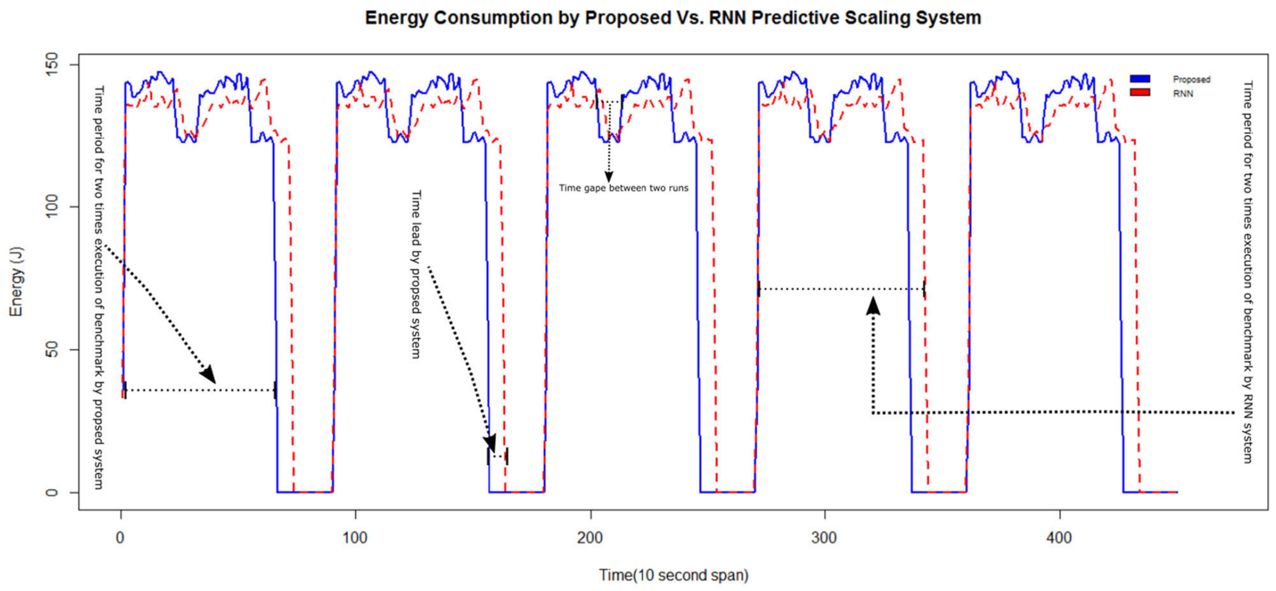

FIGURE 21. Comparison of proposed system with RNN based predictive scaling system.

workload starts after another with a gap of two seconds. This gap minimizes the effect of thermal issues on the performance of workloads.

B. EXPERIMENTAL SETUP

Our experimental setup consists of a computer with CPU @ 1.7 GHz, 4GB RAM and Ubuntu OS. Ubuntu kernel provides a DVFS mechanism with its CPUFreq[8] core. The CPUFreq core provides different frequency scaling governors and frequency scaling policies. We set each CPU core governor to "performance" and policy to "maximum scaling limit". For power measurement, we use the powerstat tool[9] that is used to measure the power consumption of a computer CPU with the battery as a power source. Then, we deploy predictive scaling system (i.e. either our system or from related work) on the system. In this setting, each CPU core runs at a maximum frequency, and total dependence is on the deployed predictive scaling system. For each predictive system and the No-Scaling system, we run CPU benchmark ten times with

[8]https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[9]http://manpages.ubuntu.com/manpages/xenial/man8/powerstat.8.html

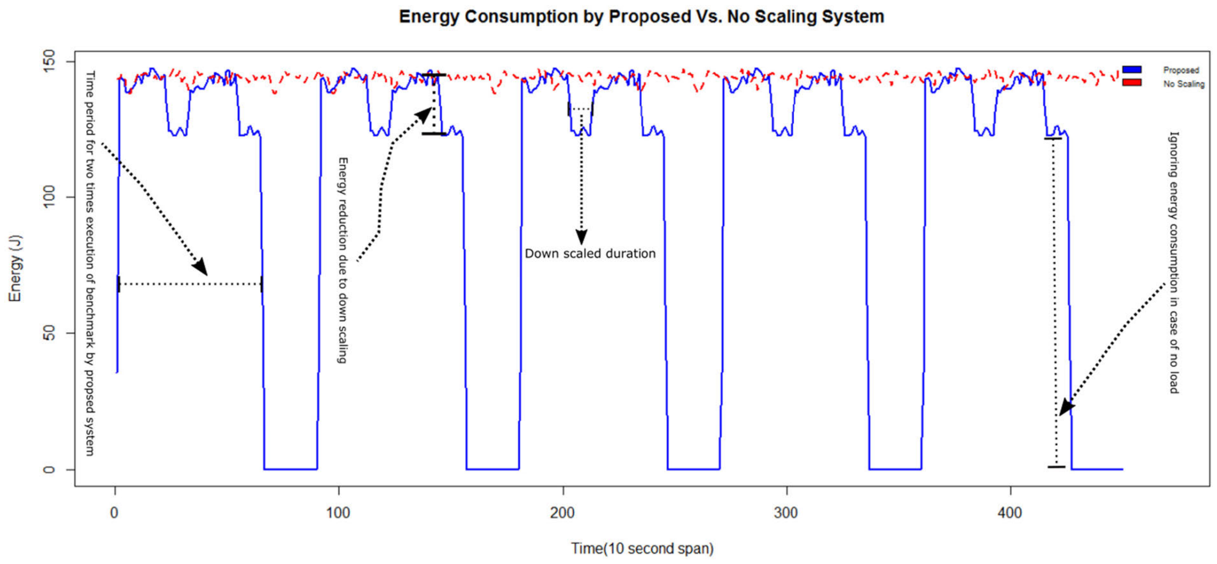

FIGURE 22. **Comparison of proposed system with No-Scaling system.**

a gap of 96.9 seconds (i.e. average setup time for Amazon EC2 VM with Linux based OS) after each run [53].

C. COMPARISON

In comparison experiments, we start by the deployment of AR-based prediction system and run the benchmark for ten times with a delay of 96.6 seconds. We take execution time of two successive runs of AR as a baseline for the remaining predictive system. The break between two consecutive runs is different for each predictive scaling system. This difference shows the execution time capability of each predictive scaling system for the benchmark. Figure 16 presents the energy consumption comparison between proposed and AR predictive scaling systems. The AR-based system consumes a large amount of energy due to its erroneous scaling decisions. It also takes more time in executing benchmark workloads due to scaling CPU to few cores when workload requires more cores to execute. The time taken by the AR system for executing benchmark two times is the baseline time for proposed and other scaling systems under discussion.

Figure 17 presents a comparison of the proposed system with ARIMA based system for energy consumption. ARIMA based system has large scaling overheads due to large prediction errors. The large scaling overheads of ARIMA make the system to consume more energy than the proposed system.

Figure 18 presents a comparison between the proposed and AR-NN based prediction systems. AR-NN has smaller scaling errors than AR, ELM and ARIMA based system. But it has large scaling errors than the proposed system. It takes more time in executing two consecutive benchmark runs than the proposed system. Thus it consumes more energy than the proposed system.

ELM based predictive scaling system takes more time in executing benchmark due to erroneous scaling decisions than the proposed system. Comparison of energy consumption and execution time of benchmark between proposed and ELM based system is shown in figure 19.

Comparison between MLP and proposed predictive scaling systems is shown in figure 20. MLP based system consumes more energy and spends more time than the proposed system due to large prediction and scaling errors.

Figure 21 presents the energy and execution time comparison between RNN and proposed systems. RNN has better prediction accuracy than ELM, MLP, AR, ARIMA and AR-NN based systems. Thus it has performed better in execution time and energy consumption than ELM, MLP, AR, ARIMA and AR-NN based systems. But it consumes more energy than the proposed system due to larger prediction and scaling errors than the proposed system.

Figure 22 presents a comparison of the proposed predictive scaling system with No-Scaling system. In No-Scaling CPU use all cores with maximum capacity. Results show that in the case of high loads, both systems have nearly the same energy consumption. While is a case of low loads proposed system has better energy efficiency due to downscaling of CPU cores. The no-scaling mechanism has better performance than predictive scaling systems with large scaling errors when there are few occurrences of low loads.

Figure 23 presents a summary of energy consumption profiles of all predictive scaling systems and No-Scaling system. The No-Scaling system consumes 15% of the total energy consumed by all systems. Each of the ARIMA, ELM and AR-based predictive scaling systems consumes 13% of the total energy consumption. Each of the MLP and AR-NN based systems consumes 12% of total energy consumption. Predictive scaling system based on RNN consumed 11% of total energy while our proposed system consumes 10% of the total energy.

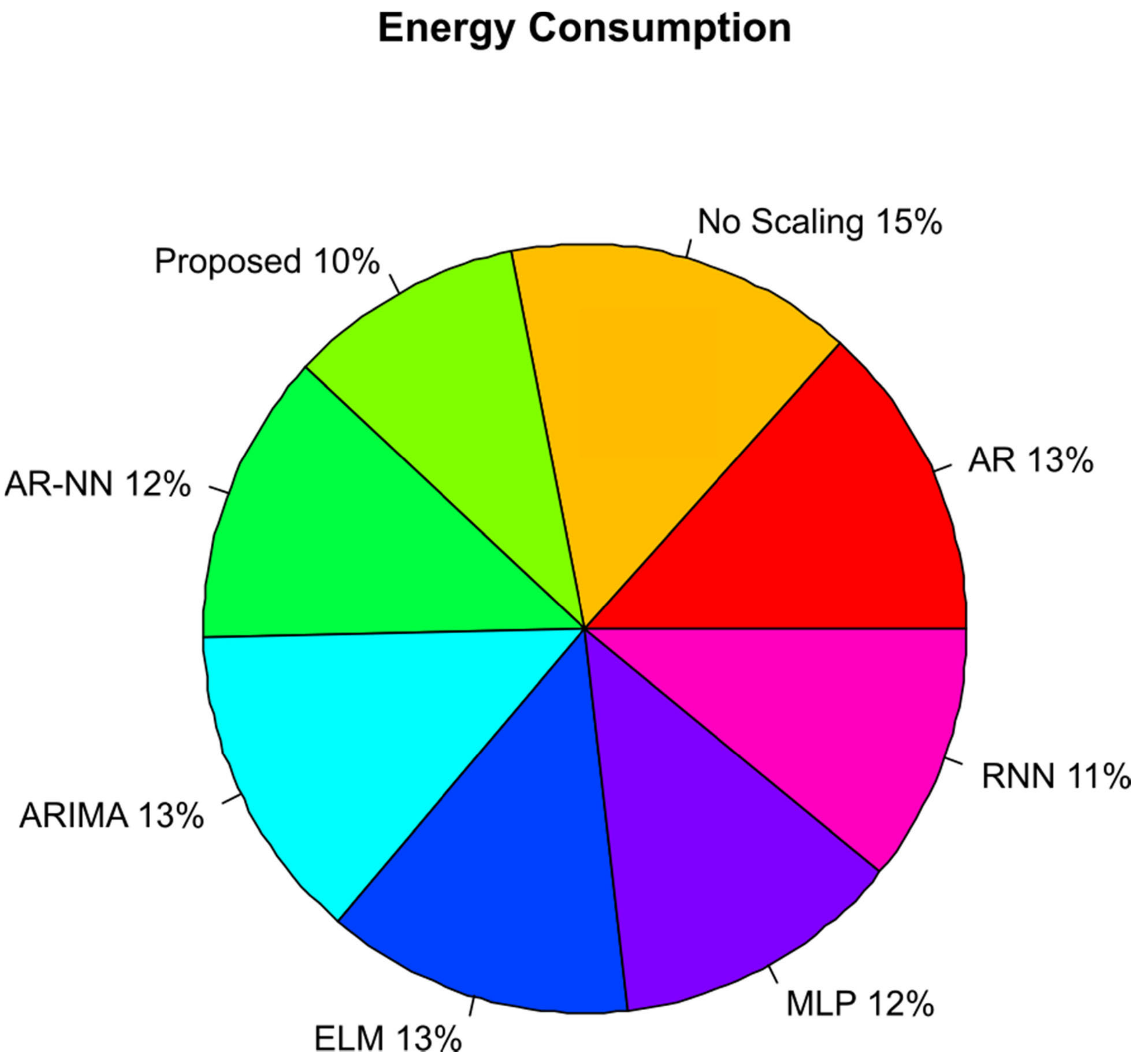

FIGURE 23. Percentage of total energy consumption by each system.

X. CONCLUSION

Our prediction model has shown the best prediction accuracy of prediction models consisting of MLP, ARIMA, ELM, AR, KNN, SVM, AR-NN and RNN. We integrated our prediction model with the scaling system and validated in a real-time environment. We further compared our predictive scaling system with predictive scaling systems based on MLP, ARIMA, ELM, AR, AR-NN and RNN by running CPU benchmark of Geekbench for execution time and energy efficiency. Our predictive scaling system has shown better performance and energy efficiency than predictive scaling systems based on models from literature and the system without the CPU scaling mechanism. Our predictive scaling system has minimized energy by 5% than the no-scaling system, 3% than ARIMA, ELM and AR. It has provided better energy efficiency by 2% than MLP and AR-NN and 1% than RNN.

XI. FUTURE DIRECTIONS

Our directions and recommendations for future studies in solving cloud server resource prediction and scaling problems include enhancement in evolvability and development of predictive scaling system and suitable orientation of the prediction system in a real-time environment. Our current predictive scaling system undergoes offline evolvability and development for prediction and coarse-grained scaling for scaling system (cores or group of cores based scaling). It can be extended to experience online evolvability and growth with fine granularity in scaling (i.e. hardware architecture and OS-dependent).

We recommend virtualization layer level implementation on IaaS server for infusing online evolvability, development and fine granularity in the architecture of predictive scaling system. Physical resources have limited granularity as compared to virtual resources. Also, most types of virtual resources have granular instances (of memory, CPU, storage, etc.) while few physical resources offer (like CPU core and hardware thread level) granularity.

Recommendation for suitable orientation of prediction system includes orientation along with our current scaling system (core wise or node wise) and adjustment along with DVFS system of OS. We recommend, take CPU load directly from CPU scheduler as input for our prediction system instead of resource monitor for faster predictions. Then conduct core-wise scaling decisions according to our scaling system. In case of orientation along with DVFS, feed the predicted CPU load to DVFS system for providing predicted load based DVFS mechanism. These two orientations discussed for OS-layer implementations can also be implemented in the virtualization layer of an IaaS server. In the virtualization layer, due to more granular resources, our predictive scaling system can be applied in the resource level to federated level.

REFERENCES

[1] G. Pallis, "Cloud computing: The new frontier of Internet computing," *IEEE Internet Comput.*, vol. 14, no. 5, pp. 70–73, Sep. 2010.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST Special Publication 800-145, 2011. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[3] *Forecast and Methodology, 2014—2019 White Paper*, CVN. Index, Cisco, San Jose, CA, USA, 2015.

[4] K. Mason, M. Duggan, E. Barrett, J. Duggan, and E. Howley, "Predicting host CPU utilization in the cloud using evolutionary neural networks," *Future Gener. Comput. Syst.*, vol. 86, pp. 162–173, Sep. 2018.

[5] G. M. Wamba, Y. Li, A.-C. Orgerie, N. Beldiceanu, and J.-M. Menaud, "Cloud workload prediction and generation models," in *Proc. 29th Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, Oct. 2017, pp. 89–96.

[6] J. Koomey, "Growth in data center electricity use 2005 to 2010," *New York Times*, Aug. 2011, vol. 9.

[7] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, "United states data center energy usage report," Lawrence Berkeley Nat. Lab., Berkeley, CA, USA, Tech. Rep. LBNL-1005775, 2016.

[8] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, "Understanding and abstracting total data center power," in *Proc. Workshop Energy-Efficient Design*, vol. 11, Jun. 2009.

[9] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[10] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, May 2010.

[11] A. Fox, "Above the clouds: A berkeley view of cloud computing," Dept. Elect. Eng. Comput. Sci., University of California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS28.13, 2009.

[12] N. J. Kansal and I. Chana, "Cloud load balancing techniques: A step towards green computing," *IJCSI Int. J. Comput. Sci.*, vol. 9, no. 1, pp. 238–246, 2012.

[13] I. Manousakis, Á. Goiri, S. Sankar, T. D. Nguyen, and R. Bianchini, "CoolProvision: Underprovisioning datacenter cooling," in *Proc. 6th ACM Symp. Cloud Comput. (SoCC)*, 2015.

[14] A. Beloglazov and R. Buyya, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 7, pp. 1366–1379, Jul. 2013.

[15] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 931–945, Jun. 2011.

[16] R. Brown, *Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431*, Lawrence Berkeley Nat. Lab., Berkeley, CA, USA, 2008.

[17] L. Shang, L.-S. Peh, and K. N. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *Proc. 9th Int. Symp. Highperform. Comput. Archit.*, 2003.

[18] B. Dougherty, J. White, and D. C. Schmidt, "Model-driven auto-scaling of green cloud computing infrastructure," *Future Gener. Comput. Syst.*, vol. 28, no. 2, pp. 371–378, Feb. 2012.

[19] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment," *IEEE Netw.*, vol. 29, no. 2, pp. 56–61, Mar. 2015.

[20] Y. Zhang, W. Sun, Y. Inoguchi, "Predict task running time in grid environments based on cpu load predictions," *Future Gener. Comput. Syst.*, vol. 24 no. 6, pp. 489–497, 2008.

[21] P. A. Dinda, D. R. O'Hallaron, "Host load prediction using linear models," *Cluster Comput.*, vol. 3, no. 4, pp. 265–280, 2000.

[22] K. B. Bey, F. Benhammadi, A. Mokhtari, Z. Guessoum, "CPU load prediction model for distributed computing," in *Proc. 8th Int. Symp. Parallel Distrib. Comput.*, Jun. 2009, pp. 39–45.

[23] J. J. Prevost, K. Nagothu, B. Kelley, and M. Jamshidi, "Prediction of cloud data center networks loads using stochastic and neural models," in *Proc. 6th Int. Conf. System Syst. Eng.*, Jun. 2011.

[24] S. Ismaeel and A. Miri "Multivariate time series ELM for cloud data centre workload prediction," in *Proc. Int. Conf. Hum.-Comput. Interact.* Cham, Switzerland: Springer, Jul. 2016, pp. 565–576.

[25] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications–QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, Oct. 2015.

[26] F. Farahnakian, T. Pahikkala, P. Liljeberg, and J. Plosila, "Energy aware consolidation algorithm based on k-nearest neighbor regression for cloud data centers," in *Proc. IEEE/ACM 6th Int. Conf. Utility Cloud Computing*, Dec. 2013, pp. 256–259.

[27] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Towards an autonomic auto-scaling prediction system for cloud resource provisioning," in *Proc. 10th Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst.*, May 2015, pp. 35–45.

[28] M. Sladescu, "Event aware workload prediction: A study using auction events," in *Proc. Int. Conf. Web Inf. Syst. Eng.* Berlin, Germany: Springer, 2012.

[29] K. Nygren, "Stock prediction-a neural network approach," M.S. thesis, Dept. Math. Phys., Royal Inst. Technol., KTH, Stockholm, Sweden, 2004.

[30] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," in *Proc. 2nd Int. Conf. Cloud Comput. Technol. Sci.*, Indianapolis, IN, USA, Nov. 2010, pp. 456–463.

[31] R. Prodan and V. Nae, "Prediction-based real-time resource provisioning for massively multiplayer online games," *Future Gener. Comput. Syst.*, vol. 25, no. 7, pp. 785–793, 2009.

[32] J. Ali, F. Zafari, G. M. Khan, and S. A. Mahmud, "Future clients' requests estimation for dynamic resource allocation in cloud data center using cgpann," in *Proc. 12th Int. Conf. Mach. Learn. Appl. (ICMLA)*, vol. 2, 2013, pp. 331–334.

[33] R. N. Calheiros and R. Buyya, "Energy-efficient scheduling of urgent bag-of-tasks applications in clouds through DVFS," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2014, pp. 342–349.

[34] S. Eyerman and L. Eeckhout, "Fine-grained DVFS using on-chip regulators," *TACOACM Trans. Archit. Code Optim.*, vol. 8, no. 1, pp. 1–24, Apr. 2011.

[35] R. Nathuji and K. Schwan, "VirtualPower: Coordinated power management in virtualized enterprise systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 265–278, 2007.

[36] Z. Ullah, S. H. Qazi, and G. M. Khan, "Adaptive resource utilization prediction system for infrastructure as a service cloud," *Comput. Intell. Neurosci.*, vol. 2017, Jul. 2017, Art. no. 4873459.

[37] K. Li, "Quantitative Modeling and Analytical Calculation of Elasticity in Cloud Computing," *IEEE Trans. Cloud Comput.*, to be published.

[38] G P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, Jan. 2003.

[39] J. Varia and S. Mathew, *Overview of Amazon Web Services*. Seattle, WA, USA: Amazon Web Services, 2014.

[40] Y. Hu, B. Deng, F. Peng, B. Hong, Y. Zhang, and D. Wang, "A survey on evaluating elasticity of cloud computing platform," in *Proc. World Autom. Congr. (WAC)*, Jul. 2016.

[41] K. Kostantos, "OPEN-source iaas fit for purpose: A comparison between opennebula and openstack," *Int. J. Electron. Bus. Manage.*, vol. 11, no. 3, p. 191, 2013.
[42] W. Dawoud, I. Takouna, and C. Meinel, "Elastic VM for cloud resources provisioning optimization," in *Proc. Adv. Comput. Commun.*, 2011, pp. 431–445.
[43] G. Toffetti, S. Brunner, M. Blöchlinger, J. Spillner, and T. M. Bohnert, "Self-managing cloud-native applications: Design, implementation, and experience," *Future Gener. Comput. Syst.*, vol. 72, pp. 165–179, Jul. 2017.
[44] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2017, pp. 64–73.
[45] A. Ilyushkin, A. Ali-Eldin, N. Herbst, A. V. Papadopoulos, B. Ghit, D. Epema, and A. Iosup, "An experimental performance evaluation of autoscaling policies for complex workflows," in *Proc. 8th ACM/SPEC Int. Conf. Perform. Eng. (ICPE)*, 2017, pp. 75–86.
[46] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mobile Netw. Appl.*, vol. 16, no. 3, pp. 270–284, Jun. 2011.
[47] A. A. Shahin, "Automatic cloud resource scaling algorithm based on long short-term memory recurrent neural network," Jan. 2017, *arXiv:1701.03295*. [Online]. Available: https://arxiv.org/abs/1701.03295
[48] J. F. Miller, "Cartesian genetic programming," in *Proc. Cartesian Genetic Program. Natural Comput.*, 2011, pp. 17–34.
[49] M. Mahsal Khan, A. Masood Ahmad, G. M. Khan, and J. F. Miller, "Fast learning neural networks using Cartesian genetic programming," *Neurocomputing*, vol. 121, pp. 274–289, Dec. 2013.
[50] G. Khan, S. Khan, and F. Ullah, "Short-term daily peak load forecasting using fast learning neural network," in *Proc. 11th Int. Conf. Intell. Syst. Design Appl. (ISDA)*, 2011.
[51] M. Khan, G. Khan, and J. F. Miller, "Evolution of optimal anns for non-linear control problems using Cartesian genetic programming," in *Proc. ICAI*, 2010.
[52] M. Khan, G. Khan, and J. Miller, "Efficient representation of recurrent neural networks for Markovian/non-Markovian non-linear control problems," in *Proc. ISDA*, 2010, pp. 615–620.
[53] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Proc. 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 423–430.
[54] Z. Mwaikambo, A. Raj, R. Russell, J. Schopp, and S. Vaddagiri, "Linux kernel hotplug CPU support," in *Proc. Linux Symp.*, vol. 2, 2004.

QAZI ZIA ULLAH was born in Maidan Bandai, Dir Lower, Pakistan, in 1981. He received the B.S. degree in computer engineering from COMSATS University Islamabad, Abbottabad Campus, the M.S. degree in electrical engineering from the University of Azad Jammu & Kashmir, Mirpur, in 2010, and the Ph.D. degree in computer engineering from Bahria University Islamabad, in 2013.

From 2006 to 2008, he was a Lecturer with the Peshawar College of Engineering, UET Peshawar. From 2008 to 2011, he was a Lecturer with Azad Jammu & Kashmir University. From 2011 to 2012, he was a Lecturer with the COMSATS Institute of Information Technology (now COMSATS University Islamabad), Attock Campus. Since 2012, he has been an Assistant Professor with the Electrical Engineering Department, COMSATS University Islamabad. He is the author of three articles and the reviewer of ten articles. His research interests include resource management in cloud computing and cluster computing environments, evolvable hardware, digital systems design, heterogeneous networks, multiobjective optimization, neural networks, genetic programming, and embedded systems.

GUL MUHAMMAD KHAN received the degree (Hons.) from UET Peshawar, and the Ph.D. degree in intelligent system design from the University of York, U.K. During the Ph.D., he devised a brain inspired learning to learn system that has the capability of learning for itself at runtime. He joined UET as an Assistant Professor on Tenure Track System (TTS), in August 2008. He is the Pioneer of cartesian genetic programming evolved developmental network (CGPDN) providing an indirect method of decoding for neural programs. He is amongst the top neurodevelopmental scientists around the world. He has also introduced a new concept for neuro-evolution, and presented new algorithms for Automatic generation of feed forward (CGPANN) feedback (CGPRNN), Markovian and non-Markovian non-linear systems. He has introduced the concept of plastic networks, and presented algorithms for feed forward and feedback system, termed as plastic CGPANN and plastic RCGPANN. In March 2012, he established a Research Facility, Centre for Intelligent Systems and Networks Research (CISNR), UET Peshawar, where he was appointed as the Director of the Centre, still serving. In a short period of three years, the he obtained research funding of more than 50 million, completed five research projects, with three in progress, supervising over 25 Ph.D./MPhil students and Researchers. He won the Research Productivity award for the year 2017. He is the sole author of a book published by springer titled: Evolution of Artificial Neural Development.

SHAHZAD HASSAN received the M.S. degree in telecommunication engineering from the University of Melbourne, Australia, and Ph.D. degree in information and communication engineering degree from the Huazhong University of Science and Technology, China. He is currently a Senior Assistant Professor with the Department of Computer Engineering, Bahria University, Pakistan. His research interests are WSN, wireless communication, the IoT antenna, cloud computing, and SDN.

• • •