IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 30, 2017, accepted December 2, 2017, date of publication December 19, 2017, date of current version March 9, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2785280

Markov Prediction Model for Host Load Detection and VM Placement in Live Migration

SUHIB BANI MELHEM[1], (Member, IEEE), ANJALI AGARWAL[1], (Senior Member, IEEE), NISHITH GOEL[2], AND MARZIA ZAMAN[2]

[1]Department of Electrical and Computer Engineering, Concordia University, Montreal, QC H3G 1M8, Canada
[2]Cistel Technology, Ottawa, ON K2E 7K3, Canada

Corresponding author: Suhib Bani Melhem (s_banim@encs.concordia.ca)

This work was supported by the Natural Sciences and Engineering Research Council in collaboration with Cistech Ltd., Canada.

ABSTRACT The design of good host overload/underload detection and virtual machine (VM) placement algorithms plays a vital role in assuring the smoothness of VM live migration. The presence of the dynamic environment that leads to a changing load on the VMs motivates us to propose a Markov prediction model to forecast the future load state of the host. We propose a host load detection algorithm to find the future overutilized/underutilized hosts state to avoid immediate VMs migration. Moreover, we propose a VM placement algorithm to determine the set of candidates hosts to receive the migrated VMs in a way to reduce their VM migrations in near future. We evaluate our proposed algorithms through CloudSim simulation on different types of PlanetLab real and random workloads. The experimental results show that our proposed algorithms have a significant reduction in terms of service-level agreement violation, the number of VM migrations, and other metrics than the other competitive algorithms.

INDEX TERMS VM live migration, host overload/underload detection, VM placement, CloudSim.

I. INTRODUCTION

With immense success and fast growth within the past few years, cloud computing has been established as the dominant computing paradigm in information technology (IT) industry, wherein it utilizes dissipated resource benefits and supports resource sharing and time access flexibility. Therefore, to effectively manage applications and resources it is crucial to use the models and tools that create an application profile, which is used to apply optimal models to determine the most suitable amount of resource for each workload. Virtual machines migration is one of the most popular ways to manage resources, and live VM migration is the most used technique to reload or rearrange the resources in the data center to keep the delivered services available. Live VM migration is defined as a technique that migrates the entire operating system and its associated applications from one host to another where a user does not notice any interruption in his service. It plays an important role to facilitate online maintenance, load balancing, and energy management as part of resource management [1].

In modern data centers, resource management and allocation during VM migration has become more challenging due to their rapid growth, high dynamics of hosted services, resource elasticity, and guaranteed availability and reliability. Thus, the performance of applications in large virtualized data centers is highly dependent on data center architecture and smooth network communication among VMs. The communication cost of a network can be reduced by minimizing the VMs migration between hosts. Therefore, clients and service providers need to build a cloud computing infrastructure that does minimize not only operational costs but also total network load.

The resource utilization of a data center may change over time due to a creation of new VMs and/or hosts, or due to a failure of existing hosts, or due to the removal of existing VMs. There is a need to reorganize the VMs and the hosts to provide load balancing or server consolidation depending on the SLA with the end users and other issues. In cloud data center management, the three most important research problems that have been addressed in the live migration are the host overload/underload detection, VM selection, and VM placement.

In the first phase, host detection, a host may be in an overloaded state or in an underloaded state. If a host is underutilized, then all the VMs from this host can be migrated and the host will go to sleep/shutdown mode, or the host will be considered as a good candidate to receive the migrated VMs from the overloaded hosts in the future. On the other hand,

when a given host is overloaded some VMs must be selected to migrate from this host to other hosts. The challenges in the host overload/underload detection are to reduce the power consumption, minimize SLA violation, and to avoid performance degradation.

Once a decision to migrate VMs from a given host is made, VM selection phase selects one or more VMs from the full set of VMs running on the host. The selected VMs must be moved to other hosts. VM selection approaches are different based on the parameters that are considered to select the migrated VMs. The challenge in choosing one or more VMs for migration is a vital decision for resource management. The VM migration process consumes the network and CPU resources from both source and destination hosts besides making the VM unavailable for a certain amount of time. The performance of other VMs that are running on source and destination hosts are also affected due to increased resource demands during the VM migration process.

Finally, a given VM placement algorithm is applied to selected underloaded hosts to receive the migrated VMs. Many factors should be considered to develop a new optimal VM placement algorithm, such as the resource availability of host (i.e., CPU, memory, disk storages and network bandwidth), the total energy consumption in the data center, and inter-VM traffic. The goal of VM placement is to deliver best possible QoS to the applications running on VMs. Once a decision to migrate a VM from a source host to a destination host is known as a result of the selection process, then the migration process will take place either locally or widespread [30].

Algorithm 1 illustrates the overall live migration procedure based on the host status which can be either an overloaded or an underloaded state. In the overload host detection procedure, one of the host detection algorithms is applied to determine if the host is overloaded. In case a host is overloaded, a Boolean variable called *migration_decision_overloaded* is continuously checked until the required number of VMs are selected and stored in *vmstomigrate*[] array. The selection can be done using any VM selection algorithm. Active hosts that currently carry VMs are determined using *getactivehosts* function. One of the VM placement algorithms is applied to map selected VMs to destination hosts.

In the underload host detection procedure, one of the host detection algorithms is applied to determine if the host is underloaded. In case the host is underloaded, there is no VM selection process. All the VMs in the underloaded host must be migrated. The host is switched to an idle state.

The main contribution in this paper is to introduce efficient algorithms by studying host detection and VM placement. Existing algorithms consider the trade-off between power consumption and SLA violation at the current host state. Our proposed algorithms consider both the current host utilization state and the future host utilization state. Markov based prediction algorithms are embedded in a way detailed

Algorithm 1 Live Migration Procedure

```
1  Input: Host
2  Output: Do certain procedure based on the host status
3  //Overloaded host detection procedure
4  boolean migration_decision_overloaded ← false
5  boolean selection_is_done ← false
6  migration_decision_overloaded
   ← DetectionPolicy(host_is_overloaded())
7  while migration_decision_overloaded = true do
8      //One of the VM selection algorithms is applied
       vmstomigrate[] ←
       SelectionPolicy(host).add
9      migration_decision_overloaded ←
               DetectionPolicy(host_is_overloaded())
10     selection_is_done ←true
11 if selection_is_done = true then
12    Activehosts[] ← getactivehosts()
13    PlacementPolicy(vmstomigrate[], Activehosts[])
14 //Underloaded host detection procedure
15 boolean migration decicion_underloaded ← false,
16 migration_decision_underloaded
   ← DetectionPolicy(host_is_overloaded())
17 if migration decision_underloaded = true do
18     vmstomigrate[] ← allvms
19     Activehosts[] ← getactivehosts()
20     PlacementPolicy(vmstomigrate[], Activehosts[])
```

in the sections below in the host detection and VM placement algorithms. Our contributions are summarized as follows:

- In contrast to the existing VM consolidation and load balancing methods which mostly rely on the current resource utilization of hosts, Markov chain model considers both current and future resource utilization. In order to predict the future utilization, we used the first-order Markov chain model to build Markov host prediction model.
- We propose a host load detection algorithm called Median Absolute Deviation Markov Chain Host Detection algorithm (MadMCHD) to find the future overutilized hosts state and the future underutilized hosts state for better host detection performance in the live migration. In addition, we propose an efficient prediction algorithm to enhance VM placement process. We improve the live migration process by combining the proposed algorithms for better performance.
- We implement and evaluate Markov host prediction model and the algorithms on a simulated large-scale data center using the real PlanetLab workloads and random workload.
- We study the impact of the data length of host status history in our algorithms such that they perform the best on four well-known VM selection methods found in the literature. In addition, we investigate how the four VM selection methods have impact on the performance

in terms of the energy consumption, the number of SLA violations, the number of migrations, and other metrics as outlined in Section V(C).

This paper starts by introducing related works in Section II. Section III explains our proposed forecasting model. Section IV presents our proposed host load detection algorithm, VM placement algorithm and the system architecture. Section V presents our experimental setup, performance metrics and baseline algorithms. In section VI, experimental results are analyzed. Section VII shows the concluding remarks and future directions.

II. RELATED WORK

Over the last two decades, there has been major significant research in data center resource management and allocation during VM migration. Many host over-load/underload detection algorithms have been pro-posed to generate optimal computing resource utilization and energy consumption reduction along data center. Authors in [2] proposed the averaging threshold-based algorithm (THR). It computes the mean of the n last CPU utilization values and compares it to the previously defined threshold. The algorithm detects under-loaded state if the average of the n last CPU utilization measurements is lower than the specified threshold. This algorithm is unsuitable for a dynamic environment.

In [3]–[5] authors proposed four policies in two categories. The first category is adaptive utilization threshold based algorithms that include two policies: Median Absolute Deviation (MAD) and InterQuartile Range (IQR). These policies offer auto-adjustment of the utilization thresholds based on a statistical analysis of historical data obtained during the lifetime of the VMs. The objective is to alter the value of the upper utilization threshold based on the strength of deviation of the CPU utilization. MAD is defined as a measure of statistical dispersion that performs better with distributions without a mean or variance. Also, it is a more robust estimator of scale in comparison to sample variance or standard deviation. The main disadvantage in MAD is that the magnitude of the distances of a small number of outliers is inappropriate. IQR is another measure of statistical dispersion. It is called the midspread or middle fifty which means the difference between the third and first quartiles in descriptive statistics. This category has a poor prediction of host overloading. Moreover, when a host has encountered the same CPU utilizations in the past, the value of the threshold in these approaches is measured around 100%, resulting in a more aggressive consolidation of VMs and more SLA violation.

The second category is Regression based algorithms that include two policies: Local Regression (LR) and Robust Local Regression (RLR). These depend on the predicting of the future CPU utilization. They perform better forecasting of host overloading but has more complexity. LR is an approach that fits a curve that shows the trend in the data. A host is overloaded in case the maximum migration time is closer than a safety margin to the trend line. In RLR, a comparison between maximum migration time and expected value is weighted, before making the decision of overloading in the host. This category is influenced by the presence of outliers and does not reflect the behavior of the bulk of the data.

Many more algorithms for the host load detection have been proposed in the literature [6]–[11], [36]. Most of the existing algorithms depend only on the historical data of the data center to determine the static or dynamic threshold. In this paper, we use the historical data to build probabilistic model that can predict the future host load more efficiently.

Average traffic latency reduction is the objective that researchers in [12] concentrate on; to achieve this objective a traffic aware VM placement algorithm has been proposed. Two traffic models have been proposed, namely known as partitioned and global. In partitioned model, the only allowed communication is the one between the VMs in the same partition, whereas in the global traffic model, the communication is not constrained on the VMs in the same partition with a constant flow rate. In this algorithm, better network scalability is satisfied by reducing the traffic going through the switches. This can be explained by the fact that this algorithm is moving the VMs through a minimum number of switches.

In [13] authors formulated the VM placement problem as a multi-objective optimization problem to minimize total resource wastage, energy consumption, and thermal dissipation cost. The authors proposed an improved genetic algorithm with a fuzzy multi-objective evaluation to search for solutions for allocating VMs.

In [14] researchers proposed a VM placement algorithm based on the Ant Colony Optimization (ACO) meta-heuristic where the placement is computed in a dynamic way according to the current load by modeling the workload consolidation problem as an instance of the Multidimensional Bin-Packing (MDBP) problem. The goal of this algorithm is to pack the VMs into fewer servers. The algorithm needs the knowledge about all the workload and its related resource requirements to compute the placement.

Authors in [15] formulated the VM placement problem as a multi-objective Ant Colony Optimization (ACO) algorithm to minimize SLA violation, total resource wastage, and power consumption. In ACO algorithm, each ant constructs a solution for selecting VM to the target server. The constructed solution is estimated by the suitable function, which combines SLA violation, resource consumption, and power consumption.

In [16] authors proposed a joint energy-aware and application aware VM placement strategy based on the theory of multi-objective optimization by exploring a balance between server energy consumption and the communication network energy consumption in the data center. The algorithm aims to meet the conditions of the server-side constraints, to minimize network data transmission, and to reduce power consumption in data centers. The considered parameters are the distance between the switches that interconnect physical hosts, constraints of servers and the application dependencies among VMs of composite applications.

In [17] researchers applied the genetic algorithm to address the VM placement problem considering reducing energy consumption and the communication network among hosts. The authors present a VM placement model considering two functions. The first function is a linear function of its workload that shows the energy consumed by a server and the energy consumed when the server is idle. The second one is a function of the amount of data exchanged among the VMs that displays energy consumed by the network.

In [18] authors proposed a hybrid genetic algorithm (HGA). The HGA algorithm approach is used to allocate VMs efficiently than the genetic approach in [16]. A repairing procedure is embedded for converting the proposed solution into a feasible one. This can be accomplished by the means of local optimization procedure and resolving the existing violations in order to improve the overall quality of a solution.

Researchers in [19] proposed Family Genetic Algorithm (FGA) for VM placement to overcome the limitations of the Genetic approaches [17], [18]. These limitations are the premature convergence and the high processing time. In [21] researchers proposed VM Scheduler placement algorithm to reduce the time of allocation of VM to the server and to optimize the resource utilization. The algorithm represents the list of resources in a binary search tree (BST) instead of representing them in a queue. The algorithm generates two BSTs, one for VM specs and one for hosts. The VM scheduler takes the VM that has the maximum requirement and searches for a candidate host which best fits the requirement of VM.

In [3] and [20] authors proposed Power Aware Best Fit Decreasing (PABFD) algorithm for VM placement to move the VMs from the overloaded host to underloaded host or from underloaded host for server consolidation. The algorithm selects the destination host to receive the migrated VM that causes the least increase in the power consumption. The algorithm relies on the traditional greedy algorithm to optimize the allocation of VMs.

In [21] researchers proposed host utilization and minimum correlation (UMC) VM Placement Algorithm to reallocate VMs from overutilized hosts or from underutilized host. The considered parameters are host utilization and the correlation between the resources of a VM with the VMs present on the host correlation. The algorithm selects the destination host to receive the migrated VM if its CPU utilization has the lowest correlation with all VMs CPU utilization on that host.

In conclusion, the main disadvantage of the existing work found in the literature for both the VM placement and host detection algorithms is their reliance on the current host resource utilization. We propose to use historical data to build probabilistic model that can predict the future host load more efficiently. We present a Markov-based VM placement and host load detection approaches, respectively with the objective to allocate a VM based on the current and future resource utilization of host and VMs to mitigate the unneeded VM migrations for better SLA violation, number of VM migrations and the energy consumption in the whole system.

III. PROPOSED MARKOV HOST PREDICTION MODEL

This section explains the forecasting model used to effectively decide whether it is really necessary to migrate a VM depending on the present as well as the predicted future load based on previously observed values using Markov model prediction technique [22].

In the Markov chain, the observed variable W is discretized, so the observation sequence $w_1, w_2, \ldots, w_n$ can be described using a discrete scalar observation sequence $\{w_1, w_2, \ldots, w_n\}$ as proposed in our forecasting model, the last w observations of a given host CPU utilization, where each of the variables w_n may take one of M different states $\{S_1, S_2, \ldots, S_M\}$. In our proposed Markov model, in the proposed algorithms, three different states for a given host are possible, namely underloaded (U), normal loaded (N) and overloaded (O).

The Markov model will be used to model the host detection depending on historical data that will be maintained in a log file. The historical training set is stored in a database, in our forecasting model the prediction will take place when we have at least 10 historical data observations stored in the database, the number is used in other algorithms [3]–[5]. The Markov model is built using three states given by $\{S_1 = \mathrm{U}, S_2 = N, S_3 = O\}$. The stochastic variable χ is a discrete random variable taking one of these three values, where Algorithm 2 will be applied periodically by each host manager to find χ for each observation and register it in the host log file.

Algorithm 2 Host Detection State

Input: host CPU utilization of host j ($CPUu(H_j)$, lower threshold, and upper threshold.
Output: χ (current host state).
If $CPUu\left(H_j\right) \leq$ *lower threshold* ***then***
    $\chi \leftarrow U$
else If *lower threshold* $<$ $CPUu\left(H_j\right)$ $<$ *upper threshold* ***then***
    $\chi \leftarrow N$
else If $CPUu\left(H_j\right) \geq$ *upper threshold* ***then***
    $\chi \leftarrow O$
return χ

Algorithm 2 shows the pseudo-code of host detection state for each observation. Three parameters are inputs to this algorithm. The first parameter is the host CPU utilization, which is calculated by dividing the total MIPS requested on the total host MIPS. The other parameter is the lower threshold, which is assigned a value of 0.1. The upper threshold is taken from the MAD algorithm which is explained in [5]. Each host has an underloaded (U), over- loaded (O) or normal loaded (N) state, which can be easily found by comparing the current CPU utilization value ($CPUu$) by the lower and upper thresholds. After the host load state is determined it is stored in the log file in order to be used in our proposed Markov prediction algorithm.

It should be noted that the first-order Markov chain is most widely used in describing dynamic processes, wherein the conditional probability of an observation w, at time n (i.e., w_n) only depends on the observation, w, at time $n-1$ (i.e., w_{n-1}) as shown in (1). Moreover, the joint probability of n observations, $P(w_1, w_2, \ldots, w_n)$ using the first order Markov chain can be given by (2). Our Markov detection algorithm starts working after collecting 10 historical observations (n = 10).

$$P(w_n|w_{n-1}, w_{n-2}, \ldots, w_1) \approx P(w_n|w_{n-1}) \quad (1)$$

$$\mathrm{P}(w_1, \ldots, w_\mathrm{n}) = \prod_{i=1}^{n} P(w_i|w_{i-1}) \quad (2)$$

The conditional probabilities $p(w_n = S_j|w_{n-1} = S_i)$ are referred to as state transition probabilities or simply transition probabilities. The transition probabilities describe the probability of the system at state S_j at time n given that the system was at state S_i at time $n-1$. In most cases, we assume that the transition probabilities are homogeneous, which means that the probabilities do not change over time, so

$$p\left(w_n = S_j|w_{n-1} = S_i\right) = p(w_{n+T} = S_j|w_{n-1+T} = S_i) \quad (3)$$

where T is a positive integer larger or equal to one. The transition probabilities can be written as a transition matrix, which is of dimension $M * M$ for a system with M (*where* $M = 3$) different states $\{S_1, S_2, \ldots\ldots, S_M\}$.

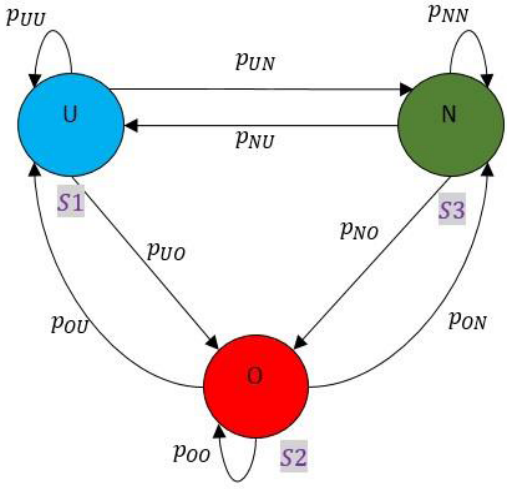


FIGURE 1. States and transition probabilities of the host detection Markov model.

The state and transition probabilities of a given Markov chain can be shown using graph. Fig. 1 shows our host detection Markov model with three discrete states {O, U, N} with every periodic time we would transit to a (possibly) new state based on the probabilities in (4). The system model starts in one of these states and moves successively from one state to another. Each move is called a step. The probability p_{ij} represents the chance of the system model to be in the current state S_i and moves to next state S_j.

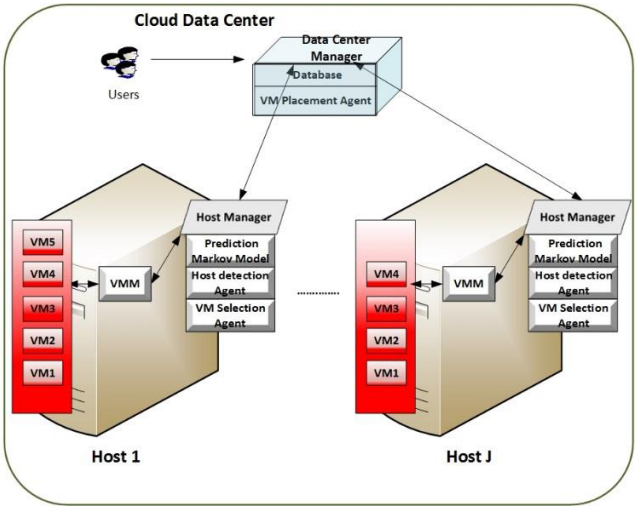


FIGURE 2. System architecture.

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1M} \\ p_{21} & p_{22} & \cdots & p_{23} \\ \vdots & \vdots & & \vdots \\ p_{M1} & p_{M2} & & p_{MM} \end{bmatrix}$$

$$= \begin{array}{c} \\ U \\ N \\ O \end{array} \begin{array}{c} \begin{array}{ccc} U & N & O \end{array} \\ \begin{bmatrix} p_{UU} & p_{UN} & p_{UO} \\ p_{NU} & p_{NN} & p_{NO} \\ p_{OU} & p_{ON} & p_{OO} \end{bmatrix} \end{array} \quad (4)$$

Instead of immediately migrating some of its VMs we can check whether the migration is required or not. The algorithm takes states and transition probabilities of a given host j detection from Markov model as an input and makes the decision of migration and the decision of hosting VMs as an output. The decision is based on the current CPU utilization and the future CPU utilization.

IV. PROPOSED SYSTEM

In this section, we present Markov-based host detection and VM placement algorithms for cloud data center. In section A, our proposed system architecture is explained. The overload/underload host detection algorithm and VM placement algorithm are explained in section B. Finally, an illustrative scenario clarifies the algorithms.

A. SYSTEM ARCHITECTURE

The target system is an IaaS environment, represented by a large-scale data center. The data center consists of $\leq$ J heterogeneous hosts where each host contains multiple VMs. Multiple VMs can be allocated to each host through Virtual Machine Monitor (VMM). Besides, each host and VM are characterized by the CPU performance metrics defined in term of Millions Instructions Per Second (MIPS), the amount of RAM and network band-width. The target system model is depicted in Fig. 2 which is a modified version of the

model described in [35]. Our model includes two important parts: A Data Center manager that has an extra predictive VM placement functionality, and the Host Manager that has an extra Markov model prediction agent for host detection.

Fig. 2 shows the Host Manager and the Data Center Manager components. Host manager resides on every host for keeping continuous observation on CPU utilization of the node. Data center manager interacts with the host managers.

Host Manager consists of the following components:

- Host detection agent: responsible for detecting the current load state of the host, which can be either underloaded or overloaded.
- VM selection agent: responsible for finding the VM that has to be migrated.
- Prediction Markov model: responsible for finding the future load state of the host.
- VMM: responsible for monitoring host as well as sending gathered information to the data center manager. In addition, VMM performs actual resizing and migration of VMs as well as changes in power modes of the PMs.

Data Center Manager consists of the following components:

- VM placement agent: responsible for performing the migration from overloaded/underloaded hosts to the candidate hosts based on a predictive Markov model.
- Database: data structure that contains all the information about the hosts and the utilization of each host.

Our proposed algorithms suggest that the load state host detection algorithm and the VM placement algorithm should not only depend on the current overall rewards gained from migrating the VMs, but also the future rewards should be taken into consideration for better SLA violation, and number of VM migrations. Host manager interacts with the VMM manager in order to initiate the VM migration process after finishing the host detection, and VM selection processes. It also interacts with the data center manager in order to initiate the VM placement.

B. THE PROPOSED WORK

The problem of VM migration can be divided into four parts: (1) determining which hosts are overloaded; thus one or more VM migration is required from the host under consideration, (2) determining which hosts are underloaded so that all VMs should be migrated from those hosts; (3) Selecting VMs that should be migrated from overloaded hosts. (4) finding new placement for the migrated VMs by choosing the good candidate hosts [22]. We have proposed three algorithms which resolve the first, second and fourth issues of migration. For VM selection multiple selection algorithms given in [4] are used.

1) HOST UNDERLOAD/OVERLOAD DETECTION

Algorithm 3 describes the host overload/underload detection mechanism. Upper and lower thresholds for CPU utilization are assigned first. These can be assigned either statistically

Algorithm 3 Overload/Underload Host Detection

```
1  Input: host, lower threshold = 0.1, upper threshold =
   0.9, B (FOMCHSD or MadMCHD).
2  Output: migration_decision_underloaded (T/F),
   migration_decision_overloaded (T/F).
3  migration_decision_underloaded ← false
4  migration_decision_overloaded ← false
5  while hostactive = true do
6   if logfile.Length >= 10 then
7    //calculate current CPU utilization of host h
8    utilization ← total Req. Mips/ Total host Mips
9   Switch(B)
10   Case FOMCHSD : break;
11   Case MADMCHD :
12        upper threshold ← 1 − s ∗ MAD
13  //find current utilization using Algorithm 1
14   current state ←
      host_detection_state (utilization, lower threshold,
                                        upper threshold)
15  //find future utilization using Markov prediction
    technique
16  future state ← future_Markov_utilization_state
    (current state)
17  If future state = O then
18    migration_decision_overloaded ← True
19  else If current state = U and future state = U then
20    migration_decision_underloaded ← True
21  return migration_decision_underloaded,
          migration_decision_overloaded
```

using First Order-Markov Chain Host State detection algorithm (FOMCHSD) or dynamically using Median Absolute Deviation Markov Chain Host Detection algorithm (*MadMCHD*). In *MadMCHD*, Median Absolute Deviation (MAD) algorithm is used, which is based on statistical analysis of historical data collected during the lifetime of VMs [3]. For a univariate data set $w_1, w_2, \ldots, w_n$, the *MAD* is defined as the median of the absolute deviations from the median of the data set:

$$MAD = median_i(|w_i - median_j(w_j)|) \tag{5}$$

The MAD is the median of the absolute values of deviations (residuals) from the data's median. In the proposed overload detection algorithm, the upper CPU utilization threshold (Tu) is defined as given in (6)

$$Tu = 1 - s * MAD \tag{6}$$

where $s \in R^+$ is a parameter of the method defining how strongly the system tolerates host overloads. In other words, the parameter s allows the adjustment of the safety of the method: a lower value of s results in a higher tolerance to variation in the CPU utilization.

After our algorithm is triggered, the first thing to calculate is the current CPU utilization, and then to determine whether the static or dynamic values are considered for the upper

and the lower threshold by checking the value of the input parameter B. As mentioned before, the values of the upper and lower values are assigned statically or dynamically using *MAD*. In case B = *FOMCHSD*, the lower threshold value is equal to 0.1 and the upper threshold value is 0.9. *FOMCHSD* is a static algorithm.

In case B = *MadMCHP* the value of the lower threshold is also equal to 0.1 and the value of the upper threshold is calculated using equation in line 12. Our proposed algorithm is triggered when the length of the history data stored in the log file is more than 10.

The current host load state is determined by comparing the value of the current utilization with the lower and the upper threshold. The future load state is predicted using our Markov prediction model. If the future predicted load state is overloaded, then the *migration_decision_overloaded* is assigned a true value and the host is considered for migration. For the underload host detection, if the current state and the future state is underloaded then the *migration_decision_underloaded* is assigned a true value and the host is considered for energy saving or to receive migrated VMs.

2) VM PLACEMENT

VM placement algorithm is the last phase that comes after the detection of the overloaded/underloaded hosts and after the suitable VMs are selected to be migrated. During this phase, suitable hosts are to be found to migrate all the selected VMs, which fits the requirements of these VMs. In the literature, a single built in VM placement exists in CloudSim [4], [5] called Power Aware Best Fit Decreasing (*PABFD*), where all the VMs are sorted based on their current CPU utilization in a descending order. Each VM is allocated to a host with the least increase of the power consumption caused by the allocation. We have modified the existing VM placement algorithm by adding the Markov prediction model into the *PABFD*. In our Markov Power Aware Best Fit Decreasing (*MPABFD*) algorithm, the future host load state is predicted based on the historical data collected and stored in the log file.

Algorithm 4 describes our MPABFD algorithm that results in a host to receive the selected VM. The resource availability is first checked for all the active hosts, bearing in mind that all the candidate hosts are not overloaded after the allocation. The candidate temporary host list is stored in array *TempHostlist*1[]. Next the future state for all the candidate hosts stored in *TempHostlist*1[] are checked. If the future state is overloaded, then the host is excluded from the array. A new temporary array, *TempHostlist*2[], is generated, which is a subset of *TempHostlist*1[]. Finally, power constraint is considered where a host with the minimum power has higher priority to be selected.

3) ILLUSTRATIVE SCENARIO

Consider 3 heterogeneous hosts $h =< h1, h2, h3 >$ and 7 VMs $V =< v1, v2, v3, v4, v5, v6, v7 >$ allocated on them.

Algorithm 4 Markov Power Aware Best Fit Decreasing (MPABFD) Algorithm

```
1   Input: hostlist, selected_vm.
2   Output: a host to receive the selected VM
3   minPower ← MAX
4   allocated_host ← None
5   foreach host in hostlist do
6     If (host has enough resources for the selected_vm
        && hostisactive = true && hoststateafterallo-
        cation () ! = O) then
7       TempHostlist1[] ← add.host
8   foreach host in TempHostlist1[] do
9     future state
    ← host.future_Markov_utilization_state(state)
10    If (future state == U or N) then
11      TempHostlist2[] ← add.host
12  foreach host in TempHostlist2 do
13    power ← estimatePower(host, selected_vm)
14    If (power < minPower) then
15      minPower ← power
16      allocated_host ← host
17  return allocated_host
```

The loads of VMs are allocated to each host as following, $h1 =< v1 = 0.4, v2 = 0.2, v3 = 0.3 >$, $h2 =< v4 = 0.1 >$, $h3 =< v6 = 0.2, v7 = 0.2 >$. The upper threshold is assumed to be a dynamic threshold, $UT\ (h1, h2, h3) =< 0.8, 0.7, 0.7 >$. In addition to the VM loads, each host has extra loads equal to $EL\ (h1, h2, h3) =< 0.03, 0.04, 0.05 >$.

Host 1 detection agent determines an overload situation has occurred according to:

$$\begin{aligned} h1_{load} &= v1_{load} + v2_{load} + v3_{load} + h1_{El} \\ &= 0.4 + 0.2 + 0.3 + 0.03 = 0.93. \\ h1_{load} &\geq h1_{UT} = 0.93 \geq 0.8 \end{aligned}$$

The aim is to migrate a VM in order to avoid SLA violation. To check the host load future state before migrating some VMs, Markov prediction model agent will calculate the future state using the historical data in h1 given by Historical (h1) $=< u, u, u, o, u, n, n, n, n, u, u, u, u, u, n, o, n, o, o, o >$. The future host load state is calculated as:

$$P(w_n = O | w_{n-1} = O) = P_{OO} = \frac{P(w_n = O, w_{n-1} = O)}{P(O)}$$

$$= \frac{P(w_n = O, w_{n-1} = O)}{P(w_n = O, w_{n-1} = U) + P(w_n = O, w_{n-1} = N) + P(w_n = O, w_{n-1} = O)}$$

$$P_{OO} = \frac{2}{1 + 1 + 2} = 0.5$$

Similarly, $P_{OU} = 0.25$ and $P_{ON} = 0.25$. Note that the host will stay in the overload situation, therefore some VMs should be migrated. Let the selection agent select $v2$ to be migrated. To find the destination host for allocating $v2$, MPABFD starts to investigate the first condition, to find the candidate hosts with the capacity requirement still under the

threshold after allocating $v2$ as:

$$h2_{Newload} = h2_{load} + v2_{load} = 0.14 + 0.2 = 0.34$$
$$h3_{Newload} = h3_{load} + v2_{load} = 0.45 + 0.2 = 0.65$$

As noted, both new loads are less than their upper thresholds. The second condition is now investigated on both the candidate hosts to predict the future state using their historical data. Considering Historical (h2) $=< u, o, u, o, u.n.n.n.n, u, o, u, u, o, n, o, o, o, o, u >$, we calculate $P_{UU} = 0.1667$, $P_{UN} = 0.1667$ *and* $P_{UO} = 0.6667$. Host 2 will move to overloaded state. Similarly considering Historical (h3) $=< u, u, n, o, n, n, u, n, n, n, o, o, n, n, n, u, n, n, o, n >$, we calculate $P_{NU} = 0.1818$, $P_{NN} = 0.5454$ and $P_{NO} = 0.2727$. Host 3 will stay in the normal state. It is therefore recommended VM $v2$ to move to host 3 in order to reduce the number of VM migrations and to avoid the SLA violation in the future.

V. EXPERIMENTAL SETUP

In this section, we describe the simulation setup of our proposed approach. We explain the two types of workloads, PlanetLab called a real workload, and random workload. Finally, the evaluation metrics will be described.

A. SIMULATION SETUP

It is difficult to do experiments in a very noticeably dynamic environment like cloud because using real test delimits the experiments to the scale of the infrastructure and makes reproducing the results an extremely difficult undertaking [24]. In addition, measuring performance in real cloud environment is very sophisticated and time-consuming [25]. For these reasons, the CloudSim simulation tool has been chosen to test our approaches before deploying them in real cloud. Other simulators like GangSim, SimGrid, GridSim [26]–[28] do not provide suitable environment that can be directly used for modeling cloud computing environment. They are unable to isolate the multilayer service abstractions i.e. SaaS, PaaS and IaaS required by Cloud. On the other hand, The CloudSim tool supports modeling and simulation of data centers on a single physical computing node that contains implemented algorithms in order to compare them with the proposed approach.

To evaluate the efficiency of our algorithms with the existing algorithm, we have used the same experiment setup as used in [2] with some different workload. A data center has been simulated having J heterogeneous physical hosts and V virtual machines. The value of J and V depends on the type of workload which is specified in Table 1 [29]. In each workload, half of hosts are HP ProLiant ML110 G4 servers 1,860 MIPS each core, and the other half consists of HP ProLiant ML110 G5 servers with 2,660 MIPS each core. Depending on the CPU and memory capacity four types of single-core VMs are used: High-CPU Medium Instance: 2500 MIPS, 0.85 GB; Extra Large Instance: 2000 MIPS, 3.75 GB; Small Instance: 1000 MIPS, 1.7 GB and Micro Instance: 500 MIPS, 0.633 GB. The characteristics of these VM types are similar to Amazon EC2 instance types.

TABLE 1. Characteristics of the workload data (CPU utilization).

Workload Type	Date	Host	VMs	Mean (%)	SD (%)
Real (PlanetLab)	03/03/2011	800	1052	12.31	17.09
	22/03/2011	800	1516	9.26	12.78
	03/04/2011	800	1463	12.39	16.55
	20/04/2011	800	1033	10.43	15.21
Random	----------	50	50	-------	-------

B. WORKLOAD DATA

To make the simulation based evaluation applicable, we evaluate the Markov Prediction Model approach on random workload and real-world publicly available workloads:

- Real Workload (PlanetLab data) [29]: This is provided as a part of the CoMon project; it is a monitoring infrastructure for PlanetLab. In this project, the CPU utilization data is obtained every five minutes from more than a thousand VMs from servers located at more than 500 places around the world. Data is stored in ten different files. We chose two different days from the workload traces gathered during March 2011 and one day from April 2011 of the project. Through the simulations, each VM is randomly assigned a workload trace from one of the VMs from the corresponding day. Table 1 shows the characteristics of each workload.
- Random Workload: Requests for provisioning of 50 heterogeneous VMs that fill the full capacity of the simulated data center are submitted by the users. Each VM runs an application with the variable workload, which is modeled to generate the utilization of CPU according to a uniformly distributed random variable. Each application has a length that determines the number of instructions with MI. The application runs for 150,000 MI that is equal to 10 minutes of the execution on 250 MIPS CPU with 100% utilization.

C. PERFORMANCE METRICS

To compare the performance of our proposed algorithms with the existing algorithms we have chosen eight metrics which are previously defined: SLA violation, percentage of SLA violation time per active host and SLA%, performance degradation that occurs due to migration of VM from one host to another while balancing load or switching off underutilized servers, average SLA violation which describes how many times allocated resources are less than required resources, total number of VM migration occurred either for hotspot mitigation or for VM consolidation, total energy consumption by the physical resources for executing variable workloads, and finally number of hosts that are switching off.

- SLA Violation: In a cloud environment, a service level agreement (SLA) is agreed between the service provider

and the user to ensure the required level of service. SLA contains various details of service level that will be provided to a user, such as, minimum capacities of CPU, RAM, storage, and bandwidth. In case of SLA violation, a party that is responsible for its breach has to pay a fine to the other party. The CPU usage by a VM arbitrarily varies over time. The host is oversubscribed, i.e. if all the VMs request their maximum allowed CPU performance, and the total CPU demand exceeds the capacity of the CPU. It is defined that when the request for the CPU performance exceeds the available capacity, a violation of the SLA established between the resource provider and the customer occurs. For our studies, SLA violation is calculated as shown in (7) [5]:

$$SLA\ Violations\ (SLAV) = SLATAH * PDM \tag{7}$$

where SLAV denotes SLA violation, SLATAH represents SLA violation Time per Active Host, and PDM stand for Performance Degradation due to Migrations. Following equations can be used to calculate SLATAH and PDM.

- SLA violation time per active host (SLATAH): is the observation that if a host serving applications is experiencing the 100% utilization, the performance of the applications is bounded by the host's capacity; therefore, VMs are not being provided with the required performance level. In other word, it means SLA violations due to overutilization [5].

$$SLATAH = \frac{1}{J}\sum_{j=1}^{J}\frac{T_{sj}}{T_{aj}} \tag{8}$$

where J is number of hosts, T_{sj} is the total time that utilization of host j reaches 100 %, and T_{aj} is the lifetime (total time that host is active) of host j. When host utilization reaches 100 %, the applications performance is bounded by the host.

- Performance degradation due to migration (PDM): Live migration is the process of moving VMs from one host to another one (without suspension), it has a negative impact on user applications performance. Dumitrescu and Foster [26] show that this impact depends on application behavior, and the performance degradation can be estimated as 10% of CPU utilization. In other word, it means the SLA violations is due to migration.

$$PDM = \frac{1}{V}\sum_{v=1}^{V}\frac{Cd_v}{Cr_v} \tag{9}$$

where V is the number of VMs, Cd_v estimated as 10% CPU utilization of VM_v in all migrations, Cr_v is total CPU requested by VM_v.

- Average SLA violation: is measured as the mean of the difference between total requested resources (MIPS) by all the VMs. Equation (10) can be used to calculate

$$Average\ SLAV = \frac{\sum_{v=1}^{V}(requestedMIPS) - \sum_{v=1}^{V} allocatedMIPS}{V} \tag{10}$$

where V is the number of VMs.

- Overall SLA violation: is measured as the mean of the difference between total requested resources (MIPS) by all the VMs and total allocated resources (MIPS) [35]. Equation (11) can be used to calculate

$$Overall\ SLAV = \frac{\sum_{v=1}^{V}(requestedMIPS) - \sum_{v=1}^{V} allocatedMIPS}{\sum_{v=1}^{V}(requestedMIPS)} \tag{11}$$

where V is the number of VMs.

- Number of VM migrations: a higher number of VM migrations increases the network load, and results in performance degradation. Equation (12) can be used to calculate the number of migrations during a given time interval [32].

$$Migrations\ (P, t1, t2) = \sum_{j=1}^{J}\int_{t1}^{t2} Mig_j(P) \tag{12}$$

where P represents the current placements of VMs, J is the number of hosts, $Mig_j(P)$ shows the number of migration of host j between time intervals t_1 and t_2 for the placement P.

- Energy Consumption: In order to measure the power consumption of a given server at a time t with placement P [33]. Equation (13) can be used to calculate

$$W_j(P, t) = k * W_{max} + (1 - k) * W_{max} * U_j(P, t) \tag{13}$$

where W_{max} is the power consumption of the server at 100% utilization, k is the static power coefficient that is equal to the amount of power consumption by an idle processor. According to [34], an idle processor consumes 70% of the power consumed when its utilization is 100%. Therefore, in our experiments, k is set to 70%. In this model, $U_j\ (P, t)$ is the current CPU utilization of a host j at time t, which has a linear relationship with the power consumption. Total energy consumption of all the servers between time t_1 and t_2, can be calculated using (14).

$$Energy\ (P, t_1, t_2) = \sum_{j=1}^{J}\int_{t1}^{t2} W_j(P, t) \tag{14}$$

Table 2 illustrates the amount of energy consumption of two types of HP G4 and G5 servers at different load levels. The table shows the energy consumption is reduced efficiently when under-utilized PMs switch to the sleep mode [5].

TABLE 2. The energy consumption at different load levels in Watts.

Server	sleep	0%	10%	20%	30%	40%
HP G4	10	86	89.4	92.6	96	99.5
HP G5	10	93.7	97	101	105	110

Server	50%	60%	70%	80%	90%	100%
HP G4	102	106	108	112	114	117
HP G5	116	121	125	129	133	135

- Number of host shutdowns: consolidation is applied to reduce the number of active physical hosts, the quality of VM consolidation is inversely proportional to H, the mean number of active hosts over n time steps [20]:

$$H = \frac{1}{n}\sum_{i=1}^{n} a_i \tag{15}$$

where a_i is the number of active hosts at the time *step* $i = 1, 2, \ldots, n$. A lower value of H represents a better quality of VM consolidation.

VI. EXPERIMENTAL RESULTS

In this section, we first present the impact of the data length of host status history in our algorithm that makes it perform the best on four different VM selection polices with three different PlanetLab workloads and a random workload. We then show the impact of four different VM selection polices on our algorithms. Then, we discuss our experimental results in comparison to the benchmark algorithms. Finally, the impact of proposed placement algorithm on *MadMCHD* algorithm is investigated.

A. MAXIMUM DATA LENGTH OF HOST STATUS HISTORY OF MARKOV MODEL

One of the important parameter for Markov model is to determine the maximum data length. Consequently, we first investigate a different range of data length in order to find the most suitable length for the four different VM selection polices. To perform this experiment, we study this parameter with three different PlanetLab workloads and a random workload. To choose the best data length, we rely on the aforementioned eight metrics. We have observed through this experiment that each data length parameter affects VM selection policies differently. Therefore, we have chosen the data length parameter that performs well in most of four VM selection policies. We have selected a range for data length from 30 to 180. We have not increased the range over 180 because of time complexity.

We have studied the impact of the mentioned range on the eight metrics. However, for the sake of space, we have shown the impact of data length on SLA violation and number of VM migration metrics as shown in Fig. 3 and Fig. 4 respectively. According to these figures, we have chosen the data length parameter, 120, and this parameter is used for the comparison experiments. For instance, we calculate the average of SLA violation metric when the work load is 20110303 and we have found the following: when the *mc* policy is used and data length is 30, the average of SLA violation metric is 4.2581. Also, the average SLA violation metric for the data length 60, 90, 120, 150, 180 are 2.9934, 2.78045, 1.984, 2.20512, and 2.0664 respectively. Based on these numbers, we can see that the best data length is 120. From Fig. 4, when the work load is 20110322 is used, we have found that the average for number of VM migration is 3296 when the data length is 120, while the average of VM migration is 3250 when the data length is 180. Since this is a slight difference, we consider 120 as the most suitable data length to avoid time complexity when the data length is 180.

B. COMPARISON WITH OTHER BENCHMARKS

We are further interested in comparing our proposed algorithms with the state-of-the-art algorithms. To perform this comparison, we employ the aforementioned eight metrics in order to assess our results. Our comparison process is to study the algorithms' performance in the entire selection process which includes host detection, VM selection, and VM placement.

We compare the proposed algorithm, MadMCHD, with the state-of-the-art five host detection algorithms, namely iqr, mad, lrr, lr, and thr (which is a static threshold set to 0.8) [3]–[5]. We have selected the benchmarks since their implantations are available. Besides, we investigate the impact of four well-known VM selection polices on the proposed model, which are described below. The VM selection algorithms include:

- Maximum Correlation (mc) is inspired that high correlation between tasks and resource usage might lead to server overloading. mc uses the multiple correlation coefficient which corresponds to the squared correlation between the predicted and the actual values of the dependent variable [5].
- Minimum Migration Time (mmt): selects VMs based on the value of the migration time, the less the better. The migration time can be easily computed as the amount of RAM utilized by the VM divided by the additional network bandwidth available for the current allocated host [5].
- Maximum Utilization (mu): Choosing the VMs to migrate from the hotspot based on the largest possible CPU usage can be expected to minimize the number of migrations [2].
- Random Choice (rc): selects the necessary number of VMs by picking them according to a uniformly distributed random variable [23].

For *iqr, lr, lrr, mad, thr*, and *MadMCHD*, we use the well-known placement method which is called PABFD [3]. The above scenarios are illustrated in Fig. 5 and Fig. 6. The main goal of these experiments is to substantiate the threshold adaptability in hypothesis by evaluating the performance of the proposed algorithm across single workload (20110322)

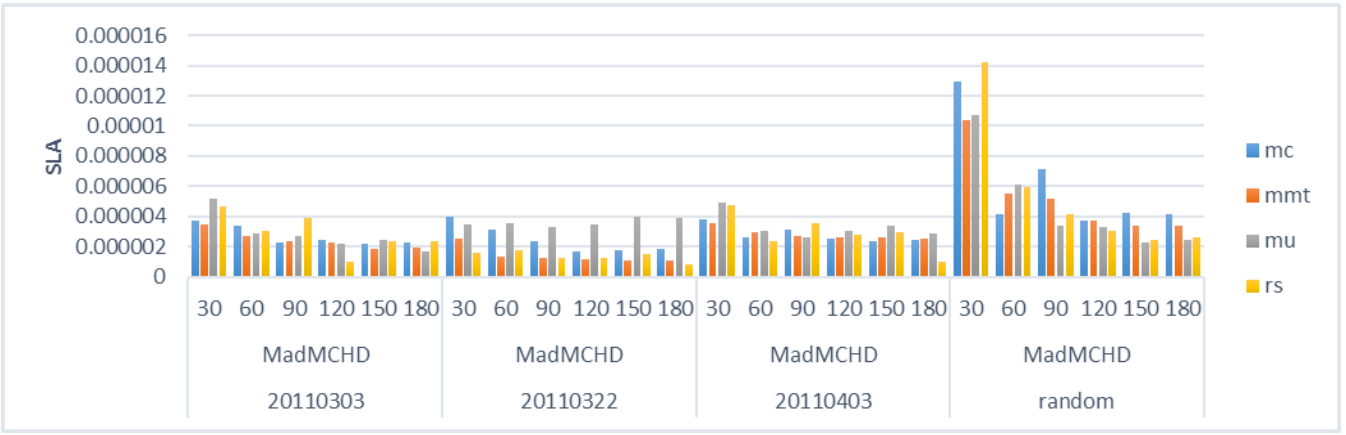


FIGURE 3. The impact of data length on the SLA metric.

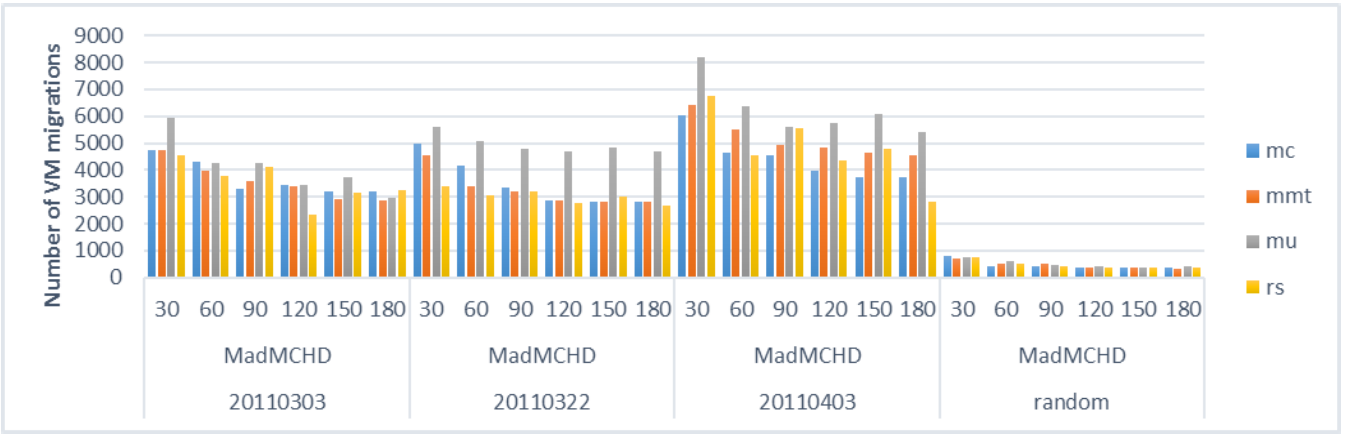


FIGURE 4. The impact of data length on the number of VM migration metric.

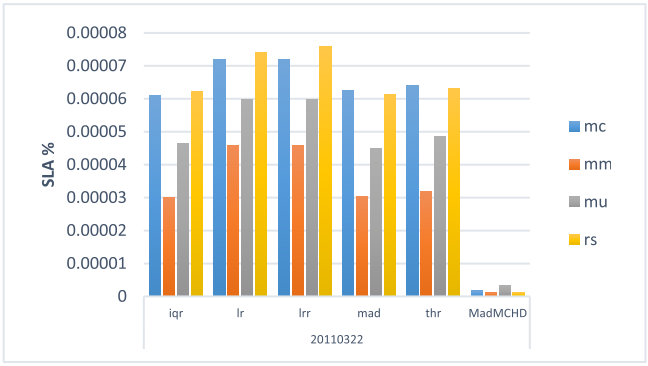
that traces from more than a thousand PlanetLab servers and one random workload. For the readers who are interested in different real workloads, we advise them to refer to our previous work [31]. In the following we compare our results with the minimum value for each selection algorithms when applied to host detection algorithms. For example, when selection algorithm *mc* is applied to all state-of-the-art detection algorithms, we compare our result with the one which gives minimum value (example SLA % in Fig. 5(a)).

From the simulation results depicted in Fig. 5 (a) and Fig. 6 (a), it is completely obvious that the proposed algorithm significantly outperforms the other algorithms in terms of SLA violation for both 20110322 Planetlab real workload and the random workload, since our proposed host load detection algorithm avoids immediate VMs migration. It reduces SLA violation metric by 97.19%, 96.16%, 92.34%, and 90% for the real workload, and by 98.25%, 97.98%, 98.39, and 98.54% for the random workload for VM selection policies *mc, mmt, mu,* and *rs* respectively.

From the simulation results depicted in Fig. 5 (b) and Fig. 6 (b), it is completely obvious that the proposed algorithm significantly outperforms the other algorithms in terms of number of VM migrations for both 20110322 Planetlab real workload and the random workload, since our proposed algorithm avoids immediate VMs migration. The proposed host load detection algorithm reduces number of VM migrations metric by 88.73%, 89.90%, 85.35%, and 89.15% for the real workload, and by 83.97%, 87.74%, 84.61%, and 80.07% for the random workload for VM selection policies *mc, mmt, mu,* and *rs* respectively.

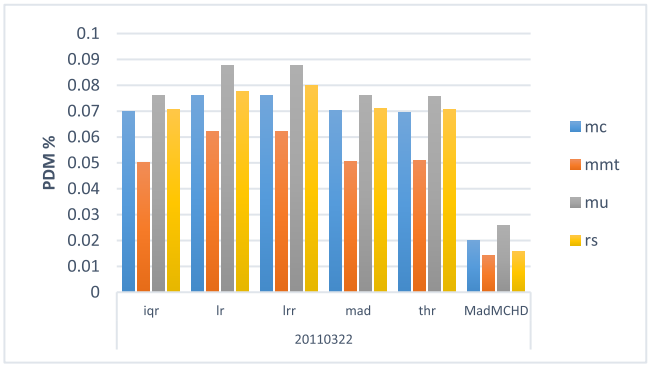
From the simulation results depicted in Fig. 5 (c) and Fig. 6 (c), it is completely obvious that the proposed algorithm significantly outperforms the other algorithms in terms of PDM for both 20110322 Planetlab real workload and the random workload, since our proposed algorithm reduces total CPU requested by *VMs*. The proposed host load detection algorithm reduces PDM migration metric by 71.02%, 72.11%, 58.52%, and 79.05% for the real workload, and by 78.28%, 73.87%, 83.35%, and 83.56% for the random workload for VM selection policies *mc, mmt, mu,* and *rs* respectively.

From the simulation results depicted in Fig. 5 (d) and Fig. 6 (d), it is completely obvious that the proposed algorithm significantly outperforms the other algorithms in terms of SLATAH for both 20110322 Planetlab real workload and the random workload, since our proposed algorithm reduces total time of staying overutilized. The proposed host load detection algorithm reduces SLATAH

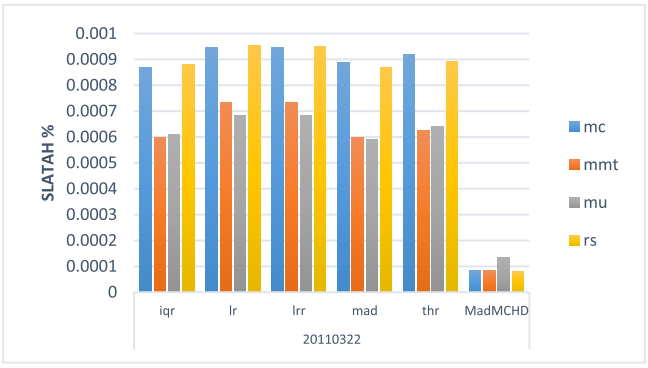


(a)

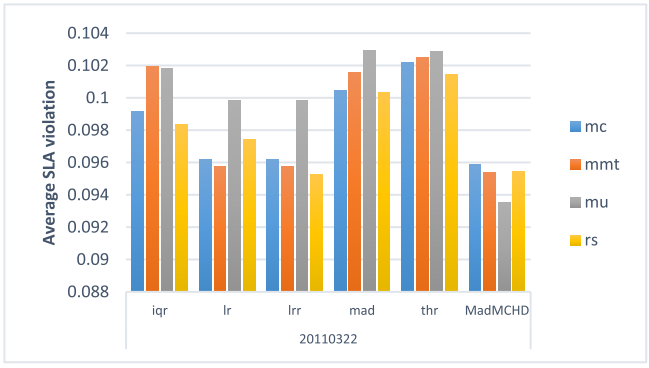
(b)



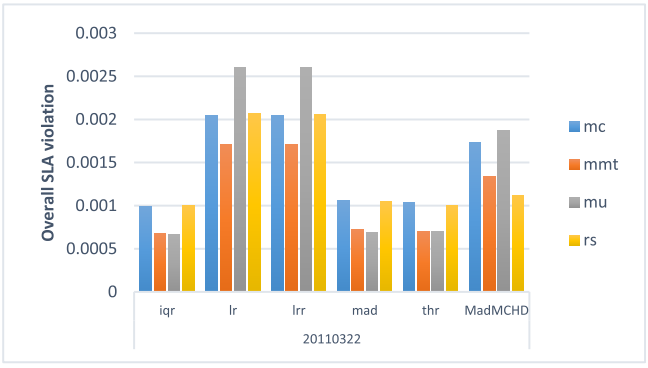
(c)



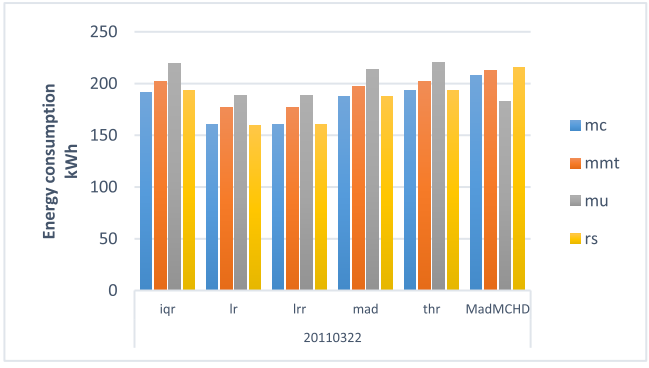
(d)



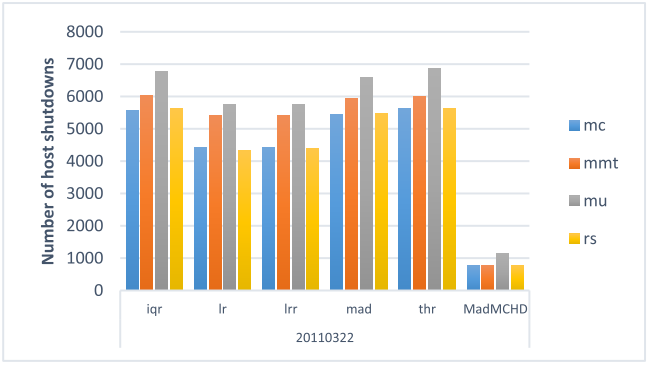
(e)



(f)



(g)



(h)

FIGURE 5. Comparison of Host Detection Algorithms with four VM selection policies for the 11th March 2011 Planetlab workload trace.

FIGURE 6. Comparison of Host Detection Algorithms with four VM selection policies for a random workload trace.

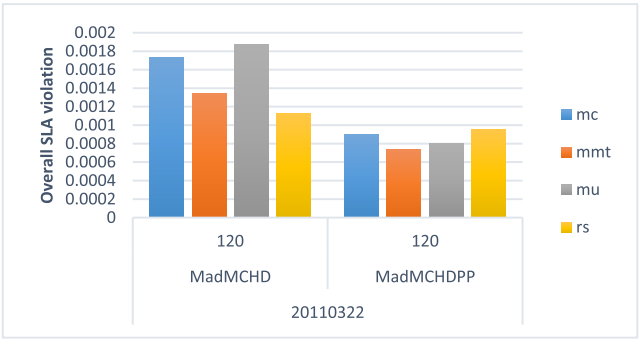


FIGURE 7. Overall SLA violation for a real workload trace.

migrations metric by 90.27%, 86.29%, 78.31%, and 90.83% for the real workload and by 84.58%, 86.30%, 82.19%, and 83.40% for the random workload for VM selection policies *mc, mmt, mu,* and*rs* respectively.

Fig. 5 (e) shows that the proposed algorithm slightly outperforms the other algorithms in terms of the average SLA violation for 20110322 Planetlab real workload. Fig. 6 (e) shows that proposed algorithm is almost similar to the *mad* and *iqr* algorithms in term of the average SLA violation, and the performance of the proposed algorithm is not much better than that of *lr, lrr* and*thr* algorithms for the random workload.

Fig. 5 (f) shows that the proposed algorithm slightly outperforms the *lr* and *lrr* algorithms in term of overall SLA violation for 20110322 Planetlab real workload. It should be noted that the performance of *thr*, mad and *iqr* algorithms still outperform the other algorithms. It is completely obvious from Fig. 6 (f) that the proposed algorithm significantly outperforms the other algorithms in terms of overall SLA violation for the random workload. The proposed host load detection algorithm reduces overall SLA violation metric by 81.05%, 81.22%, 76%, and 80.07% for VM selection policies *mc, mmt, mu,* and *rs* respectively.

Fig. 5 (g) and Fig. 6 (g) show that the proposed algorithm is almost similar to the *thr, mad* and*iqr* algorithms in term of the energy consumption. It should be noted that the performance of the proposed algorithm is not much worse than that of *lr* and*lrr* algorithms.

From the simulation results depicted in Fig. 5 (h) and Fig. 6 (h), it is completely obvious that the proposed algorithm significantly outperforms the other algorithms in terms of number of host shutdowns for both 20110322 Planetlab real workload and the random workload, since our proposed algorithm reduces number of active hosts. The proposed host load detection algorithm reduces number of host shutdowns metric with minimum improvement reach by 82.44%, 85.44%, 80.31%, and 82.52% for the real workload, and by 81.45%, 84.36%, 82.59%, and 81.42% for the random workload for VM selection policies *mc, mmt, mu,* and*rs* respectively.

C. THE IMPACT OF PROPOSED PLACEMENT ALGORITHM ON MADMCHD ALGORITHM

We investigate the impact of our proposed *MPABFD* placement algorithm when it is used in combination with our proposed *MadMCHD* host detection algorithm, termed as *MadMCHDPP* as compared to another combination where the host detection algorithm *MadMCHD* is used with the state-of-the-art placement algorithm *PABFD*, termed as *MadMCHD*. For both combinations, the four selection policies are used, which are *mc, mmt, mu* and*rs*. Fig. 7 shows that the proposed combination *MadMCHDPP* reduces overall SLA violation metric by 47.80%, 45.52%, 47.03% and 14.86% for the real workload for VM selection policies *mc, mmt, mu,* and*rs* respectively.

VII. CONCLUSION AND FUTURE

We present Median Absolute Deviation Markov Chain Host Detection algorithm (*MadMCHD*) based on a dynamic utilization threshold. The proposed algorithm avoids immediate VMs migration in cloud data center by predicting the future host CPU utilization. The current host CPU utilization is calculated and compared with the lower and the upper threshold to determine the current host state. The future host state is predicted using the proposed Markov host prediction model. The proposed algorithm determines when to migrate VMs to achieve server consolidation and load balancing for all the host states.

We present Markov Power Aware Best Fit Decreasing (*MPABFD*) algorithm to enhance VMs placement process. The future candidate host load state is predicted to avoid overloaded state of that host after a short period. We combine the proposed algorithms in the selection process phases in the live migration for better performance, *MadMCHD* as a host detection algorithm, *MPABFD* as a VM placement algorithm, and some of the state of the art algorithms as a VM selection. We investigate the impact of these VM selection polices on the proposed model.

The experimental results show that increasing of the data length of Markov model results in an enhanced performance until a certain value, after which not much improvement in performance is obtained. This value is chosen to not further increase the time complexity of the system.

The experimental results show that *MadMCHD* algorithm can minimize SLA violation rate, number of VM migration, and the other metrics significantly as compared to the most commonly used *thr, mad, iqr, lr and lrr* algorithms. The new combination of the proposed *MadMCHD* and *MPABFD* algorithms shows overall SLA violation is reduced significantly.

The existing VM selection and VM placement approaches focused on minimizing SLA violation rate, minimizing number of VM migration, reducing performance degradation, reducing the number of physical host, VM allocation time and the data center energy consumption. It should be noted that no proactive criteria exist for live WAN migration that minimizes the number of the IP reconfigurations. It is known that if the time needed for IP reconfiguration for all migrated VM users increases, then there will be an increase in the interruption of service, network overhead and performance degradation.

REFERENCES

[1] E. Bauer and R. Adams, *Reliability and Availability of Cloud Computing*. Hoboken, NJ, USA: Wiley, 2012.

[2] A. Beloglazov and R. Buyya, "OpenStack neat: A framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds," *Concurrency Comput., Pract. Exper.*, vol. 27, no. 5, pp. 1310–1333, 2015.

[3] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency Comput., Pract. Exper.*, vol. 24, no. 13, pp. 1397–1420, Sep. 2012.

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

[5] A. Beloglazov, "Energy-efficient management of virtual machines in data centers for cloud computing," Ph.D. dissertation, Dept. Comput. Inf. Syst., Univ. Melbourne, Parkville, VIC, Australia, 2013.

[6] F. Farahnakian, P. Liljeberg, and J. Plosila, "LiRCUP: Linear regression based CPU usage prediction algorithm for live migration of virtual machines in data centers," in *Proc. 39th IEEE EUROMICRO Conf. Softw. Eng. Adv. Appl. (SEAA)*, Sep. 2013, pp. 357–364.

[7] K. Maurya and R. Sinha, "Energy conscious dynamic provisioning of virtual machines using adaptive migration thresholds in cloud data center," *Int. J. Comput. Sci. Mobile Comput.*, vol. 3, no. 2, pp. 74–82, 2013.

[8] S. S. Masoumzadeh and H. Hlavacs, "An intelligent and adaptive threshold-based schema for energy and performance efficient dynamic VM consolidation," in *Proc. Eur. Conf. Energy Efficiency Large Scale Distrib. Syst.*, 2013, pp. 85–97.

[9] L. Salimian, F. S. Esfahani, and M. Nadimi-Shahraki, "An adaptive fuzzy threshold-based approach for energy and performance efficient consolidation of virtual machines," *Computing*, vol. 98, no. 6, pp. 641–660, 2016.

[10] A. Horri, M. S. Mozafari, and G. Dastghaibyfard, "Novel resource allocation algorithms to performance and energy efficiency in cloud computing," *J. Supercomput.*, vol. 69, no. 3, pp. 1445–1461, 2014.

[11] E. Arianyan, H. Taheri, and S. Sharifian, "Novel energy and SLA efficient resource management heuristics for consolidation of virtual machines in cloud data centers," *Comput. Elect. Eng.*, vol. 47, pp. 222–240, Oct. 2015.

[12] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[13] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proc. IEEE/ACM Int. Conf. Green Comput. Commun. (GreenCom), Int. Conf. Cyber, Phys. Soc. Comput. (CPSCom)*, Dec. 2010, pp. 179–188.

[14] E. Feller, L. Rilling, and C. Morin, "Energy-aware ant colony based workload placement in clouds," in *Proc. IEEE/ACM 12th Int. Conf. Grid Comput.*, Sep. 2011, pp. 26–33.

[15] F. Ma, F. Liu, and Z. Liu, "Multi-objective optimization for initial virtual machine placement in cloud data center," *J. Inf. Comput. Sci.*, vol. 9, no. 16, pp. 5029–5038, 2012.

[16] D. Huang, D. Yang, H. Zhang, and L. Wu, "Energy-aware virtual machine placement in data centers," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2012, pp. 3243–3249.

[17] G. Wu, M. Tang, Y.-C. Tian, and W. Li, "Energy-efficient virtual machine placement in data centers by genetic algorithm," in *Neural Information Processing*. Berlin, Germany: Springer, 2012, pp. 315–323.

[18] M. Tang and S. Pan, "A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers," *Neural Process. Lett.*, vol. 41, no. 2, pp. 211–221, 2015.

[19] C. T. Joseph, K. Chandrasekaran, and R. Cyriac, "A novel family genetic approach for virtual machine allocation," *Procedia Comput. Sci.*, vol. 46, pp. 558–565, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915001544

[20] 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.

[21] S. K. Mandal and P. M. Khilar, "Efficient virtual machine placement for on-demand access to infrastructure resourcesin cloud computing," *Int. J. Comput. Appl.*, vol. 68, no. 12, pp. 6–11, 2013.

[22] E. Fosler-Lussier, "Markov models and hidden Markov models: A brief tutorial," Int. Comput. Sci. Inst., Berkeley, CA, USA, Tech. Rep. TR-98-041, 1998.

[23] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput. (CCGrid)*, May 2010, pp. 577–578.

[24] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities," in *Proc. IEEE Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jun. 2009, pp. 1–11.

[25] S. Ray and A. De Sarkar, "Execution analysis of load balancing algorithms in cloud computing environment," *Int. J. Cloud Comput., Services Archit.*, vol. 2, no. 5, pp. 1–13, Oct. 2012.

[26] C. L. Dumitrescu and I. Foster, "GangSim: A simulator for grid scheduling studies," in *Proc. IEEE Int. Symp. Cluster Comput. Grid (CCGrid)*, vol. 2. May 2005, pp. 1151–1158.

[27] A. Legrand, L. Marchal, and H. Casanova, "Scheduling distributed applications: The SimGrid simulation framework," in *Proc. 3rd IEEE/ACM Int. Symp. Cluster Comput. Grid (CCGrid)*, May 2003, pp. 138–145.

[28] R. Buyya and M. Murshed, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing," *Concurrency Comput., Pract. Exper.*, vol. 14, nos. 13–15, pp. 1175–1220, 2002.

[29] K. S. Park and V. S. Pai, "CoMon: A mostly-scalable monitoring system for PlanetLab," *ACM SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 47–65, 2006.

[30] S. B. Melhem, A. Agarwal, N. Goel, and M. Zaman, "Selection process approaches in live migration: A comparative study," in *Proc. 8th IEEE Int. Conf. Inf. Commun. Syst. (ICICS)*, Apr. 2017, pp. 23–28.

[31] S. B. Melhem, A. Agarwal, N. Goel, and M. Zaman, "A Markov-based prediction model for host load detection in live VM migration," in *Proc. 5th IEEE Int. Conf. Future Internet Things Cloud (FiCloud)*, Aug. 2017, pp. 32–38.

[32] S. Mustafa, B. Nazir, A. Hayat, and S. A. Madani, "Resource management in cloud computing: Taxonomy, prospects, and challenges," *Comput. Elect. Eng.*, vol. 47, pp. 186–203, Oct. 2015.

[33] N. Tziritas, C.-Z. Xu, T. Loukopoulos, S. U. Khan, and Z. Yu, "Application-aware workload consolidation to minimize both energy consumption and network load in cloud environments," in *Proc. 42nd IEEE Int. Conf. Parallel Process. (ICPP)*, Oct. 2013, pp. 449–457.

[34] X. Fan, W. D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 13–23, 2007.

[35] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proc. 8th Int. Workshop Middleware Grids, Clouds e-Sci.*, 2010, p. 4.

[36] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, N. T. Hieu, and H. Tenhunen, "Energy-aware VM consolidation in cloud data centers using utilization prediction model," *IEEE Trans. Cloud Comput.*, to be published. [Online]. Available: http://ieeexplore.ieee.org/search/searchresult.jsp?queryText=Energy-aware%20VM%20consolidation%20in%20cloud%20data%20centers%20using%20utilization%20prediction%20model&newsearch=true

SUHIB BANI MELHEM (M'17) received the bachelor's degree in computer engineering from the Jordan University of Science and Technology, Jordan, the M.Eng. degree in electrical and computer engineering from Concordia University, Canada, and the M.Eng. degree in computer engineering from the Jordan University of Science and Technology. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Concordia University. His current research interests include cloud computing, resource management for virtual machine live migration, and decision algorithms.

ANJALI AGARWAL (SM'03) received the B.E. degree in electronics and communication engineering from the Delhi College of Engineering, India, in 1983, the M.Sc. degree in electrical engineering from the University of Calgary, AB, Canada, in 1986, and the Ph.D. degree in electrical engineering from Concordia University, Montreal, QC, Canada, in 1996. She was a Lecturer with IIT Roorkee. She was a protocol design engineer and a software engineer in industry. She is currently a Professor with the Department of Electrical and Computer Engineering, Concordia University. Her current research interests are in the various aspects of cloud networks, heterogeneous networks, and wireless networks, including the security and virtualization of cognitive radio networks.

NISHITH GOEL received the degree in Jodhpur, India, and the M.A.Sc. degree in electrical engineering and the Ph.D. degree in systems design engineering from the University of Waterloo. He began his professional career at Bell-Northern Research, Ottawa, in 1984, and then he moving on to Northern Telecom in 1988. He is currently the CEO of Cistel Technology, an information technology company he founded in 1995, which has operations in Canada, and the USA Veteran Technology Executive and Entrepreneur. He is also a Co-Founder of CHiL Semiconductor, IPine Networks, and Sparq Systems. He is also the Chair of the Queen's Center for Energy and Power Electronics Research, Queen's University. He serves on various corporate boards of directors. His research interests are in information technology, wireless networks, and network security.

MARZIA ZAMAN received the M.Sc. and Ph.D. degrees in electrical and computer engineering from the Memorial University of Newfoundland, Canada, in 1993 and 1996, respectively. She started her career at the Software Engineering Analysis Laboratory, Nortel Networks, Ottawa, ON, Canada, in 1996, where she later joined the OPTera Packet Core Project as a Software Developer. She has many years of industry experience as a Researcher and a Software Designer at Accelight Networks, Excelocity, Sanstream Technology, and Cistel Technology. Since 2009, she has been with the Centre for Energy and Power Electronics Research, Queen's University, Canada, and one of its industry collaborators, Cistel Technology, on multiple power engineering projects. Her research interests include renewable energy, analog and digital control techniques for power converters, wireless communication, embedded systems, machine learning, and software engineering.