A Markov-based Prediction Model for Host Load Detection in Live VM Migration

Suhib Bani Melhem*, Anjali Agarwal*, Nishith Goel[†] and Marzia Zaman[†]
*Department of Electrical and Computer Engineering
Concordia University, Montreal, Canada
Emails: {s_banim, aagarwal} @encs.concordia.ca
[†]Cistech Limited, Ottawa, Canada
Emails: nishith@cistech.ca, Marzia@cistel.com

***Abstract*— Host load detection algorithm determines if a given host is overload or underloaded then the decision can be made to migrate VMs to achieve host/server consolidation and load balancing in cloud data centers while satisfying the QoS constraints. Presently, host load detection is a challenging problem in the cloud data center management specially with high dynamic environment for the host load. In this paper, we propose a novel Markov-based prediction algorithm to forecast the future load state of the host. The experimental results demonstrate that the proposed algorithm has better performance than the other competitive algorithms. The results for different types of PlanetLab real and random workloads show significant reduction of the SLA violation and the number of VM migrations.**

Keywords*— Live Migration; Host underload/overload detection; Virtual machine consolidation; Data center.

I. Introduction

In modern data centers, resource management and allocation during live virtual machine (VM) migration is getting more challenging every day 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, while minimizing the communication burden to avoid congestion, latency, etc. 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 minimizes not only operational costs but also total network load. It should be noted that the key aspect that is directly related to network resource management in the data center is how to minimize network overhead and load balancing-VM migration, which is still an active area of research.

The resource utilization of a data center may change over time due to creation of new VMs and/or hosts, or due to 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.

In cloud data center management, the three most important research problems that have been addressed in the live migration are the host load detection, VM selection, and VM placement.

This paper focuses on the first critical problem that considerably influences the decision making of other problems. The host load detection problem can be divided to overload host detection and underload host detection. 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, the host overload is the process of determining when a given host is overloaded so that 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. However, some solutions based on the last observed utilization for decision making may cause unnecessary migrations, thus increasing the overhead: the energy for VM migration, the performance degradation of the hosted applications, and the extra traffic [1, 2].

In this paper, we propose a host load detection algorithm called Median Absolute Deviation Markov Chain Host Detection algorithm (MADMCHD).

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. Section V presents our experimental setup, performance metrics and 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 overload/underload detection algorithms have been proposed to generate optimal computing resource utilization and energy consumption reduction along data center. Authors in [3] 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 underload 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 [4][13-14] 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 higher 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 made 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.

In literature, many algorithms for the host load detection have been proposed [5-10]. 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, Markov Host Prediction is proposed that uses historical data to build probabilistic model that can predict the future host load more efficiently.

III. Proposed Markov Host Prediction Model

The main goal of the proposed detection algorithm is to improve the SLA violation, and to reduce the number of VM migrations. Our algorithms effectively decide whether it is really necessary to migrate a VM depending on the present as well as the predicted future load based on Markov prediction technique. Markov forecasting model is used to predict future values based on previously observed values [11]. We have applied this technique to historical data to make forecasts.

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 $\{S_1, S_2, \ldots \ldots, S_M\}$. In our proposed Markov model, in the proposed algorithm 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 [4][13-14]. The Markov model is built using three states which are given by $\{S_1 = U,\ S_2 = N,\ S_3 = O\}$. The stochastic variable χ is a discrete random variable taking one of these three values, where Algorithm 1 will be applied periodically by each host manager to find χ for each observation and register it in the host log file. The pseudo-code of host detection for each observation is as follows:

Algorithm 1: Host Detection State

```
1 Input: host CPU utilization of host j ($CPUu(H_j)$), lower
  threshold, and upper threshold.
2 Output: $\chi$ (current host state).
3 If $CPUu(H_j) \leq lower\ threshold$ then
4     $\chi \leftarrow U$
5 else If $lower\ threshold < CPUu(H_j) < upper\ threshold$ then
6     $\chi \leftarrow N$
7 else If $CPUu(H_j) \geq upper\ threshold$ then
8     $\chi \leftarrow O$
9 return $\chi$
```

Three parameters are inserted as inputs in this algorithm. The first input is the host CPU utilization which is calculated by dividing the total request MIPS on the total host MIPS. The other parameter is the lower threshold which is assigned to the value of 0.1. The upper threshold is taken from the MAD algorithm which is explained in [14]. Each host has an underloaded (U), overloaded (O) or normal load (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 Equation 1. Moreover, the joint probability of n observations, $P(w_1, w_2, \dots, w_n)$ using the first-order Markov chain can be given by Equation 2. Our Markov detection algorithm starts working after collecting 10 historical observations (n =10).

$$P(\, w_n \,\big|\, w_{n-1}, w_{n-2}, \dots, w_1) \;\approx\; P(\, w_n \,\big|\, w_{n-1}\,) \tag{1}$$

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

The conditional probabilities $p\,(w_n = S_j \,\big|\, w_{n-1} = S_i)$ 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

$$\begin{aligned} p(w_n = S_j \,\big|\, w_{n-1} = S_i) &= p(w_{n+T} \\ &= S_j \,\big|\, w_{n-1+T} = S_i) \end{aligned} \tag{3}$$

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 $\{S_1, S_2, \dots \dots, S_M\}$.

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1M} \\ p_{21} & \mathrm{p}_{22} & \cdots & p_{23} \\ \vdots & \vdots & & \vdots \\ p_{M1} & p_{M2} & & p_{MM} \end{bmatrix} = \begin{array}{c|ccc} & U & N & O \\ \hline U & p_{UU} & p_{UN} & p_{UO} \\ N & p_{NU} & p_{NN} & p_{NO} \\ O & p_{OU} & p_{ON} & p_{OO} \end{array} \tag{4}$$

The state and transition probabilities of a given Markov chain can be shown using graph. Figure 1 shows our host detection Markov model with three discrete states $\{O, \mathrm{U}, \mathrm{N}\}$. Every periodic time it would transit to a (possibly) new state based on the probabilities in Equation 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.

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.

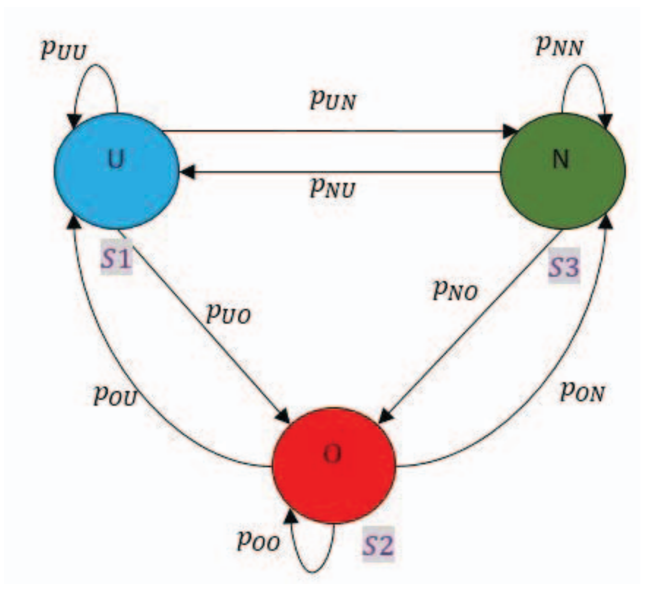

Figure 1: States and Transition probabilities of the Host detection Markov Model.

IV. Proposed System

The target system is an IaaS environment, represented by a large-scale data center. The data center consists of a maximum of 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 terms of Millions Instructions Per Second (MIPS), the amount of RAM and network bandwidth. The target system model is depicted in Figure 2 which is a modified version of the model described in [12]. This model includes two parts: a data center manager and the placement agent which are similar to global manager in [12], and the Markov model prediction agent and the host detection agent, which is a modified version of a local manager introduced in [12].

As shown in Figure 2, our modified version of the local manager is composed of different components as described below:

- Host manager is a component that resides on every host for keeping continuous observation on CPU utilization of the node.
- Host detection agent: this component is responsible on detecting the current load state of the host, which can be underloaded, normal loaded or overloaded.
- VM selection agent: this component is responsible on finding the VM that has to be migrated.

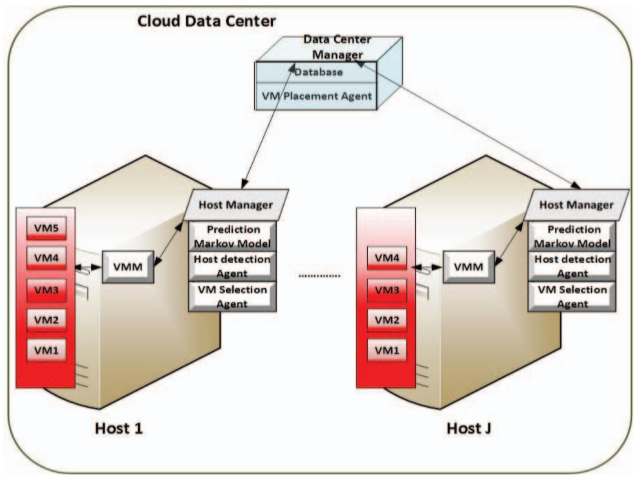

Figure 2: System Architecture.

- Prediction Markov model: this component is responsible on finding the future load state of the host.
- VMM has responsibility 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 physical machines.
- Database is a data structure that contains all the information about the hosts and the utilization of each host.
- VM placement agent: this component is responsible on performing the migration from overload/underload hosts into the candidate hosts.
- Data center manager: is an intermediary component to interact with the host managers and other agents.

The host manager interacts with the prediction Markov model. Our proposed algorithms suggest that the load state host detection 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, VM selection and VM placement processes. It also interacts with the data center manager in order to initiate the VM placement.

Algorithm 2 describes the host underloaded/overloaded detection mechanism. CPU utilization upper and lower thresholds can be assigned either statistically 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 [4].

Algorithm 2: Overloaded/Underloaded 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     $migration\_decision\_underloaded \leftarrow false$
4     $migration\_decision\_overloaded \leftarrow false$
5     $\boldsymbol{while}\ hostactive = true\ \boldsymbol{do}$
6      $\boldsymbol{if}\ logfile.Length >= 10\ \boldsymbol{then}$
7     //calculate current CPU utilization of host h
8     $utilization \leftarrow total\ Req.Mips/\ Total\ host\ Mips$

9     $\boldsymbol{Switch}(\boldsymbol{B})$
10     $\boldsymbol{Case}\ FOMCHSD$: $\boldsymbol{break}$;
11     $\boldsymbol{Case}\ MADMCHD$:
12         $upper\ threshold \leftarrow 1 - s * MAD$

13    //find current utilization using Algorithm 1
14    $current\ state \leftarrow$
      $host\_detection\_state\ (utilization, lower\ threshold,$
                                      $upper\ threshold)$

15    //find future utilization using Markov prediction
      technique
16     $future\ state \leftarrow$
      $future\_Markov\_utilization\_state(current\ state)$

17    $\boldsymbol{If}\ current\ state = O\ and\ future\ state = O\ \boldsymbol{then}$
18    $migration\_decision\_overloaded \leftarrow True$
19    $\boldsymbol{else\ If}\ current\ state = N\ and\ future\ state = O\ \boldsymbol{then}$
20    $migration\_decision\_overloaded \leftarrow True$
21    $\boldsymbol{else\ \ If}\ current\ state = U\ and\ future\ state = O\ \boldsymbol{then}$
22    $migration\_decision\_overloaded \leftarrow True$
23    $\boldsymbol{else\ If}\ current\ state = U\ and\ future\ state = U\ \boldsymbol{then}$
24    $migration\_decision\_underloaded \leftarrow True$

25    return $migration\_decision\_underloaded,$
          $migration\ \_decision\_overloaded$
```

In the case of B = FOMCHSD, the lower threshold value will be equal to 0.1 and the upper threshold value is 0.9. FOMCHSD is a static algorithm. In the case of 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 13. Our proposed algorithm is triggered when the length of the history data stored in the log file is more than 10.

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. After that, the current host load state is determined by comparing the value of the current utilization with the lower and the upper threshold. The output state parameter is then determined. In order to confirm the host load state, the future load state has to be 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 underloaded 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.

We compare our proposed method with five host detection algorithms in [4][13-14] which are implemented in CloudSim [16]. We used the same VM selection algorithms as in [12] and VM placement method as in [13], in these algorithms. The VM selection policy selects a VM based on the value of the migration time, the less the better, this policy is named Minimum Migration Time (MMT). And 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. The VM allocation algorithm selects the destination host to receive the migrated VM, which cause the least increase in the power consumption. The algorithm relies on the traditional greedy algorithm to optimize the allocation of VMs.

V. Experimental Setup

In this section, first we describe the simulation setup for our proposed approach evaluation. Then, we explain the two types of workloads, PlanetLab which will be called a real workload and random workload.

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 [17]. In addition, measuring performance in real cloud environment is very sophisticated and time-consuming [18]. For these reasons, the CloudSim simulation tool has been chosen to test our approaches before deploying them in real cloud. 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 [3] with some different workload. A data center has been simulated having J heterogeneous physical hosts and N virtual machines. The value of J and N depends on the type of workload which is specified in Table 1. 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 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 [14].

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) [15]**: 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.

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	-------	-------

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

VI. Experimental Results

We selected three performance metrics to compare the proposed algorithms with the existing algorithms, which are SLA violation, total energy consumption and the total number of VM migrations [4]. We compare with algorithms presented in [4][13-14] including IQR, MAD, LRR, LR, and THR (which is a static threshold set to 0.8). The main goal of these experiments is to substantiate the threshold adaptability in hypothesis by evaluating the performance of the proposed algorithm across four workloads. The four workloads include three real workloads (20110303, 20110322 and 20110403) that traces from more than a thousand PlanetLab servers and one random workload.

From the simulation results depicted in Figure 3 and Figure 4, it is completely obvious that the proposed algorithm significantly outperforms the other algorithms in terms of SLA violation and number of VM migrations for all the real workloads. Figure 3 and Figure 4 show that the proposed host load detection algorithm reduces SLA violation metric up to 89.30%, 91.68%, and 87.72% for three real workloads, respectively. It should also be noted that it reduces the number of VM migrations up to 81.92%, 84.14% and 81.83% for the three real workloads, respectively. It can be seen from

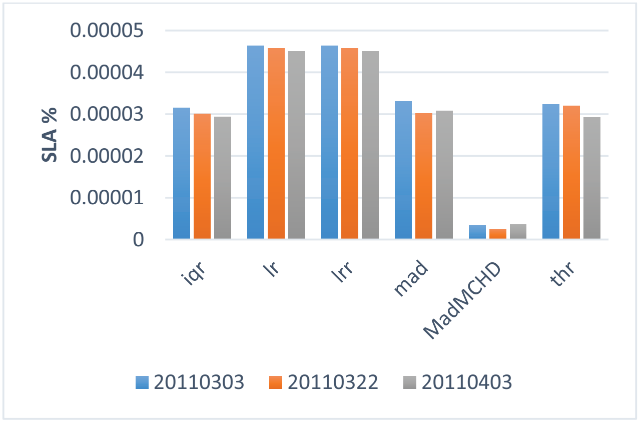

Figure 3: SLA violation for real workloads trace

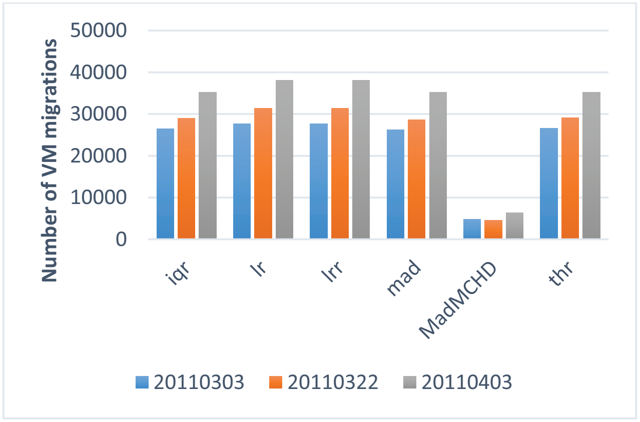

Figure 4: Number of VM migrations for real workloads trace

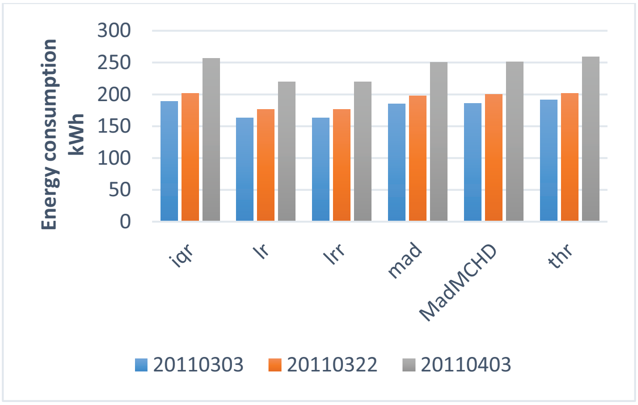

Figure 5: Energy consumption for real workloads trace

Figure 5 that proposed algorithm is similar to the THR 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.

Figure 6 and Figure 7 show that the proposed algorithm significantly outperforms the other algorithms in terms of SLA violation and number of VM migrations for the random workload. It can be seen that the proposed algorithm reduces

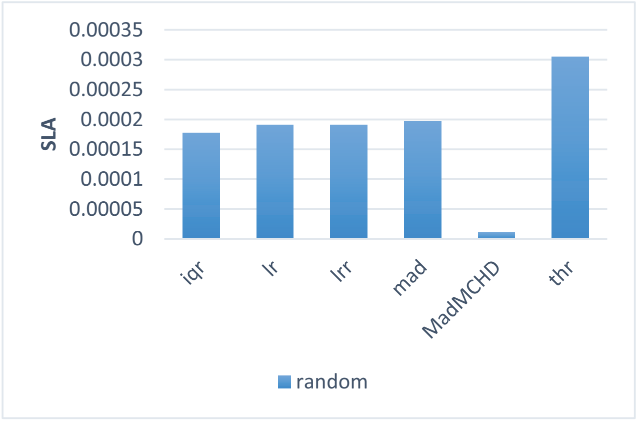

Figure 6: SLA violation for a random workload trace

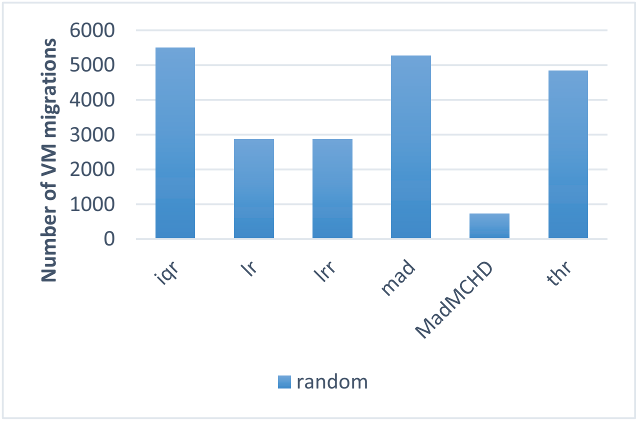

Figure 7: Number of VM migrations for a random workload trace

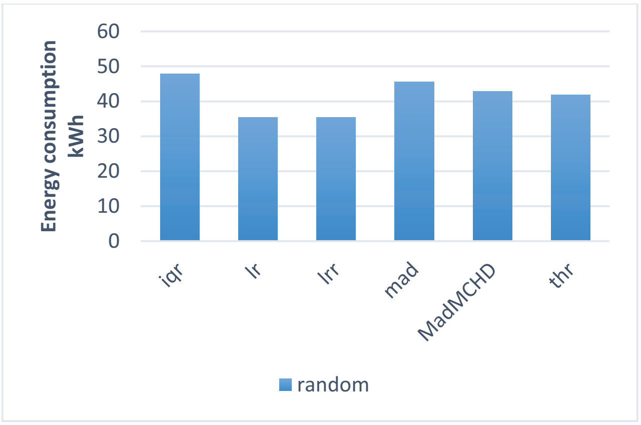

Figure 8: Energy consumption for a random workload trace

SLA violation metric up to 94.15%, and it also reduces the number of migrations up to 74.55%. Figure 8 shows that the proposed algorithm is better than the IQR and MAD algorithms, and provides slightly similar performance to that of LR and LRR algorithms in terms of energy consumption for the random workload.

It should be noted that the improvements in both the SLA violation rate and number of VM migrations have direct impact on the performance degradation, and these improvements have impact on reducing network load by avoiding many VMs migration.

VII. Conclusion

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, then the future host state will be 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 state. The experimental results show that MADMCHD algorithm can minimize SLA violation rate and number of VM migrations significantly compared to the most commonly used THR, MAD, IQR, LR and LRR algorithms.

Acknowledgement

The authors would like to acknowledge the financial support provided by Natural Sciences and Engineering Research Council (NSERC) in collaboration with Cistech Limited, Canada.

References

[1] T. C. Ferreto, M. A. S. Netto, R. N. Calheiros, and C. A. F. De Rose, “Server consolidation with migration control for virtualized data centers,” *Future Generation Computer System*, vol. 27, pp. 1027–1034, 2011.

[2] I. Takouna, E. Alzaghoul, and C. Meinel, “Robust virtual machine consolidation for efficient energy and performance in virtualized data centers,” in The IEEE International Conference on Green Computing and Communications, September 2014.

[3] A. Beloglazov and R. Buyya. “OpenStack neat: A framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds,” Concurrency and Computation: Practice and Experience, vol. 27, no. 5, pp. 1310-1333. 2015.

[4] 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 and Computation: Practice and Experience, vol. 24, no. 13, pp.1397-1420. 2012.

[5] 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 Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference, pp. 357-364. IEEE, 2013.

[6] K. Maurya and R. Sinha. “Energy conscious dynamic provisioning of virtual machines using adaptive migration thresholds in cloud data center,” Int.J.Comput.Sci.Mob.Comput, vol.3, no. 2, pp.74-82. 2013.

[7] S. S. Masoumzadeh and H. Hlavacs. “An intelligent and adaptive threshold-based schema for energy and performance efficient dynamic VM consolidation,” In Energy Efficiency in Large Scale Distributed Systems, pp. 85-97. Springer, 2013.

[8] 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, pp. 1-20. Springer, 2015.

[9] A. Horri, M. S. Mozafari and G. Dastghaibyfard. “Novel resource allocation algorithms to performance and energy efficiency in cloud computing,” The Journal of Supercomputing, vol. 69, no. 3, pp. 1445–1461. Springer, 2014.

[10] E. Arianyan, H. Taheri and S. Sharifian. “Novel energy and SLA efficient resource management heuristics for consolidation of virtual machines in cloud data centers,” Computers & Electrical Engineering, vol. 47pp. 222-240. Elsevier, 2015.

[11] E. Fosler-Lussier. “Markov Models and Hidden Markov Models: A Brief Tutorial,” Technical Report (TR-98-041), International Computer Science Institute, Berkeley, California. 1998.

[12] Beloglazov and R. Buyya. “Energy efficient allocation of virtual machines in cloud data centers,” In Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference, pp. 577-578. 2010.

[13] A. Beloglazov, J. Abawajy and R. Buyya. “Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing,” Future Generation Computer Systems, vol. 28, no. 5 (2012), Elsevier Science, pp. 755-768, 2012.

[14] A. Beloglazov. “Energy-efficient management of virtual machines in data centers for cloud computing,” Ph.D. thesis, The University of Melbourne, 2013.

[15] K.S. Park, V.S. Pai, “CoMon: a mostly-scalable monitoring system for PlanetLab”, ACM SIGOPS Operating Systems Review, pp.65-47, 2006.

[16] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, R. Buyya, “CloudSim: a toolkit for modeling and simulation of Cloud computing environments and evaluation of resource provisioning algorithms”. Journal of Software: Practice and Experience, vol. 41, pp.23–50, 2011.

[17] R. Buyya, R. Ranjan and R. N. Calheiros. "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities," In High Performance Computing & Simulation, 2009. HPCS'09. International Conference, pp. 1-11. IEEE, 2009.

[18] S. Ray and A. De Sarkar. “Execution analysis of load balancing algorithms in cloud computing environment,” International Journal on Cloud Computing: Services and Architecture (IJCCSA), vol. 2, no. 5, pp. 1-13. 2012.