Hybrid Algorithm Naive Bayesian Classifier and Random Key Cuckoo Search for Virtual Machine Consolidation Problem

Yasser Moaly[1]
Department of Computer Science
Faculty of FCAI Cairo University, Cairo, Egypt

Basheer A.Youssef[2]
Doctor, Department of Computer Science
Faculty of FCAI Cairo University, Cairo, Egypt

***Abstract*—The trade-off between Energy consumption and SLA violation presents a serious challenge in cloud computing environments. A non-aggressive virtual machine consolidation algorithm is a good approach to reduce the consumed energy as well as SLA violation. A well-known strategy to deal with the virtual machine consolidation problem consists of four steps: host overloading detection, host under-loading detection, virtual machine selection and virtual machine placement. In this paper, the previous strategy is modified by merging the last two steps virtual machine selection and virtual machine placement, to avoid any poor solutions caused by solving both steps separately. In the host overloading/under-loading detection steps, we classified host status into five classes: Over-Utilized, Nearly Over-Utilized, Normal Utilized, Under-Utilized and Switched Off, then an algorithm, based on the Naive Bayesian Classifier, was introduced in order to detect the future host state for minimizing the number of virtual machine migrations; as a result, the energy consumption and performance degradation due to migrations will be minimized. In the virtual machine selection and placement steps, we introduced an algorithm based on the Random Key Cuckoo Search to reduce the energy consumption and enhance the SLA violation. To assess the algorithm, real data traces for 10 days, were used to verify the proposed algorithms. The experimental results proved that the proposed algorithms can significantly reduce the consumed energy as well as the SLA violation in data centers.**

Keywords*—Cloud computing; Naive Bayesian classifier; Random Key Cuckoo Search; Energy-efficiency; SLA-aware; Virtual Machine consolidation

I. Introduction

Pay-as-you-go basis [1] [2] has increased demand in Cloud computing (CC), especially after the important role played by big data and the internet of things (IOT). In addition, Cloud datacenters consume huge energy, which is mostly produced by non-green energy, resulting in a lot of carbon emissions as well as a high service cost. According to the United States Data Center Energy Usage Report in 2016, in 2010, the energy consumed by U.S data centers constituted about 2% of the global energy consumption [3]. Additionally, a six-month study that was conducted by Barroso & Hölzle (2007) showed that 5000 physical machines (PMs) used nearly around 10-50% of their total capacity [4].

A non-aggressive virtual machine (VM) consolidation is a good approach for reducing energy consumption with attention to the SLA violation (SLAV). VM consolidation is the technique of reallocating virtual machines (VMs) on PMs, subject to minimizing the used resources while keeping the other resources in the sleep mode. Nonetheless, one problem arises, i.e. the VM consolidation issue is an NP-Hard problem [5]. To solve the VM consolidation problem Beloglazov & Buyya (2012) proposed an effective strategy, consisting of four steps [6]: host overloading detection (HOD), host under-loading detection (HUD), VM selection (VMS) and VM placement (VMP). This strategy has been used by most researchers to handle the VM consolidation problem. However, this research found that the week point of this strategy is that it solves both VMS and VMP problems separately, which ultimately produces a poor solution. Therefore, in this study, the aforesaid strategy has been modified through merging the steps of VMS and VMP into a single step. The work in [7], [8] has shown that there is a linear relationship between CPU utilization and power consumption. Thus, in the HOD & HUD detection, it is proposed, using the PM State detection based on the Naive Bayesian classifier (PMSDNBC) [9], to detect the future state of the PM, based on the past CPU historical usage.

Within this study, the PM status is classified into five classes: Over-Utilized (OU), Nearly Over-Utilized (NOU), Normally Utilized (NU), Under-Utilized (UU) and Switched off (SO). The proposed prediction model will help in minimizing the VM migration, resulting in minimizing the energy consumption as well as performance degradation due to migrations. For VMS and VMP, a new algorithm called VM selection and placement using Random Key Cuckoo search (VMSPRKCS), was proposed to handle both VMS and VMP steps. Random Key Cuckoo Search (RKCS) [10] is a novel approach using Cuckoo Search (CS) [11], based on random keys encoding schema (RKES) [12]. CloudSim [13], with a real data set presented from PlanetLab [14], has been used to evaluate the proposed algorithm, and the experimental results showed that the proposed algorithm significantly reduces the consumed energy as well as SLAV.

The rest of the paper is organized as follows: the related work is to be discussed in Section II, a power consumption model in Section III, the Random Key Cuckoo Search concept in Section IV, the proposed overload detection as well as VM selection and placement algorithms in Section V, Simulation Setup and Results in Section VI.

II. Related Work

Since VM consolidation problem is regarded as an NP-Hard problem, it remains an interesting area for researchers, who need to work a lot more towards achieving optimization. Some approaches have used bin-packing optimization algorithms to handle the problem. Shi, Furlong and Wang [15] proposed a greedy algorithm based on the First-fit Decreasing (FFD) algorithm in attempts to reduce the energy consumed by datacenters. FFD has been widely used to find a near optimal solution to the vector pin backing problem. However, the proposed algorithm has a high performance, it can easily be stuck in a local optimum.

Keller, Tighe, Lutfiyya and Bauer [8], proposed new six placement policies based on combinations of order strategies for VMs and PMs. For PMs, they used 3 order strategies: (1) Increasing, which orders the partially-utilized and underutilized PMs in an increasing order according to the CPU utilization; (2) Decreasing, which orders the partially-utilized and underutilized PMs in a decreasing order according to the CPU utilization; and (3) Mixed, which orders the underutilized PMs in a decreasing order and the partially-utilized hosts in an increasing order according to the CPU utilization. For VMs, they only used two strategies: (1) the Decreasing strategy to order all VMs in a decreasing order according to the CPU load; and (2) the Increasing strategy to order all VMs in an increasing order according to the CPU load. The DCSim simulator [16], [17] has been used to evaluate the proposed policies, and the results showed that ordering PMs in a decreasing order and VMs in an increasing order according to the CPU utilization performs better in terms of the energy reduction than the other proposed policies.

Some other approaches have used Artificial Intelligence (AI) prediction models to predict the future PM state in attempts to reduce the energy consumption, the number of VM migrations and performance degradation due to migration. Lei Qiao,Bo Liu,Yang Hua,Qing Zhao & Xiong Fu [18], presented an algorithm based on the Genetic Expression Programming (GEP) [19] to consolidate VMs according to historical data. The algorithm focuses only on 2 steps, VM selection and VM placement. The algorithm carries out dynamic VM migrations for overloaded PMs and under loaded PMs, taking into consideration the proposed prediction model. The algorithm is compared to the most common algorithm Power- aware Best-fit Decreasing (PABFD) algorithm [6], and the results showed that the presented algorithm has a significant improvement in Service Level Agreement Time per Active Host (SLATAH) and in minimizing the number of virtual machine consolidations. Lianpeng Li,Jian Dong,Decheng Zuo & Jin Wu [20] built a prediction model based on Robust Simple Linear Regression (RobustSLR) to detect the future CPU utilization of hosts; then, they proposed overload/under load detection and VMP algorithms based on the proposed prediction model. In the VMP, they modified the PABFD algorithm in order to check the future host state using the proposed prediction model; in case, PM future state will be overloaded, and thus no VMs will be migrated to it. The proposed algorithm was assessed by the CloudSim toolkit with a real data presented from PlanetLab, and the results showed that the presented algorithm could reduce the energy consumption by at most 25.43% and SLA violations by at most 99.16%, compared with PABFD and the most common VM state detection & selection policies.

Lianpeng Li,Jian Dong,Decheng Zuo & JIaxi Liu [21] proposed a prediction model based on the Simple Bayesian Classifier to detect the future host state. They used the mean function to convert the CPU utilization history for the last hour, and then used them as the Bayes features and the host states (overloaded or not overloaded) as Bayesian Classifier labels. Then, they calculated the posterior probability using the prior probability and condition probability, and then the classifier predicted that that input vector belonged to the class having the highest posterior probability. The algorithm has been compared with the most common PDFA algorithm, and the results showed that the proposed algorithm performed better in terms of energy consumption and SLAV than did PDFA.

Other approaches have used meta-heuristic algorithms to find a near optimal solution. Dabiah Ahmed Alboaneen, Huaglory Tianfield & Yan Zhang [22], applied the Glowworm Swarm Optimization (GSO) Algorithm for the VMP problem. They used a CloudSim toolkit with real data presented from PlanLab to assess the algorithm. The results showed that GSO can perform better than many common placement policies. Khaoula BRAIKI & Habib YOUSSEF [23], proposed a multi-objective algorithm based on the Particle Swarm Optimization (PSO) algorithm to improve resource utilization while minimizing the energy consumption. Sanaz Tavakoli-Someh & Mohammad Hossein Rezvani [24], proposed a multi-objective NSGA-II meta heuristic algorithm to handle the objectives of: (1) maximizing the resources utilization; and (2) minimizing the number of used PMs. The experimental result showed that the proposed algorithm performed better than such basic approaches as the FFD algorithm and the best-fit decreasing (BFD) schemes.

Another statistical analysis approach, based on historical data, has been proposed by Beloglazov and Buyya [6]. It provides four overload host detection policies: Local Regression (LR), Median Absolute Deviation (MAD), Robust Local Regression (RLR) and Interquartile Range (IQR), in addition to 3 selection policies: Minimum Migration Time (MMT), Random Selection (RS) and Maximum Correlation (MC). The CloudSim simulator with PlanetLab traces has been used to evaluate the proposed overload state detection and VM selection policies. As shown by the results, the LR overload detection policy with MMT selection policy significantly surpasses the other policies in reducing SLAV and the number of VM migrations. That is why we considered LR_MMT 1.2 as the benchmark and compared the proposed approach in this research with it using the same traces provided from PlanetLab.

III. Power Consumption Model

According to different studies, the most energy consumed by servers is mostly consumed by the CPU rather than other resources [7], [25]. Hence, for simplicity, we can represent the PM energy consumption with its CPU utilization. We used a real power consumption data model, illustrated in Table I provided by SPEC power [26] for two servers, HP ProLiant ML110 G4 and HP ProLiant ML110 G5, to calculate the energy consumption.

TABLE. I. SPECPOWER POWER CONSUMPTION OF TWO SERVERS: HP PROLIANT G4 & G5 AT DIFFERENT LOAD LEVELS IN WATTS

CPU	**HP ProLiant G4**	**HP ProLiant G5**
0%	86	93.7
10%	89.4	97
20%	92.6	101
30%	96	105
40%	99.5	110
50%	102	116
60%	106	121
70%	108	125
80%	112	129
90%	114	133
100%	117	135

IV. THE RANDOM KEY CUCKOO SEARCH (RKCS) CONCEPT

The RKCS is a novel approach of Cuckoo Search Meta-heuristic algorithms based on a random key encoding schema. It is very effective in the combinatorial problems, as it will be used to apply Lévy flight on the old solution to generate a new cuckoo solution from it.

A. *Cuckoo Search (CS)*

The CS is a population based meta-heuristic optimization algorithm inspired by the natural behavior of cuckoo birds. It was presented by Xin-She Yang and Suash Deb in 2009. When cuckoo birds want to lay eggs, they do not build their own nests, but they rather use some other birds' nest. In case that other bird discovered that those eggs were not their own, it would either throw them or abandon its nest and build another one elsewhere. Below are the laws CS:

1) Number of hosts is fixed.

2) A nest can host only one egg.

3) Each parasitic egg represents a solution in the search space.

4) The transaction of generating a new solution (Cuckoo one) from old solution is accomplished using Lévy flight.

5) Height quality solutions with good fineness values will be carried forward to the next generation.

6) There is a probability pa [0, 1] to discover the parasitic foreign egg by the host bird.

7) For a cuckoo i, the new parasitic cuckoo (solutions) Xi(t+1) generated by using Lévy flight using Equation #1 where $\alpha > 0$ is the step size, and product $\oplus$ means entry-wise multiplication.

8) Lévy flight is a random step length in a random angle in which the random step length is generated from the Lévy distribution using equation #2.

$$x^{(t+1)} = x^{(t)} + \alpha \oplus \text{ Levy}(\lambda)\alpha \qquad (1)$$

$$\text{Lévy} \sim u = x^{-\lambda}, (1 < \lambda < 3) \qquad (2)$$

B. *Random-Key Encoding Scheme (RKES)*

The RKES is a transformation technique which is used to represent a vector in a continuous search space in a combinatorial form. The conception of RKES is to generate a random weight value from [0, 1] to each item in the array vector, and then order it in an ascending order according to the weight values. Table II illustrates a sample RKES.

TABLE. II. RANDOM-KEY ENCODING SCHEME (RKES)

WEIGHTS	0.8	0.6	0.7	0.3	0.1	0.4
DECODED AS	6	3	4	2	4	7

V. PROPOSED OVERLOAD / UNDERLOAD DETECTION, VM SELECTION AND VM PLACEMENT ALGORITHMS

An effective strategy proposed by Beloglazov and Buyya (2012) [6] to deal with the VM consolidation problem consists of the following four steps:

1) Host Overloading Detection (HOD): to detect the overloaded PMs; some VMs must be migrated from them to eliminate SLAV.

2) Host Under-loading Detection (HUD): to detect PMs with low utilization resources; all VMs must be migrated from them if possible, and switch them into the sleep mode.

3) Virtual Machine Selection (VMS): responsible for selecting some VMs from overloaded PMs in order to migrate them to another suitable PMs.

4) Virtual Machine Placement (VMP): responsible for finding a new suitable PMs for the selected VMs.

In this research, we merged the steps, VMS and VMP into one to avoid any poor solution resulting from solving both separately. In the HOD & HUD steps, a prediction algorithm based on the Naive Bayesian Classifier is proposed to predict the future host state for minimizing the VMs migrations. In the VMS and VMP steps, we proposed an algorithm based on the RKCS to reduce the energy consumption and SLAV. Solution construction, host state prediction model, and the proposed VM selection as well as placement policy are discussed in the following sections.

A. *Solution Construction*

Assume that we have n VMs and m PMs; in the context of VM consolidation problem and RKCS, a single egg solution S_i is represented by: the VMs random Keys ($VMRK_i$), the PMs random keys ($PMRK_i$) and the VM migrations mappings ($VMMM_i$), which are constructed as shown in Tables III, IV and V, respectively.

Each element in $VMRK_i$ will have a VM index (VMI_i) and VM weight value (VMW_i). Each item in $PMRK_i$ will have: PM Index (PMI_i), PM Status weigh value ($PMSW_i$) and PM weight value (PMW_i). Finally, $VMMM_i$ represents the placement mappings; each value in the array represents a PM index to which will the VM will be placed. For instance, if the second element of array is 7, then the second VM will be migrated to PM of index 7.

TABLE. III. VIRTUAL MACHINES RANDOM KEY

VMI_1	VMI_2		VMI_3	VMI_4	…	VMI_n
VMW_1	VMW_2		VMW_3	VMW_4	…	VMW_n

TABLE. IV. PHYSIAL MACHINES RANDOM KEYS

PMI_1	PMI_2	PMI_3	PMI_4	…	PMI_m
$PMSW_1$	$PMSW_2$	$PMSW_3$	$PMSW_4$	…	$PMSW_m$
PMW_1	PMW_2	PMW_3	PMW_4	…	PMW_m

TABLE. V. VIRTUAL MACHINES MIGRATION MAPPINGS

VM_1	VM_2	VM_3	VM_4	…	VM_n
PMI_1	PMI_2	PMI_3	PMI_4	…	PMI_n

B. Physical Machine State Prediction Model using Naive Bayesian Classifier (PMSDNBC)

The study presented by Haikun Liu, Hai Jin, Cheng-Zhong Xu & Xiaofei Liao [27], stated that live migration is not free, and that is why predicting the PM future state will play a significant role in minimizing the number of VM migrations, resulting in minimizing energy consumption and performance degradation due to the migration. In this research, a new algorithm called PM state predection using Naive Bayesian classifier (PMSDNBC) was introduced to detect the future host state. The Naive Bayesian classifier is an AI machine learning model used to classify different objects based on certain features. In the current research, the model is used to predict the PM future state according to the past CPU utilization history. For the classifier features, we constructed n+1 dimensional feature vector using the latest CPU utilization history CPU_t, CPU_{t-1}, CPU_{t-2}, CPU_{t-3}, CPU_{t-4}.. CPU_{t-n+1} in the time t, t-1, t-2, t-3, t-4 … t-n+1 for the last n preceding points. Then, we got input vector F = [CPU_t, CPU_{t-1}, CPU_{t-2}, CPU_{t-3}, CPU_{t-4}... , CPU_{t-n+1}], and basically, as we need to detect the future state of PM, we used the PM states as the Bayes classifier labels. In most researches the PM status is classified into the following three states: (1) Over-Utilized, (2) Under-Utilized and (3) Switched Off. For the Over-Utilized PMs, some VMs must be migrated from them to other non Over-Utilized PMs in order not to elimnate the SLA. For the Under-Utilized PMs, all PMs must be migrated from them, if possible, in order to switch them to the sleep mode. In this reserach we found the below weak points in this classification:

- PMs which will accept migrations from the Over-Utilized PMs and PMs from which all VMs must be migrated, if possible, are both treated as "Under-Utilized".
- In the previous classification, there is no state to describe the PMs which are not Over-Utilized and are not able to accept new migrations, as they are near to be Over-Utilized.

For this reason, in this research we classified the PM statuses into the below five classes:

1) *Over-Utilized (OU):* PMs which are over utilized, so some VMs must be migrated from them in order not to violate SLA.

2) *Nearly Over-Utilized (NOU):* PMs which are near to be over-utilized. These types of PM will not accept any migrations nor migrate VMs from them.

3) *Normal Utilized (NU):* PMs which fall between nearly over-utilized and underutilized thresholds. These types of PMs will accept new migrations but not VMS will be migrated from them.

4) *Under-Utilized (UU):* PMs which are underutilized; all VMs hosted on these PMs should be migrated away to other suitable PMs if possible, then switch these UU PMs to the sleep mode.

5) *Switched Off (SO):* Switched off PMs.

Accordingly, classifier labels will be L = {OU, NOU, NU, UU, SO}. Static threshold values based on the experimental results have been used to identify: OU, NOU, NU, UU and SO thresholds; the values are illustrated in Table VI. Classifier dataset is constructed according to the below rules:

1) *In case the CPU* utilization value equals 0, the value of “SO” label will be set to 1 and the other labels will be set to minimum product effect value (MPEV).

2) *In case the CPU* utilization value is greater than the over-utilized threshold, the value of the “OU” label will be set to 1 and the other labels will be set to MPEV.

3) *Otherwise*, we will calculate the percentage of how far the current state from the upper and lower thresholds using equations #3 & #4 and the reset labels are set to MPEV.

$$\mathrm{VUTi} = \frac{(\mathrm{UPT}-\mathrm{LWT})-(\mathrm{UPT}-\mathrm{CPUi})}{(\mathrm{UPT}-\mathrm{LWT})} \quad (3)$$

$$\mathrm{VLTi} = \frac{(\mathrm{UPT}-\mathrm{LWT})-(\mathrm{CPUi}-\mathrm{LWT})}{(\mathrm{UPT}-\mathrm{LWT})} \quad (4)$$

Where UPT is the upper threshold value, LWT is the lower threshold value, VUTi is the value of state i for the upper label and VLTi is the value of state i for the lower label.

```
Algorithm 1: PMSDNBC
Input: CPU Utilization History
Output: Host State
1.  n = 10; minValue=0.0001;
2.  For i =0 to n − 1 do
3.      F[i] = cpuUtilizationHistory [n − i − 1];
4.      IF F[i] == 0 Then
5.          Set switchedOffValues[i] =1;
6.          Set Other Values to minValue;
7.      Else If F[i] > overUtilizedThreadShold Then
8.          Set overUtilizedValues [i] =1;
9.          Set Other Values to minValue;
10.     Else
11.         Calculate VUTi value equation #1;
12.         Calculate VLTi value equation #2;
13.         Set other labels values to minValue;
14.     End If;
15. Calculate P(x | OU), P(x | NOU), P(x | NU), P(x | UU)
    & P(x | SO) using equation #5.
16. Return state of highest probability
```

TABLE. VI. STATIC THRESHOLDS VALUES

THRESHOLD	RANGE
OVER-UTILIZED	CPU >= 0.8
NEARLY OVER-UTILIZED	0.8 > CPU >=0.78
NORMAL UTILIZED	0.78> CPU >=0.5
UNDER-UTILIZED	0.5> CPU > 0
SWITCHED OFF	CPU = 0

The value 0.00001 has been used as MPE, first, to avoid zero values generated from product operation and second, because it has lower effect in the product operation.

Finally, for each label l in L, the probability of P (x|l) is calculated using equation #5 and according to the Naive Bayesian Classifier, PM state will be classified as the state with the highest probability.

$$p(x|l) = \prod_{i=1}^{n} P(fi|l) \tag{5}$$

$$P(fi|l) = \frac{P(fi \cap l)}{P(l)} \tag{6}$$

C. Virtual Machine Selection and Placement using Random Key Cuckoo Search (VMSPRKCS)

VMS & VMP is the process of selecting some VMs from over-utilized and under-utilized PMs and finding a new placement for them. In this research, an algorithm based on RKCS is proposed to handle the two steps VMS & VMP into one step to avoid producing a poor solution resulting from solving both steps separately. First, we detect the state of all PMs using the proposed algorithm PMSDNBC. Then, we added all VMs of over-utilized and under-utilized PMs into one dimensional vector VMs = [V1, V2, V3 … Vn]. For the over-utilized PMs, we will migrate some VMs from them in order to not to violate SLA; for the under-utilized PMs, all VMs will be migrated from them if possible in order to switch them to the sleep mode. Since RKCS is a population based algorithm, we will have n solutions S = [s_1, s_2, s_3 … s_n], and for each solution s_i, we will have $VMRK_i$ and $PMRK_i$. The idea behind VMSPRKCS is consists of the below steps:

1) Detect the status of all PMs using PMSDNBC algorithm.

2) Add all VMs of over-utilized & under-utilized PMs into one dimensional array SVMs = [VM1,VM2,….VMn].

3) Assigning a weight value for each VM in SVMs, either generated randomly between [0-1] or by Lévy using equation #1 when generating a cuckoo egg from an old one.

4) Assigning a weight value and a status weight value for each PM. The PM weight value is either generated randomly between [0-1] or by Lévy using equation #1 from an old solution. The PM status weight value is generated according to Table VII.

5) Sorting VMs in SVMs array in an ascending order by the weight value

6) Sorting PMs in an ascending order by the status weight value then by the weight value.

7) Each VM will be allocated to the first PM fitting for it, which will not be overloaded after the migration.

8) In case the status of the old PM, from which the VM is migrated, changed from “Over-Utilized” to any new status, then the remaining VMs on the old PM added to an excluded list in order to skip their migration.

9) In case the status of the new PM, to which the VM is migrated, changed to any new status, then new PM status weight value is recalculated based on PMSDNBC algorithm and Table VII, then all PMs are reordered in an ascending order by their status weight value then by the weight value.

TABLE. VII. PM STATUS WEIGHT INDEX MAPPINGS

STATUS	STATUS WEIGHT VALUE
NORMAL UTILIZED	1
UNDER-UTILIZED	2
SWITCHED OFF	3
NEARLY OVER-UTILIZED	4
OVER-UTILIZED	5

Algorithm 2: VMSPRKCS

```
Input: PmList, VmList
Output: VMs migration mappings
1.  For i=0 to PmList.length Do
2.      Let pmStatus = getHostStatus(pmList [i])
3.      If pmStatus==’Over-Utilized’ or pmStatus==’Under-
        Utilized’ then
4.          vms.add(pmList [i].getVmList());
5.      End If
6.  End for
7.  return FindBestAllocation(PmList, vms);
```

Algorithm 3: FindBestAllocation

```
Input: PMs, VMs
Output: VMs migration mapping
1.  n=10;
2.  Generate an initial population of n host nests;
3.  While (iterations < maxIteration) do
4.      Get a cuckoo randomly solution $S_i$ by L´evy flights
5.      ApplyPlacement (pms, vms, $S_i$)
6.      Calculate $F_i$ quality of $S_i$
7.      Choose a nest among n (say, j) randomly
8.      If ($F_i > F_j$ ) then
9.          replace j by the new solution;
10.     end
11.  A fraction (pa) of worse nests is abandoned and new ones
    are built.
12.  Keep the best solutions
13.  Rank the solutions and find the current best
14.  End while
15. Return best;
```

Algorithm 4: ApplyPlacement

```
Input: PMs, VMs, S
1.  Sort S.VMRK in ascending order by Weight Value
2.  Sort S.PMRK in ascending order according by Status
    Weight value then by Weight Value
3.  Let VMRK = S.VMRK
4.  For i=0 to VMRK .length do
5.      Let vm = VMs[VMRK[i].index];
6.      If ExcludedVMs .contains(vm) then
7.          Continue;
8.      End If
9.      Let pm = vm.getHost();
10.     Let pmPreStatus = PMSDNBC(pm);
11.     pm.vmDestroy(vm)
12.     Let pmPostStatus = PMSDNBC(pm);
13.     IF pmPreStatus == "Over Utilized" and
        pmPostStatus!=  "OverUtilized" then
14.         ExecludedVms.addAll(pm.getVmList());
15.     End If
16.     FindFirstFittingPm(PMs, vm, S)
17. End For
End For
```

Algorithm 5: FindFirstFittingPm

```
Input: PMs, VM, S
Output: PM
1.  Let selectedPm = null;
2.  Let PMRK = S. PMRK;
3.  For j=0 to PMRK .length do
4.      Let pm = pms[PMRK[i].index];
5.      If isHostOverUtilizedAfterAllocation(pm,VM) then
6.          continue;
        Else
8.          Let pmPreStatus = PMSDNBC(pm);
9.          pm.vmCreate(vm);
10.         Let pmPostStatus = PMSDNBC (pm);
11.         If pmPreStatus!= pmPostStatus then
12.             Get pm  weight status value using PMSDNBC
                and Table VII;
13.             Sort PMRK in ascending order according by
                Status Weight then by Weight Value;
14.         End If
15.         selectedPm  = PM;
16.         break;
17.     End If
18. End For
19. Return selectedPm  ;
```

VI. Simulation Setup and Results

Performance metrics, Simulation setup & Experimental results are discussed in the following sections.

A. Performance Metrics

The following metrics have been considered to evaluate the proposed algorithm:

1) Energy consumption: it refers to the total energy consumed by all PMs in the datacenter, where the PM energy consumption is calculated according to a real data power consumption model provided by the SPEC power benchmark. Table I illustrates the energy consumption of two different PMs, HP G4 and HP G5, at different CPU load levels.

2) Service Level Agreement Violation (SLAV): As proposed in [6], SLAV can be measured in IaaS for two main factors: (1) SLA violation time per active host (SLATAH) resulting from the CPU utilization is of 100% and (2) the SLA violations resulting from performance degradation due to migration (PDM). The SLAV factors can be calculated using equations #7.

$$\mathrm{SLAV} = \mathrm{SLATAH} * \mathrm{PDM} \quad (7)$$

$$\mathrm{SLATAH} = \frac{1}{\mathrm{m}}\sum_{\mathrm{k}=0}^{\mathrm{m}} \frac{\mathrm{Tok}}{\mathrm{Tak}} \quad (8)$$

Where m is the number of PMs, Tok is the total time in which the PM had an over-utilized status, resulting from the 100% CPU utilization and Tak is the total time of the PM for being in the active state.

$$\mathrm{PDM} = \frac{1}{\mathrm{n}}\sum_{\mathrm{k}=0}^{\mathrm{n}} \frac{\mathrm{Cdk}}{\mathrm{Crk}} \quad (9)$$

Where n is the number of VMs, Cdk is the estimated performance degradation of the VM k due to migrations and Crk is the total requested CPU capacity by the VM j.

1) ESV: as the research target is to balance the trade-off between the SLA violation and energy consumption, it is important to consider this metric. It is simply calculated as the product of SLAV and energy consumption

$$ESV = SLAV * Enegry\_consumption \quad (10)$$

2) Number of VM migrations: it refers to the total number of VM migrations occurred over all the datacenter.

3) Shutdown hosts: it refers to the total number of PMs switched to the switched off mode over all the datacenter.

B. Simulation Setup

To evaluate the proposed algorithm, we used the CloudSim 3.0.3 toolkit simulator. Cloudsim is a well-known common simulator that supports different policies for host overload detection, VM selection and VM placement. It also provides different types of workload as well as several cloud metrics calculation, such as: Energy Consumption, SLAV, number of VM migrations, PDMA, SLATAH and number of host shutdowns. Furthermore, we used real workload traces from a real system (PlanetLab data). PlanetLab is the monitoring part of the CoMon project. It monitored CPU utilizations for more than thousand VMs hosted at more than 500 PMs which were collected during March and April 2011. Each day in the traces has a file for each VM, containing 288 values which represent the VM CPU utilization value [0-100] every 5 minutes during the day. Traces characteristics are represented in Table VIII. A datacenter comprising 800 heterogeneous PMs and more than 1000 VMs was simulated; half of the PMs were HP ProLiant ML110 G4 (Intel Xeon 3040, dual-core 1860 MHz, 4 GB, 1 Gbps) and the rest are HP ProLiant ML110 G5 (Intel Xeon 3075, dual-core 2660, 4 GB, 1 Gbps). For the VMs, four types were used, corresponding to Amazon EC2 [27] illustrated below:

1) Micro instance (613MB, 500 MIPS).
2) Small Instance (1.7 GB, 1000 MIPS).
3) Extra-large Instance (3.75 GB, 2500 MIPS).
4) High-CPU Medium Instance (0.85 GB, 2500 MIPS).

C. Experimental Results

Traces with heterogeneous states for real cloud datacenter presented from PlanetLab illustrated in Table VIII have been used to evaluate the algorithm. A study presented by Beloglazov and Buyya [6], stated that lr_MMT 1.2 performs better than other dynamic VM consolidation algorithms, so we considered it as the benchmark and compared our proposed algorithm with it. The results are illustrated in Fig. 1, 2, 3, 4, 5, 6 and 7. The experimental results show that the proposed algorithm can highly reduce the below metrics:

1) Energy consumption reduced by minimum 17.7%, by maximum 28.6% and with average 24.23%, compared with lr_MMT 1.2.

2) SLAV reduced by minimum 93.52%, by maximum 96.89% and with average 95.35%, compared with lr_MMT 1.2.

3) ESV reduced by minimum 95.5%, by maximum 97.56% and with average 96.5%, compared with lr_MMT 1.2.

4) The number of VM migrations reduced by minimum 88.07%, by maximum 90.85% and with average 89.3%, compared with lr_MMT 1.2.

5) PDM reduced by minimum 77.78%, by maximum 88.89% and with average 86.3%, compared with lr_MMT 1.2.

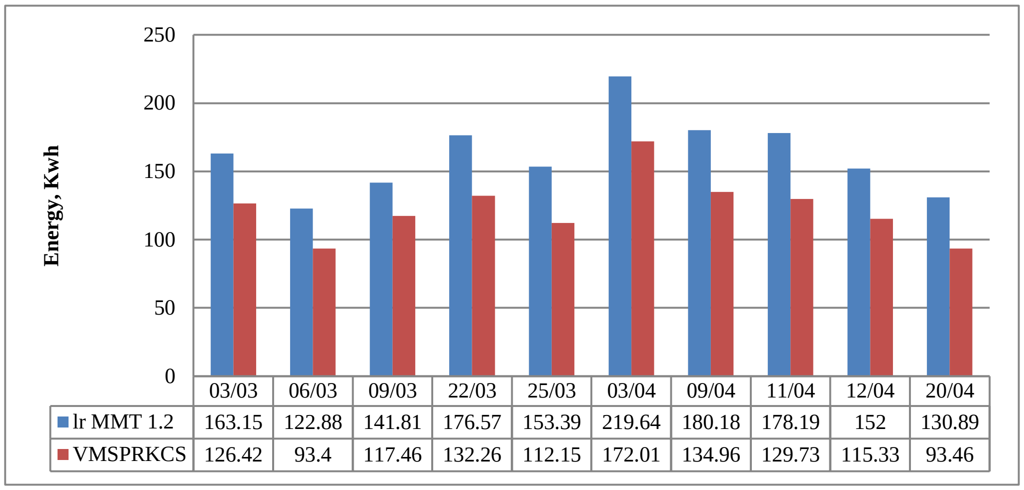
6) SLATAH reduced by minimum 60.08%, by maximum 81.9% and with average 71.07%, compared with lr_MMT 1.2.

7) Number of host shutdowns reduced by minimum 80.27%, by maximum 88.31% and with average 84.38%, compared with lr_MMT 1.2.

Finally, we ran the experiment for 10 times and calculated the median value, displaying it in terms of each performance metrics.

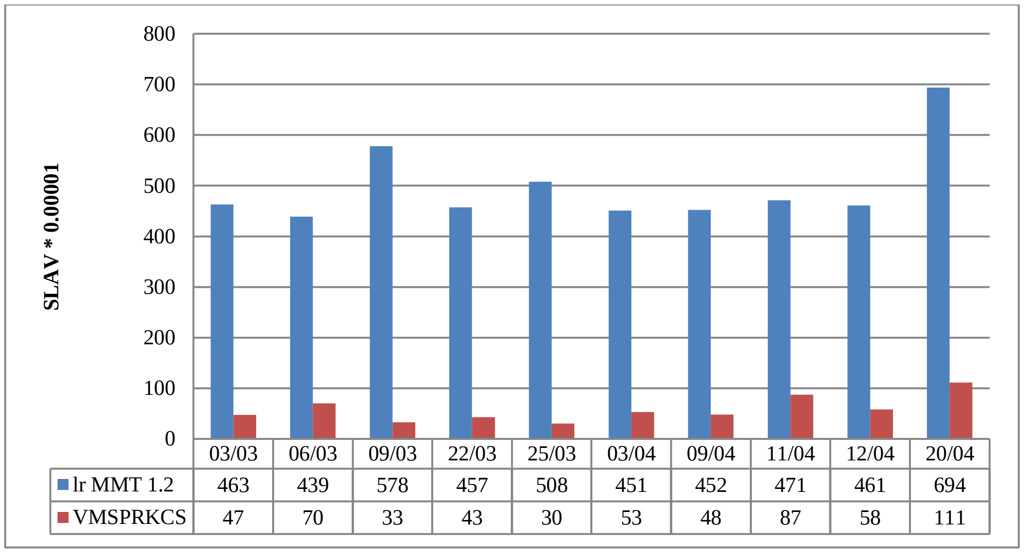
TABLE. VIII. PLANETLAB WORKLOAD TRACES CHARACTERISTICS

Date	No of Virtual Machines	Mean-Load (%)
03/03	1052	12.31
06/03	898	11.44
09/03	1061	10.70
22/03	1561	9.26
25/03	1078	10.56
03/04	1463	12.39
09/04	1358	11.12
11/04	1233	11.56
12/04	1054	11.54
20/04	1033	10.43



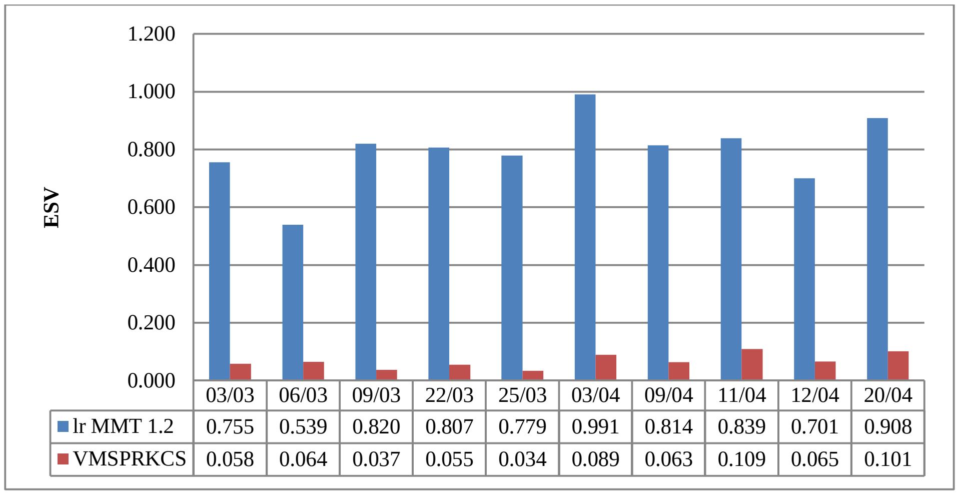
	03/03	06/03	09/03	22/03	25/03	03/04	09/04	11/04	12/04	20/04
🟦 lr MMT 1.2	163.15	122.88	141.81	176.57	153.39	219.64	180.18	178.19	152	130.89
🟥 VMSPRKCS	126.42	93.4	117.46	132.26	112.15	172.01	134.96	129.73	115.33	93.46

Fig. 1. Comparison of Energy Consumption.



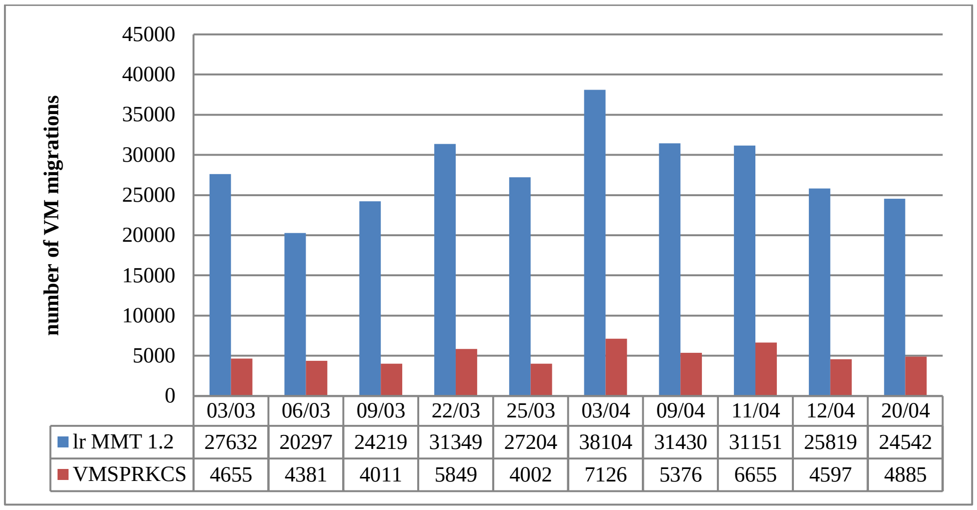
	03/03	06/03	09/03	22/03	25/03	03/04	09/04	11/04	12/04	20/04
🟦 lr MMT 1.2	463	439	578	457	508	451	452	471	461	694
🟥 VMSPRKCS	47	70	33	43	30	53	48	87	58	111

Fig. 2. Comparison of SLAV*0.0001.



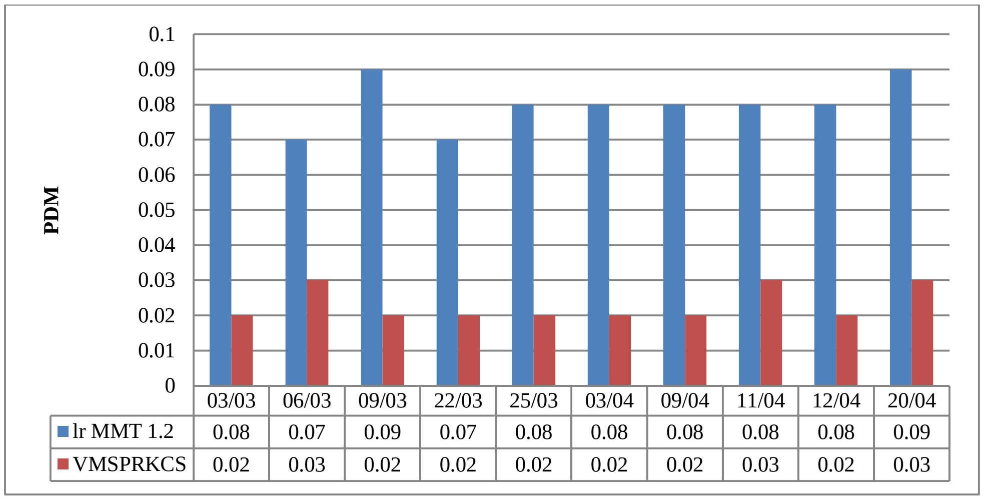
	03/03	06/03	09/03	22/03	25/03	03/04	09/04	11/04	12/04	20/04
🟦 lr MMT 1.2	0.755	0.539	0.820	0.807	0.779	0.991	0.814	0.839	0.701	0.908
🟥 VMSPRKCS	0.058	0.064	0.037	0.055	0.034	0.089	0.063	0.109	0.065	0.101

Fig. 3. Comparison ESV.



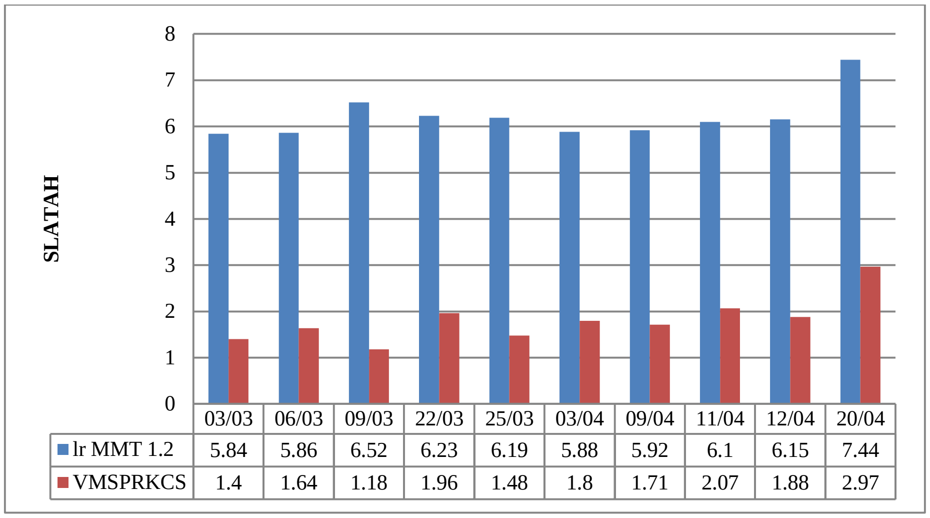
	03/03	06/03	09/03	22/03	25/03	03/04	09/04	11/04	12/04	20/04
🟦 lr MMT 1.2	27632	20297	24219	31349	27204	38104	31430	31151	25819	24542
🟥 VMSPRKCS	4655	4381	4011	5849	4002	7126	5376	6655	4597	4885

Fig. 4. Comparison of Number of VM Migrations.



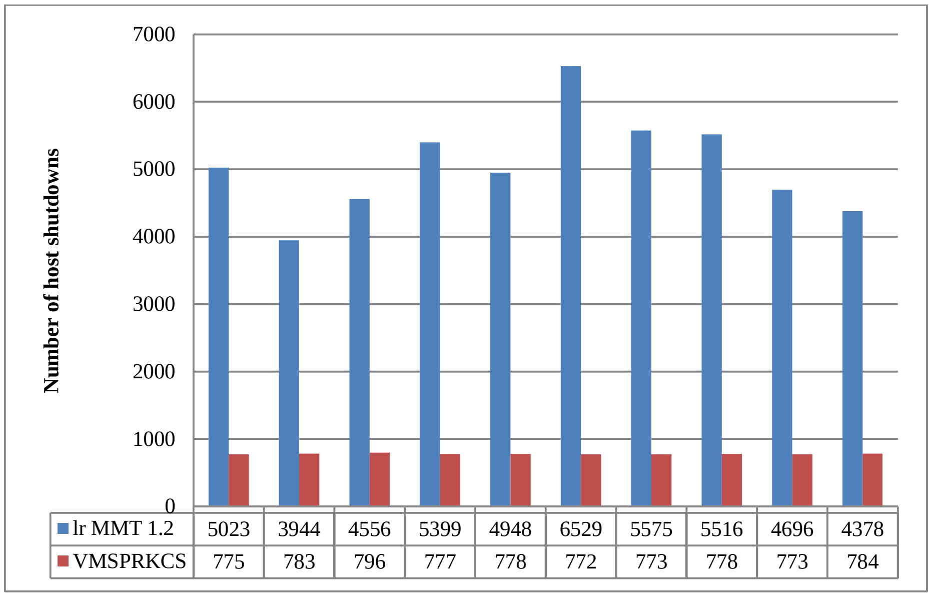
	03/03	06/03	09/03	22/03	25/03	03/04	09/04	11/04	12/04	20/04
🟦 lr MMT 1.2	0.08	0.07	0.09	0.07	0.08	0.08	0.08	0.08	0.08	0.09
🟥 VMSPRKCS	0.02	0.03	0.02	0.02	0.02	0.02	0.02	0.03	0.02	0.03

Fig. 5. Comparison of PDM.



	03/03	06/03	09/03	22/03	25/03	03/04	09/04	11/04	12/04	20/04
🟦 lr MMT 1.2	5.84	5.86	6.52	6.23	6.19	5.88	5.92	6.1	6.15	7.44
🟥 VMSPRKCS	1.4	1.64	1.18	1.96	1.48	1.8	1.71	2.07	1.88	2.97

Fig. 6. Comparison of SLATAH.



🟦 lr MMT 1.2	5023	3944	4556	5399	4948	6529	5575	5516	4696	4378
🟥 VMSPRKCS	775	783	796	777	778	772	773	778	773	784

Fig. 7. Comparison of Number of Host Shutdowns.

VII. Conclusion and Future Work

In this paper, a hybrid algorithm based on the Naive Bayesian Classifier and the Random Key Cuckoo Search is introduced to balance the tradeoff between the energy consumption and SLA violation. In addition, we modified the most common strategy for handling the VM consolidation by merging VM selection and placement steps into one to avoid any poor solution that may arise due to solving each of the two steps alone. We used Naive Bayesian Classifier to detect the future PM state in order to minimize the VMs migration, resulting in reducing energy, SLAV as well as performance degradation due to migration. We used Random Key Cuckoo Search to handle the VM selection and placement steps. In addition, CloudSim has been used with real traces provided from PlanetLab to evaluate the proposed algorithm compared with the benchmark algorithm lr_MMT 1.2 and the results have shown that the proposed algorithm can reduce the energy consumption by 24.23%, SLAV by 95.35%, ESV by 96.5%, the number of VM migrations by 89.3%, PDM by 86.3%, SLATAH by 71.07% and the number of host shutdowns by 84.38%. In this research, the objective function was considered based data center energy consumption only; in the future we are interested in calculating the objective function according to multiple objective metrics and comparing it with other meta-heuristic algorithms as well as more real datasets.

References

[1] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25(6), 599–616. doi: 10.1016/j.future.2008.12.001.

[2] Mell, P. M., & Grance, T. (2011). The NIST definition of cloud computing. doi: 10.6028/nist.sp.800-145.

[3] Shehabi, A., Smith, S., Sartor, D., Brown, R., Herrlin, M., Koomey, J, Lintner, W. (2016). United States Data Center Energy Usage Report. doi: 10.2172/1372902.

[4] Barroso, L. A., & Hölzle, U. (2007). The Case for Energy-Proportional Computing. Computer, 40(12), 33–37. doi: 10.1109/mc.2007.443.

[5] Sharma, N. K., & Reddy, G. R. M. (2015). Novel energy efficient virtual machine allocation at data center using Genetic algorithm. 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN). doi: 10.1109/icscn.2015.7219897.

[6] Beloglazov, A., & Buyya, R. (2011). 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, 24(13), 1397–1420. doi: 10.1002/cpe.1867.

[7] Fan, X., Weber, W.-D., & Barroso, L. A. (2007). Power provisioning for a warehouse-sized computer. Proceedings of the 34th Annual International Symposium on Computer Architecture - ISCA 07. doi: 10.1145/1250662.1250665.

[8] Keller, G., Tighe, M., Lutfiyya, H., & Bauer, M. (n.d.). An analysis of first fit heuristics for the virtual machine relocation problem. 2012 8th International Conference on Network and Service Management (Cnsm) and 2012 Workshop on Systems Virtualiztion Management (Svm), 406–413.

[9] Duda, R. O., Stork, D. G., & Hart, P. E. (2000). Pattern classification and scene analysis. New York: Wiley.

[10] Ouaarab, A., Ahiod, B., & Yang, X.-S. (2014). Random-key cuckoo search for the travelling salesman problem. Soft Computing, 19(4), 1099–1106. doi: 10.1007/s00500-014-1322-9.

[11] Yang, X.-S., & Deb, S. (2009). Cuckoo Search via Lévy flights. 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC). doi: 10.1109/nabic.2009.5393690.

[12] Bean, J. C. (1994). Genetic Algorithms and Random Keys for Sequencing and Optimization. ORSA Journal on Computing, 6(2), 154–160. doi: 10.1287/ijoc.6.2.154.

[13] Calheiros, R. N., Ranjan, R., Beloglazov, A., Rose, C. A. F. D., & Buyya, R. (2010). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, 41(1), 23–50. doi: 10.1002/spe.995.

[14] Park, K., & Pai, V. S. (2006). CoMon. ACM SIGOPS Operating Systems Review, 40(1), 65. doi: 10.1145/1113361.1113374.

[15] Shi, L., Furlong, J., & Wang, R. (2013). Empirical evaluation of vector bin packing algorithms for energy efficient data centers. 2013 IEEE Symposium on Computers and Communications (ISCC). doi: 10.1109/iscc.2013.6754915.

[16] Tighe, M., Keller, G., Bauer, M., & Lutfiyya, H. (n.d.). DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management. 2012 8th International Conference on Network and Service Management (Cnsm) and 2012 Workshop on Systems Virtualiztion Management (Svm).

[17] Digs-Uwo. (2014, December 7). digs-uwo/dcsim. Retrieved from https://github.com/digs-uwo/dcsim.

[18] Qiao, L., Liu, B., Hua, Y., Zhao, Q., & Fu, X. (2019). Genetic Expression Programming Based Dynamic Virtual Machine Consolidation in Cloud Computing. 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC). doi: 10.1109/iceiec.2019.8784588.

[19] Rui, Z., Shasha, H., & Hui, G. (2016). Research and Application of GEP Algorithm Based on Cloud Model. International Journal of Signal Processing, Image Processing and Pattern Recognition, 9(11), 309–318. doi: 10.14257/ijsip.2016.9.11.28.

[20] Li, L., Dong, J., Zuo, D., & Wu, J. (2019). SLA-Aware and Energy-Efficient VM Consolidation in Cloud Data Centers Using Robust Linear Regression Prediction Model. IEEE Access, 7, 9490–9500. doi: 10.1109/access.2019.2891567.

[21] Li, L., Dong, J., Zuo, D., & Liu, J. (2018). SLA-Aware and Energy-Efficient VM Consolidation in Cloud Data Centers Using Host States Naive Bayesian Prediction Model. 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). doi: 10.1109/bdcloud. 2018.00025.

[22] Alboaneen, D. A., Tianfield, H., & Zhang, Y. (2016). Glowworm Swarm Optimisation Algorithm for Virtual Machine Placement in Cloud Computing. 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld). doi: 10.1109/uic-atc-scalcom-cbdcom-iop-smartworld.2016.0129.

[23] Braiki, K., & Youssef, H. (2018). Multi-Objective Virtual Machine Placement Algorithm Based on Particle Swarm Optimization. 2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC). doi: 10.1109/iwcmc.2018.8450527.

[24] Tavakoli-Someh, S., & Rezvani, M. H. (2019). Utilization-aware Virtual Network Function Placement Using NSGA-II Evolutionary Computing. 2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI). doi: 10.1109/kbei.2019.8734978.

[25] Kusic, D., Kephart, J. O., Hanson, J. E., Kandasamy, N., & Jiang, G. (2008). Power and performance management of virtualized computing environments via lookahead control. Cluster Computing, 12(1), 1–15. doi: 10.1007/s10586-008-0070-y.

[26] The SPECpower benchmark. (n.d.). Retrieved from http://www.spec.org/power_ssj2008/.

[27] Liu, H., Xu, C.-Z., Jin, H., Gong, J., & Liao, X. (2011). Performance and energy modeling for live migration of virtual machines. Proceedings of the 20th International Symposium on High Performance Distributed Computing - HPDC 11. doi: 10.1145/1996130.1996154.