Self-adaptive Resource Management System in IaaS Clouds

Fahimeh Farahnakian*, Rami Bahsoon†, Pasi Liljeberg*, Tapio Pahikkala*
* Department of Information Technology, University of Turku, Turku, Finland.
† School of Computer Science, University of Birmingham, Birmingham, UK.
fahfar@utu.fi, r.bahsoon@cs.bham.ac.uk, pakrli@utu.fi, aatapa@utu.fi

Abstract—**Resource management in cloud infrastructures is one of the most challenging problems due to the heterogeneity of resources, variability of the workload and scale of data centers. Efficient management of physical and virtual resources can be achieved considering performance requirements of hosted applications and infrastructure costs. In this paper, we present a self-adaptive resource management system based on a hierarchical multi-agent based architecture. The system uses novel adaptive utilization threshold mechanism and benefits from reinforcement learning technique to dynamically adjust CPU and memory thresholds for each Physical Machine (PM). It periodically runs a Virtual Machine (VM) placement optimization algorithm to keep the total resource utilization of each PM within given thresholds for improving Service Level Agreement (SLA) compliance. Moreover, the algorithm consolidates VMs into the minimum number of active PMs in order to reduce the energy consumption. Experimental results on real workload traces show that our recourse management system provides substantial improvement over other approaches in terms of performance requirements, energy consumption and the number of VM migrations.**

Keywords-**Resource management, VM consolidation, reinforcement learning, energy-efficiency, SLA, green computing**

I. Introduction

Infrastructure as a Service (IaaS) has gained a lot of attraction over the past few years as a service model of cloud computing. IaaS providers such as Amazon EC2 and Rackspace are operating large data centers to deliver computing resources to cloud customers over the Internet. Several open source IaaS cloud management frameworks have been proposed in [1], [2], [3]. However, all these frameworks have a high degree of centralization and do not tolerate system component failures [4]. The centralized architectures are not scalable to control Virtual Machines (VMs) in a large-scale data center for three main problems. First, the worst-case computational complexity of a centralized controller is commonly proportional to the system size and thus cannot scale well for large-scale systems. Second, each server or Physical Machine (PM) in the data center may need to communicate with the centralized controller in every control period, and the controller may become a communication bottleneck. Third, a centralized controller may have long communication delays in the large-scale data centers.

To address these problems, we propose a novel Self-Adaptive Resource Management System (SARMS) in this paper. To achieve scalability, SARMS uses a hierarchical architecture that is partially inspired from HiVM [5] since we proved HiVM can scale up for thousands PMs. SARMS provides self-adaptive ability for resource management through an Adaptive Utilization Threshold (AUT) mechanism. This mechanism dynamically and adaptively adjusts utilization thresholds as static thresholds are not efficient for IaaS environments with mixed workloads. For this purpose, it uses Q-learning as one of the most popular of Reinforcement Learning (RL) algorithms. In Q-learning, an agent (decision-maker) learns by trial-and-error interaction with its dynamic environment and improves an existing policy in response to the change of the environment. Therefore, autonomy and adaptability are key features of Q-learning. Unlike previous works that only use the CPU threshold, AUT considers both CPU and memory thresholds in order to provide ”finer” grounds for analyzing what can cause SLA violations. In addition, SARMS runs a VM placement optimization algorithm iteratively, which uses AUT to keep the resource utilization within the thresholds, preventing a potential SLA violations. The algorithm also consolidates VMs into the minimum number of active PMs for reducing the energy consumption in IaaS cloud. Experimental results on real workloads from Google [6] and PlanetLab [7] data show that SARMS can reduce the energy consumption and the number of VM migrations while maintains required performance levels in the data center.

The remainder of the paper is organized as follows. Section II surveys some literature regarding to existing utilization threshold based resource management approaches and IaaS cloud management systems. Section III presents the proposed architecture, the AUT mechanism and the VM placement optimization algorithm. Section IV shows the implementation issue of our approach. Finally, we give experimental results and conclusion in Section V and VI.

II. Related Work

Recently, various approaches have been proposed for solving resource management as a multi-objective optimization problem. They imply a variety of possible formulations of the problem and define different objectives. Maintaining QoS between IaaS providers and their users is one of the main objectives for designing an efficient resource management approach. For this reason, most of the existing works use a static utilization threshold to avoid performance degradations. Secron [8] considers an upper threshold to prevent CPU’s PM from reach 100% utilization that leads to performance degradation. Moreover, a VM placement algorithm in [9] maintains the CPU utilization of each PM between the static upper and lower thresholds. Feller et al. [4] propose a static CPU threshold to detect under-loaded and over-loaded PMs. The simplicity and intuitive nature of these static threshold based approaches make them very appealing. However, setting static thresholds are not efficient for an environment with dynamic workloads, in which different types of applications may run on a PM. Therefore, threshold values should be tuned for each workload type and level to perform VM placement optimization efficiently. For this purpose, Beloglazov and Buyya in [10] present several methods of estimating an upper CPU threshold based on the statistical analysis of historical data. The authors also present a VM placement algorithm to migrate some VMs from a PM if the current CPU utilization of PM exceeds the upper threshold. However, this algorithm has considered only the current resource requirements and neglected the future resource demands. Therefore, it generates unnecessary VM migrations and increase the rate of SLA violations in data centers. To address this problem, we propose a regression based prediction model in [11] for forecasting resource utilization of both PMs and VMs.

Several IaaS cloud management frameworks such as OpenNebula [1], Nimbus [2], CloudStack [3], HiVM [5] and Snooze [4] have been developed during the past years. OpenNebula uses the traditional front-end and back-end system architecture. A controller in the front-end node accepts user requests and assigns them to the back-end nodes. A controller in the back-end node receives the requests and delegates them into the hypervisor. A similar system is proposed in [2], where authors present the Nimbus as IaaS cloud management framework. Either OpenNebula or Nimbus implement a centralized architecture. Moreover, CloudStack creates and manages VMs in a centralized cloud management system. To address the problem of the existing frameworks, Snooze presents a hierarchical architecture that can be scalable across many thousands of servers and VMs. We also designed a hierarchical architecture, HiVM, for improving the scalability, performing distributed VM management and energy efficient in IaaS. In this paper, we present a Self-Adaptive Resource Management System (SARMS) that uses a hierarchical architecture inspired from HiVM. In contrast to HiVM and the existing works discussed above, SARMS is different in the following ways:

- SARMS uses a novel Adaptive Utilization Threshold (AUT) mechanism to dynamically and adaptively adjust utilization thresholds using Q-learning algorithm. AUT goes beyond the existing works which only consider CPU threshold by taking into account memory. Combining both memory and CPU thresholds, SARMS can better identify causes of SLA violations and consequently prevent them from happening.
- SARMS proposes a VM placement optimization algorithm to keep the utilization of PMs within thresholds in order to avoid SLA violations. The algorithm also consolidates VMs into the minimum number of active PMs for energy consumption reduction in data center.
- We implemented and evaluated SARMS on a simulated data center using real Google and PlanetLab workloads. We experimentally show the added value of employing the AUT mechanism for VM management. We evaluate the benefits of enriching the architecture and VM management technique by comparing with HiVM. SARMS is also compared with a two-tier hierarchical architecture [12] to show the benefit of moving from two to three-tier hierarchical architecture. We experimentally evaluate the adaptivity for the utilization of resources using AUT by comparing SARMS against the three adaptive utilization threshold mechanisms and a static threshold mechanism presented in [10].

III. Self-Adaptive Resource Management System

A. Architecture

We consider a data center that consists of m heterogeneous PMs, $PM = \langle PM_1, ..., PM_m \rangle$. Each PM is characterized with D type of resources such as CPU, memory, network I/O and storage capacity. In addition, multiple VMs can be allocated to each PM through Virtual Machine Monitor (VMM) or hypervisor. In our implementation, the VMs are initially allocated to PMs based on the Best-Fit Decreasing (BFD) as one of the well-known heuristic algorithms. At any given time, users submit their requests for provisioning of n VMs, $VM = \langle VM_1, ..., VM_m \rangle$, which are allocated to the PMs. As the requested utilization of VMs and PMs vary over time, a resource management system should control physical and virtual resources according to the resource requirements. For this purpose, we present self-Adaptive resource Management system based on a hierarchical architecture. Fig. 1 shows an example of SARMS architecture which is mapped to a three-tier data center topology [13]. The three-tier topology is one of the most common network topologies for data centers due to its simplicity of wiring and reduced economical costs. In this topology, the lowest access tier contains hosts that connect to the Top-of-Rack (ToR) switches. The PMs are mounted in different racks (clusters) with a ToR switch. In the intermediate aggregation tier, the clusters are arranged into different modules with a pair of Aggregation Switches (ASs) servicing the module connectivity. Traffic from the access tier is forwarded to the core tier by ASs. Finally, the highest core tier provides secure connectivity between ASs and Core Switches (CSs) connected to the Internet.

The key idea of SARMS architecture is to split the resource management problem across multi agents where each agent solves a part of the prolem. Therefore, SARMS performs a distributed resource management based on a multi-agent based architecture that consists of four kinds of agents. At the core tier, Global Agents (GAs) receive the VM requests from the users and distribute them among different Module Agents (MAs) in the aggregation tier. Each MA dispatches the VM requests to Cluster Agents (CAs) based on the received information from CAs including the used and total capacity of clusters. Each CA receives the requests from MA and

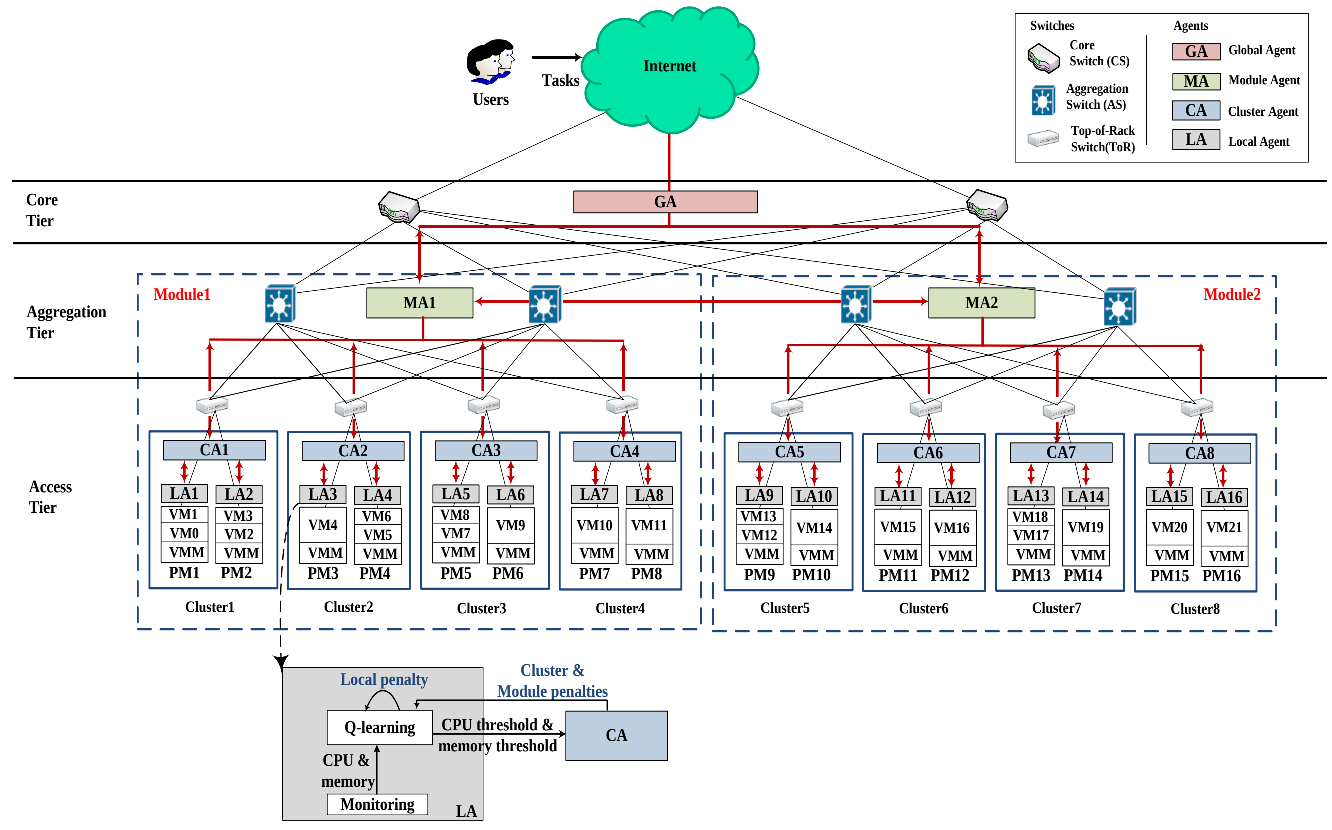

Fig. 1. An Instance of the Self-adaptive Resource Management System (SARMS) showing an instance of AUT

assigns them to Local Agents (LAs) in the cluster. Moreover, it periodically runs a VM placement optimization algorithm in order to reduce SLA violations and energy consumption in the data center. Each LA monitors a PM, detects overloaded/non-overloaded situation based on the Adaptive Utilization Threshold (AUT) mechanism and reports them to CAs. The number of LAs, CAs and MAs are equal to the number of PMs, ToRs, CSs, respectively.

B. Assumption

Each PM_i has a d-dimensional total capacity vector $C_{PM_i} = \langle C^1_{PM_i}, C^2_{PM_i}, ..., C^d_{PM_i} \rangle$, where $C^d_{PM_i}$ represents the total d-th resource capacity of PM_i. Each dimension corresponds to one type of physical resource (e.g., CPU capacity, memory, network I/O and disk storage). In addition, the used capacity vector of PM_i can be represented as $U_{PM_i} = \langle U^1_{PM_i}, U^2_{PM_i}, ..., U^d_{PM_i} \rangle$, where $U^d_{PM_i}$ denotes the used capacity of resource d. For instance, the used CPU capacity of a PM is estimated as the sum of the CPU utilization of the three VMs if three VMs are hosted by the same PM. The load of PM_i is modeled as the summation of the resource utilization ratio $R^d_{PM_i}$ in each individual resources $d \in D$ as

$$Load_{PM_i} = R^{CPU}_{PM_i} + R^{mem}_{PM_i} \tag{1}$$

We do not take into account the disk size dimension since network-attached storage (NAS) is used across the data center as main storage. $R^d_{PM_i}$ is the ratio of its used resource $U^d_{PM_i}$ to its total resource $C^d_{PM_i}$ as

$$R^d_{PM_i} = \frac{U^d_{PM_i}}{C^d_{PM_i}} \tag{2}$$

The load level of VM_i is defined as

$$Load_{VM_i} = R^{CPU}_{VM_i} + R^{mem}_{VM_i} \tag{3}$$

where $R^d_{VM_i}$ is the ratio of the requested d-th utilization of VM_i to the total d-th consumption by VM_i.

$$R^d_{VM_i} = \frac{U^d_{VM_i}}{C^d_{VM_i}} \tag{4}$$

C. PM Status Detection

To detect the status of each PM, each Local Agent (LA_i) utilizes an Adaptive Utilization Threshold (AUT) mechanism. PM_i is considered as a member of overloaded set P_{over} if the CPU or memory utilization exceeds an adaptive threshold. Otherwise, it is categorized as a member of non-overloaded set $P_{nonOver}$. Due to the variability of workload, the adaptive threshold $T^d_{PM_i}$ should be adjusted for each resource dimensions based on the current load. For this purpose, AUT uses Q-learning to learn on-line through experience from the environment and utilizes its knowledge to find a suitable value for each threshold. Thus, Q-learning provides a self-adaptive mechanism without a prior knowledge of the environment.

In Q-learning, LA_i first percepts the current state s of the environment and then performs an action a at the current time slot t. We define the state s as $(R^{CPU}_{PM_i}, R^{mem}_{PM_i})$ in AUT mechanism. $R^{CPU}_{PM_i}$ and $R^{mem}_{PM_i}$ are the CPU and memory utilization ratio, respectively.

Based on the observed state s, LA_i selects a utilization threshold values $T^d_{PM_i}$ of $\{0.5, 0.6, 0.7, 0.8, 0.9\}$ as an action a for each d-th resource. Since the risk of SLA violation is created when the utilization of a PM is close to 100%, we limit the amount of resources under 90%. During the beginning of the learning process, the LA should perform a certain amount of exploration actions, so that it selects an action randomly. Then, LA_i chooses an action based on its experience (exploitation). It is clear that selecting an action is more exploration at the beginning of learning, and is more exploitation towards the end of learning.

At the beginning of next time slot $t+1$, the environment transit into a new state $\acute{s}$ and LA_i improves its knowledge based on a reinforcement signal. This signal can reflect the success or failure of the system after an action has occurred. In this paper, we consider the signal as a total value of three penalties local, cluster and module penalties. As LA_i pays the penalties for performing action a, it tries to minimize its average long-term penalties during the learning mechanism. In fact, each penalty is an objective function in each tier of the proposed architecture. The objective function determines the impact of agent's action on the performance and power.

The **local penalty** LP_i indicates the impact of the local agent i's decision on the power and performance of PM i. As this penalty is calculated by the LA, we called the local penalty. AUT should make intelligent tradeoffs between power and performance as switching PMs into power-saving mode definitely degrades the performance level. Thus it considers a multi-objective function $LP_i(SLAV, POW)$ taking both power and performance into account, and uses it to give penalties in RL. The performance requirements can be formalized via Service Level Agreement (SLA). The SLA violation of the PM i, $SLAV^{t+1}_{PM_i}$, is the difference between the requested resources by all VMs and the actually allocated resources at time slot $t+1$. The local penalty is a linear function of SLA violation $SLAV$ and total power consumption POW consumed by PM i in the time slot $t+1$ as

$$LP_i(SLAV, POW) = SLAV^{t+1}_{PM_i} + \beta \times POW^{t+1}_{PM_i}$$

where β is a tunable coefficient indicating the relation of power and performance objectives.

The **cluster penalty** CP^j_i is sent from the cluster agent j to the local agent i. This penalty represents the impact of local agent i's decision on other PMs in the same cluster. Therefore, the cluster penalty is the mean of local penalties in the cluster j exclusive of LP_i. On the other hand, the local agent i can get an overall view of the performance and power in the cluster as

$$CP^j_i = \frac{(\sum_{y=1}^{Y} LP_y) - LP_i}{Y-1}$$

where Y is the number of local agents in the cluster j.

The CA_j is received the **module penalty** MP^z_j from the module agent z. This penalty shows the mean of cluster penalties in other clusters of the module exclusive of its cluster penalty.

$$MP^z_j = \frac{(\sum_{x=1}^{X} CP^j_x) - CP^j_i}{X-1}$$

where X is the number of cluster agents in the module z.

Finally, LA_i updates a Q-value, $Q_{t+1}(s,a)$, that is related for each pair of action-state through the total penalties P^{t+1}_i

$$P^{t+1}_i = LP_i + CP^j_i + MP^z_j$$

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha[P^{t+1}_i + \gamma \min_{a \epsilon A} Q_t(\acute{s},a) - Q_t(s,a)]$$

where α is a learning rate. The learning rate can take a value between zero and one; the value of zero means that no learning takes place by the algorithm; while the value of one indicates that only the most recent information is used. The discount factor γ is a value between 0 and 1 which gives more weight to the penalties in the near future than the far future. $Q_{t+1}(s,a)$ represents the expected power and performance caused by the action a at state s. Therefore, the local agent selects an action with the minimum Q-value when it percepts the state s again.

D. VM Placement Optimization

In order to reduce SLA violations and energy consumption, each cluster agent runs an instance of VM placement optimization algorithm (Algorithm 1) periodically. The algorithm creates a migration plan M as an output of two steps.

At the first step (line 1-22), the algorithm migrates some VMs from the over-loaded PMs for avoiding SLA violations. It starts from the VMs that require the minimum migration time. The migration time is calculated with dividing the memory assigned to the VM, by the available network bandwidth between source and destination PMs. The algorithm sorts all VMs on the p_{so} in ascending order of the used memory capacity as all network links have 1GBPS bandwidth in our simulation (line 3). Then, it starts to migrate VMs until the source PM is still considered as a member of overloaded set P_{over} (line 4 and 5). To find an appropriate destination PM p_{de} for reallocating the migrated VM v, the algorithm first considers non-overloaded set $P_{nonOver}$ in the cluster (line 7-12). If the aggregated resource utilization of VM and PM is lower than the adaptive thresholds, then the PM is selected as p_{de} (line 8). If the algorithm can not find p_{de} in the same cluster, it sends a request to the module agent to find it in the other clusters (line 13-15). The algorithm can wake up a sleeping PM if the active PMs do not have sufficient resource for allocation v. Finally, the new VM placement is added to a migration plan $M1$ as a member (line16). The migration plan is a set of 3-tuple $(p_{so}; v; p_{de})$, where the source PM p_{so}, the VM to be migrated v, and the destination PM p_{de}. Moreover, the utilization ratio of source and destination PMs are updated (line 17).

At the second step (line 23-45), the algorithm consolidates VMs on the none-overloaded PMs. The algorithm sorts the non-overloaded PMs $P_{nonOver}$ in decreasing order based on their load level (line 24). Then, it starts from the least-loaded PM of the list and considers it as a source PM p_{so} (line 25 and 26). It tries to migrate all VMs and releases p_{so}. To select which VMs first migrate from p_{so}, the algorithm sorts all VMs on p_{so} in decreasing order based on their load level (line 27). The algorithm starts from the first PM (the most-loaded PM) of set $P_{nonOver}$ (line 30) to find p_{de}. If it is not possible, the second PM will be selected and so on. The algorithm selects p_{de} that has required capacity for allocating the VM considering thresholds (line 31). Finally, the new VM placement is added to a migration plan $M2$ as a member (line 32). The resource used capacity of source and destination PMs are updated to reflect the impact of the new VM placement (line 33). The variable *success* is defined for checking whether all VMs form p_{so} are migrated or not. Either all VMs from the p_{so} are migrated if one of them fails, non of them are migrated. Therefor, the algorithm removes all tuples in the migration plan and recovers the resource capacity of source and destination PMs if the value of *success* is false (line 39-41). Otherwise, idle p_{so} is switched to the sleep mode when all of its VMs migrate from it (line 42-44). The output of the algorithm is a migration plan M that combines all migration tuples of the first and second steps (line 46 and 47). Finally, the cluster agent sends commands to local agents in the cluster based on the migration plan M for performing VM migrations.

Algorithm 1 VM placement optimization algorithm

```
 1: $M1 = \emptyset$
 2: for $p_{so} \in P_{over}$ do
 3:     $V_m \leftarrow$ sort VMs on PM $p_{so}$ in ascending order of $U_v^{mem}$
 4:     for $v \in V_m$ do
 5:         if $p_{so} \in P_{over}$ then
 6:             $p_{de} = \emptyset$
 7:             for $p \in P_{nonOver}$ do
 8:                 if $(R_p^{CPU} + R_v^{CPU} \leq T_p^{CPU})$ & $(R_p^{mem} + R_v^{mem} \leq T_p^{mem})$ then
 9:                     $p_{de} = p$
10:                     break;
11:                 end if
12:             end for
13:             if $p_{de} = \emptyset$ then
14:                 send a request to the module agent to find $p_{de}$
15:             end if
16:             $M1 = M1 \cup \{(p_{so}, v, p_{de})\}$
17:             Update $R_{p_{so}}$ and $R_{p_{de}}$
18:         else
19:             break;
20:         end if
21:     end for
22: end for
23: $M2 = \emptyset$
24: sort $P_{nonOver}$ in descending order of $Load_p$
25: for $i = |P_{nonOver}|$ to 1 do
26:     $p_{so} = P_{nonOver}[i]$;
27:     $V_m \leftarrow$ sort VMs on PM $p_{so}$ in descending order of $Load_v$
28:     for $v \in V_m$ do
29:         $success = false$
30:         for $p_{de} \in P_{nonOver} - p_{so}$ do
31:             if $(R_{p_{de}}^{CPU} + R_v^{CPU} \leq T_{p_{de}}^{CPU})$ & $(R_{p_{de}}^{mem} + R_v^{mem} \leq T_{p_{de}}^{mem})$ then
32:                 $M2 = M2 \cup \{(p_{so}, v, p_{de})\}$
33:                 Update $R_{p_{so}}$ and $R_{p_{de}}$
34:                 $success = true$
35:                 break;
36:             end if
37:         end for
38:     end for
39:     if $success = false$ then
40:         $M2 = \emptyset$
41:         Recover $R_{p_{so}}$ and $R_{p_{de}}$
42:     else
43:         Switch $p_{so}$ to the sleep mode
44:     end if
45: end for
46: $M = M1 \cup M2$
47: return $M$
```

E. VM Assignment

The VM assignment problem is solved in three tiers of SAMS. At the core and aggregation tiers, global and module agents run BFD algorithm to assign first VMs to modules and then VMs to clusters, respectively. The BFD algorithm sorts all VMs in the decreasing order based on the their load level. It aims to assign most-loaded VM to the most-loaded module or cluster if the module or cluster has enough capacity for allocating the VM. This constraint should hold in order to avoid performance degradation. In the access tier, the cluster agent assigns each VM to a PM if the PM has sufficient CPU and memory for allocating the VM based on the adaptive thresholds (line 8 in Algorithm 1). Therefore, it can limit the amount of requested resources by the VM below the adaptive thresholds and so that minimizes performance degradation.

IV. Experimental Setup and Design

A. Workload

We evaluated the proposed approach on two real workload traces: Google Cluster Data (GCD) [6] and PlanetLab data [7]. GCD provides real trace data of a Google cluster over about one-month period in May 2011.

This trace involves over 650k jobs across over 12000 heterogenous PMs. Thousands of users repeatedly used these jobs that each job consists of one or more tasks. Each task represents a Linux program possibly consisting of multiple processes and generates with a set of user customized requirements such as CPU (core-seconds), memory, disk space, disk time fraction (I/O seconds). The usage of each type of resources is collected at five minutes intervals. For our experiments, we extracted the task duration based on the time when the task was scheduled last and the time when the task finished. Furthermore, we also extracted the task utilization values of CPU and memory over the first ten days.

TABLE I
CHARACTERISTICS OF THE WORKLOAD DATA

Workload	CPU Mean (%)	CPU St.dev (%)	CPU Median	Memory Mean (%)	Memory St.dev (%)	Memory Median
GCD	10.87	10.85	7	22.87	16.05	20
PlanetLab	25.44	14.16	22	10.48	11.06	7

We use the job ID as the unique identifier for a job, and for each of these jobs we extracted a set of actual usage for each resource for all of its tasks. The attributes that we considered for CPU and memory are: the CPU rate, which indicates the average CPU utilization for a sample period of 5 minutes, and the canonical memory usage, which represents the average memory consumption for the same sampling period.

PlanetLab data is provided as a part of the CoMon project, a monitoring infrastructure for PlanetLab. In this project, the CPU and memory usage data is reported every five minutes from more than a thousand VMs and is stored in ten files. In fact, the workload is representative of an IaaS cloud environment such as Amazon EC2. GCD and PlanetLab VMs corresponding to their CPU and memory utilization characteristics are presented in Table I. In both workload traces, VM request with a CPU or RAM consumption higher than 90% and lower than 5% where also removed from the experiments.

B. Simulation Setup

To evaluate the efficiency of our proposed approach, we set up experimental environment using the CloudSim toolkit [14]. Table II summarizes the main simulation setup parameters for two proposed workloads. We simulated a data center comprising several heterogeneous PMs. The half of PMs 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. Each PM is modeled to have 2 cores, 4GB memory and 1 GB/s network bandwidth. The CPU MIPS rating and the memory amount characteristics of four VM instances used in CloudSim corresponded to Amazon EC2 [1], i.e., 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, 613 MB). α and γ (Q-learning parameters) are 0.5 and 0.8, respectively. The value of these parameters were obtained in a series of preliminary experiments.

C. Evaluation Metrics

The main aim of the evaluation is show the implications of enriching the architecture with the adaptive threshold mechanism on: i) guarantee that SLAs are not violated; ii) minimize the number of PMs used; iii) minimize the number of VM migrations. Therefore, the performance of proposed approach is assessed through the following metrics:

SLA Violations: a workload independent metric (SLAV) is proposed in [10] that can be used to evaluate the SLA delivered by any VM deployed in an IaaS. SLAV is measured by the SLA violations due to over-utilization (SLAVO) and SLA violations due to migration (SLAVM). Both SLAVO and SLAVM

TABLE II
SIMULATION SETUP PARAMETERS

	Parameter	GCD	PlanetLab
Physical Architecture	Number of PMs	1600	264
	Number of VMs	1600	264
	Number of modules	8	3
	Number of clusters	32	12
	Number of PMs in each cluster	50	22
	Number of clusters in each module	4	4
	Core Switches	8	3
	Aggregation Switches	16	6
	Top-of-Rack Switches	32	12
Control Architecture	Global Agents	4	1
	Module Agents	8	3
	Cluster Agents	32	12
	Local Agents	1600	264

metrics independently and with equal importance characterize the level of SLA violations by the infrastructure. Therefore, both performance degradation due to host overloading and due to VM migrations are proposed as a combined metric (SLAV)

$$SLAV = SLAVO \times SLAVM \tag{5}$$

In this paper, SLAVO indicates the percentage of time, during which active PMs have experienced the CPU or memory utilization of 100% as

$$SLAVO = \frac{1}{m}\sum_{i=1}^{m}\frac{T_{s_i}}{T_{a_i}} \tag{6}$$

where m is the number of PMs; T_{s_i} is the total time that the PM i has experienced the CPU or memory utilization of 100% leading to an SLA violation. T_{a_i} is the total of the PM i being the active state. SLAVM shows the overall performance degradation by VMs due to migrations as

$$SLAVM = \frac{1}{n}\sum_{j=1}^{n}\frac{C_{d_j}}{C_{r_j}} \tag{7}$$

where n is the number of VMs; C_{d_j} is the estimate of the performance degradation of the VM j caused by migrations; C_{r_j} is the total CPU capacity requested by the VM j during its lifetime. In our experiments, we estimate C_{d_j} as 10% of the CPU utilization during all migrations of the VM j.

Energy consumption: we consider the total energy consumption by the physical resources of a data center caused by application workloads. The energy consumption of PMs depends on the utilization of a CPU, memory, disk and network card. Most studies show that CPU consumes more power than other devices such as memory, disk storage and network interface [10], [15]. Therefore, the resource utilization of a PM is usually represented by its CPU utilization. Here the energy consumption is measured based on real data on power consumption provided by the results of the SPECpower

TABLE III
THE ENERGY CONSUMPTION AT DIFFERENT LOAD LEVELS IN WATTS

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

benchmark[1] instead of using an analytical model of server power consumption. Table III 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.

Number of VM Migrations: live migration has a negative impact on the performance of applications running in a VM during a migration. It also causes negative impacts in infrastructure service provider such as co-located VM performance and network congestion. Therefore, one of our objectives is minimizing the number of VM migrations.

V. EXPERIMENTAL RESULTS

We implemented and evaluated the Self-Adaptive Resource Management System (SARMS) on a simulated data center. We discuss the added value of employing an adaptive threshold mechanism and hierarchical style by comparing SARMS with the following approaches:

- HiVM [5]: uses a utilization prediction based VM management approach. This approach optimizes the VM placement according to the current and future resource utilization. We compare SARMS with HiVM to show the benefits of the proposed extensions (i.e. the introduced self-adaptivity).
- Multi Agents-based Dynamic Consolidation (MADC) [12]: proposes a two-tier hierarchical multi-agent based architecture for VM management. We compared SARMS with MADC to show the benefit of moving from two to three-tier hierarchical architecture.
- Three adaptive utilization threshold algorithms [10]: first adapt the upper CPU utilization thresholds dynamically based on the classical statistical methods: Median Absolute Deviation (MAD), the Interquartile Range (IQR) and Local Regression (LR). Then, a VM placement optimization reallocates VMs for load blanching if total CPU utilization of a PM exceeds the upper threshold. By comparing SARMS against these algorithms, we can show the impact AUT as a Q-learning based adaptive threshold mechanism against three statical analysis based adaptive algorithms.
- Static threshold (THR) method [10]: monitors the CPU utilization and migrates a VM when the current utilization exceeds 80% of the total amount of available CPU capacity on the PM. Comparison THR against SARMS can show the performance of the adaptive threshold utilization mechanism for VM management.

[1] $http://www.spec.org/power\_ssj2008/$

Figure 2 illustrates the SLAV, energy consumption and number of migrations by SARMS, HiVM, MADC, LR, MAD, IQR and THR methods in GCD workload. SARMS can reduce the SLA violations rate more efficiently than other techniques (Figure 2(a)). The obtained results can be explained by the fact that the proposed VM optimization algorithm as a main part of SARMS uses the AUT mechanism to keep the utilization of PMs below the adaptive thresholds. We also observe a significant reduction in the energy consumption 13.7%, 25.7%, 43.6%, 59.4%, 66.4% and 69.6% when we compared with HiVM, MADC, LR, MAD, IQR and THR, respectively. This is because, the VM optimization algorithm minimizes the number of non-overloaded PMs. Moreover, Figure 2(c) depicts SARMS performs well in terms of minimizing the number of migrations due to follow all-or-nothing property.

The SLAV metric for the PlanetLab workload is shown in Figure 3(a). The results show that SARMS leads to significantly less SLA violations than other benchmark algorithms. The main reason is that SARMS prevents SLA violations by migrating some VMs from a PM when memory or CPU utilization exceeds the adaptive thresholds. In addition, Figure 3(b) shows SARMS provides higher energy saving in comparison to other methods. It reduces the energy consumption up to 5.9% in PlanetLab workload. This is because, the SARMS tries to consolidate VMs into the minimum number of non-overloaded PMs by running second step of the VM placement optimization algorithm. Furthermore, SARMS is more efficient in reducing the number of migrations (Figure 3(c)). This is due to the fact the VM placement algorithm follows all-or-nothing property to migrate VMs. Thus, the algorithm can avoid unnecessary VM migrations and reduce the rate of SLA violations in data centers.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented a Self-Adaptive Resource Management System (SARMS) for efficient resource management in cloud infrastructure. SARMS uses a novel Adaptive Utilization Threshold (AUT) mechanism to dynamically adjust utilization thresholds for each Physical Machines (PMs). As a part of SARMS, we presented a VM placement optimization algorithm to prevent SLA violations by migrating some VMs from a PM when the resource utilization exceeds the threshold. Moreover, the algorithm consolidates VMs into the minimum number of active PMs to reduce the energy consumption in data centers. The obtained results of real Google and PlanetLab workload traces show that SARMS significantly outperforms benchmark algorithms in terms of energy consumption, performance requirements and number of VM migrations. As a future work, we have identified three improvement directions for the SARMS. First, we plan to improve AUT by tuning

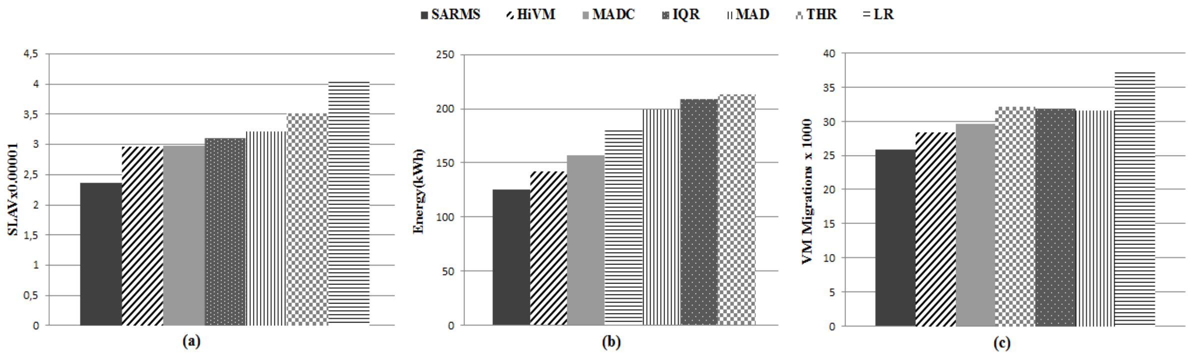

Fig. 2. The SLAV metric, energy consumption and number of VM migrations by SARMS and benchmark approaches for the GCD workload trace

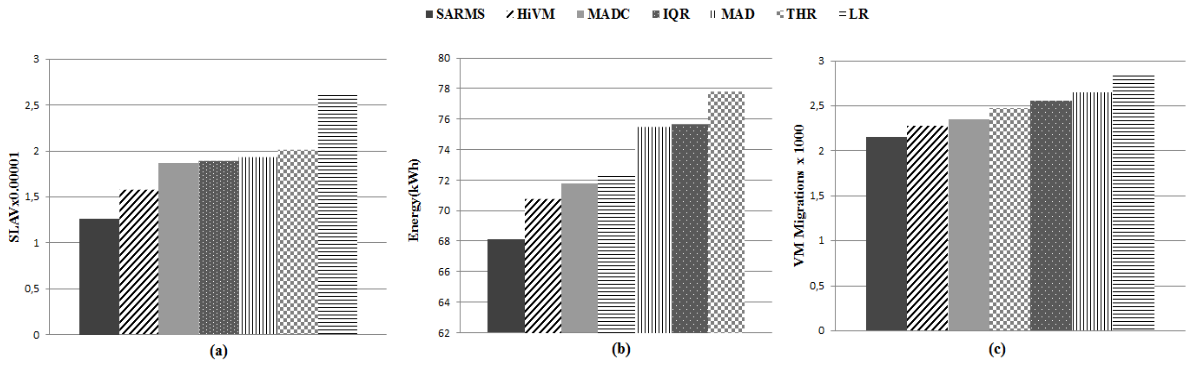

Fig. 3. The SLAV metric, energy consumption and number of VM migrations by SARMS and benchmark approaches for the PlanetLab workload trace

the learning rate parameter in Q-learning according to the current load in each PM. The second improvement proposed a network-aware VM placement algorithm to balance network traffic and improve network resource utilization the data center. The third improvement aims to evaluate SARMS in a real cloud environment.

References

[1] D. Milojii, I. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *Internet Computing, IEEE*, vol. 15, no. 2, pp. 11–14, March 2011.

[2] K. Keahey, T. Freeman, J. Lauret, and D. Olson, "Virtual workspaces for scientific applications," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012038, 2007. [Online]. Available: http://stacks.iop.org/1742-6596/78/i=1/a=012038

[3] "The apache software foundation. cloudstack: Open source cloud computing," 2012, http://www.cloudstack.org/.

[4] E. Feller, L. Rilling, and C. Morin, "Snooze: A scalable and autonomic virtual machine management framework for private clouds," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, 2012, pp. 482–489.

[5] F. Farahnakian, P. Liljeberg, T. Pahikkala, J. Plosila, and H. Tenhunen, "Hierarchical vm management architecture for cloud data centers," in *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, Dec 2014, pp. 306–311.

[6] "Traces of google workloads," 2015, http://code.google.com/p/googleclusterdata/.

[7] K. Park and V. Pai, "CoMon: a mostly-scalable monitoring system for PlanetLab," *ACM SIGOPS Operating Systems Review*, vol. 40, pp. 65 – 74, 2006.

[8] A. Murtazaev and S. Oh, "Sercon: Server consolidation algorithm using live migration of virtual machines for green computing," *IETE Technical Review*, vol. 28, no. 3, pp. 212–231, 2011.

[9] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Grid Computing and eScience, Future Generation Computer Systems (FGCS)*, vol. 28, pp. 755–768, 2012.

[10] A. Beloglazov and Buy, "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 (CCPE)*, vol. 24, pp. 1397–1420, 2012.

[11] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, and H. Tenhunen, "Utilization prediction aware vm consolidation approach for green cloud computing," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, 2015, pp. 381–388.

[12] F. Farahnakin, T. Pahikkala, P. Liljeberg, J. Plosila, and H. Tenhunen, "Multi-agent based architecture for dynamic vm consolidation in cloud data centers," in *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, Aug 2014, pp. 111–118.

[13] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani, "Data center network virtualization: A survey," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 2, pp. 909–928, Second 2013.

[14] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience (SPE)*, vol. 41, pp. 23 – 50, 2011.

[15] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," in *Autonomic Computing, 2008. ICAC '08. International Conference on*, June 2008, pp. 3–12.