THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

TRA4/17

Learn-as-you-go with Megh: Efficient Live Migration of Virtual Machines

Debabrota Basu, Xiayang Wang, Yang Hong, Haibo Chen, and Stéphane Bressan

April 2017

T e c h n i c a l R e p o r t

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

Mohan KANKANHALLI
Dean of School

Learn-as-you-go with Megh: Efficient Live Migration of Virtual Machines

Debabrota Basu*, Xiayang Wang†, Yang Hong†, Haibo Chen†, Stéphane Bressan*
*School of Computing, National University of Singapore, Singapore
†Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University, Shanghai, China

Abstract*—Cloud providers leverage live migration of virtual machines to reduce energy consumption and allocate resources efficiently in data centers. Each migration decision depends on three questions: when to move a virtual machine, which virtual machine to move and where to move it? Dynamic, uncertain and heterogeneous workloads running on virtual machines make such decisions difficult. Knowledge-based and heuristics-based algorithms are commonly used to tackle this problem. Knowledge-based algorithms, such as MaxWeight scheduling algorithms, are dependent on the specifics and the dynamics of the targeted Cloud architectures and applications. Heuristics-based algorithms, such as MMT algorithms, suffer from high variance and poor convergence because of their greedy approach. We propose a reinforcement learning approach. This approach does not require prior knowledge. It learns the dynamics of the workload as-it-goes. We formulate the problem of energy- and performance-efficient resource management during live migration as a Markov decision process. While several learning algorithms are proposed to solve this problem, these algorithms remain confined to the academic realm as they face the curse of dimensionality. They are either not scalable in real-time, as it is the case of MadVM, or need an elaborate offline training, as it is the case of Q-learning. We propose an actor-critic algorithm, Megh, to overcome these deficiencies. Megh uses a novel dimensionality reduction scheme to project the combinatorially explosive state-action space to a polynomial dimensional space with a sparse basis. Megh has the capacity to learn uncertain dynamics and the ability to work in real-time. Megh is both scalable and robust. We implement Megh using the CloudSim toolkit and empirically evaluate its performance with the PlanetLab and the Google Cluster workloads. Experiments validate that Megh is more cost-effective, incurs smaller execution overhead and is more scalable than MadVM and MMT. We explicate our choice of parameters through a sensitivity analysis.

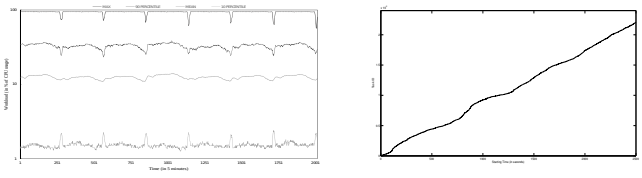
I. Introduction

Infrastructure as a Service (IaaS) environments of Cloud computing leverage virtualization technology [1] to provide a shared platform of resources accessible at any time and from anywhere through the Internet. Cloud providers allocate Virtual Machine instances (VM) on a cluster of Physical Machines (PM). As VMs allow users to share physical resources concurrently, they enhance utilization of resources and therefore increase return on investment for Cloud providers. Making such an optimal allocation is challenging because a large number of users accessing the Cloud, the diversity of applications, and the heterogeneity of hardware yield significant variations in performance not only in general-purpose IaaS Clouds but also in Clouds with specialised features like scientific computing, online transaction or data storage. Furthermore, the uncertain dynamics of workloads, creating abrupt and unpredictable changes in resource utilization, requires dynamic allocation of VMs. In order to avoid such disruption of service, [2] and [3] proposed the idea of a live migration scheme. During live migration, pages from the memory of the migrating VM are copied to the destination machine while it keeps on running on its present host. When properly carried out, this process takes place with minimal downtime and minimal noticeable effect from the user end. Live migration raises three questions to the Cloud administrator: *which* VM to move, *where*, i.e, to which physical host to move, and *when* to move?

These resource management decisions during live migration drastically affect the energy consumption of the Cloud data centres. As energy consumption contributes almost 75% of the operation cost of a data center [4], from the Cloud provider side it is the most important metric for live migration. Migration events may also cause significant deterioration of the Quality of Service (QoS) promised by the Cloud providers and can violate the Service Level Agreements (SLAs) [5]. These agreements also define monetary penalties for the Cloud providers when violated. In this work, we develop cost models for the SLA violations and the energy consumption during a live migration and aggregate them to construct an operation cost.

The workloads running on such VMs are uncertain, dynamic and heterogeneous due to eclectic requirements of users, miscellaneous applications ranging from scientific computing to data storage and diverse Cloud architectures. These features are depicted in Figures 1(a) and 1(b). This nature of workload makes energy- and performance-efficient resource management in data centres difficult. Knowledge-based and heuristics-based algorithms are applied for this purpose. Knowledge-based algorithms, such as MaxWeight scheduling algorithms [18], are oblivious to the specifics and the dynamics of Cloud architectures and applications that do not belong to their knowledge-base. Heuristics like dynamic consolidation algorithms [6], [7] do not use such specific knowledge base. They save the power by greedily accumulating a majority of VMs on a smaller number of servers. They improve the performance by taking cost-effective VM migration decisions from under- or over-utilized servers. These heuristics may become unstable while tackling uncertain dynamics. They may make suboptimal decisions due to their myopic and greedy nature.

It has motivated us to look into reinforcement learning (RL) methods. RL [8] is a paradigm of machine learning, where an agent operating in an uncertain environment tries to take optimal decisions, by learning more about the dynamics of its surroundings. If we consider the Cloud administrator system as a learning agent and the user workloads operating



(a) PlanetLab Workload (b) Google Cluster workload

Figure 1. Dynamics of PlanetLab workloads and starting times of tasks in Google Cluster.

on it with corresponding resource distribution as the uncertain environment, our problem manifests as an RL problem, Here, the system tries to take *optimal* live migration decisions by learning the dynamics of the workload and adapting accordingly. The policies or the sequence of decisions made by RL optimize a long-term goal, which is live migration and resource management of data center with minimum operation cost, based on immediate costs of actions or decisions. As the number of ways the VMs can be allocated to the hosts or PMs is combinatorially large, it creates a huge state space and also makes RL intractable. This curse of dimensionality restricts the applicability of recently proposed learning algorithms in real-life scenarios. They are either not scalable in real-time, as it is the case of MadVM, or need an elaborate offline training, as it is the case of Q-learning. We propose an actor-critic algorithm, called Megh, to solve this problem as-it-goes. It maps the state space into a smaller vector space and learns the dynamics of the workloads without assuming any model or prior knowledge. It makes Megh a robust scheme to learn the uncertainty and diversity of workloads without using any prior model. At each step, the sparsity of the projected space is leveraged to act effectively without creating any significant overhead in the course of live migration. The data structure exploiting this sparsity makes Megh time-efficient and therefore, a contending real-time solution for energy- and performance-efficient live migration.

We evaluate the performance of our system by simulating it using the CloudSim toolkit [9] over workload data extracted from PlanetLab [10] and Google Cluster [11]. We compare Megh with state-of-the-art dynamic consolidation based Minimum Migration Time (MMT) algorithms: THR-MMT, IQR-MMT, MAD-MMT, LR-MMT, and LRR-MMT [6], [7]. We also test the performance of Megh against MadVM [12], which is the most recent RL-based algorithm for dynamic resource management in a data center. Experiments prove the efficiency of Megh as it significantly reduces the total operation cost and the number of VM migrations occurring over a period of time with respect to the competing algorithms. Unlike MadVM suffering from the curse of dimensionality, Megh takes significantly smaller execution time than MMT heuristics even for large data center configurations. The results validate the robustness, efficiency and real-time execution of Megh to cost-effectively decide live migrations under uncertain workload dynamics. We also produce a comparative scalability analysis demonstrating Megh's better scalability than THR-MMT. We explicate our choices of parameters controlling the exploration-exploitation trade-off of Megh through a sensitivity analysis.

The rest of this paper is organised as follows. In Section II, we review the related work. In Section III, we depict the system model and build up the mathematical formulation to calculate costs of energy consumption and SLA violation. We introduce the problem of cost-optimal live migration as a reinforcement learning problem and formulate it mathematically in Section IV. Following that in Section V, we propose our novel algorithm Megh to solve it online. In Section VI, we elaborate the detailed experimental set-up and also evaluate the performance of Megh. We discuss the future research directions and conclude the paper in Section VII.

II. Related Works

While Megh tries to take *energy and performance efficient decisions for resource management* during live VM migrations, our methodology is based on *reinforcement learning*. Here, we review the related works in these two areas.

Dynamic VM Consolidation: A profitable strategy for Cloud vendors is the dynamic consolidation of underutilized virtual machines to fewer physical servers to save hardware, to reduce energy consumption [13] and to eliminate hotspots [14]. Due to the dynamic nature of Cloud workloads, there have been many studies in the field to investigate an optimal dynamic VM provisioning plan. One key requirement of dynamic VM consolidation is to pack VMs tightly while preserving SLAs. Mann et al. [15] recently presented an extensive survey of the problem models and optimization algorithms. Wang et al. [16] consider the dynamic network bandwidth demand for real workloads and model the VM consolidation into a Stochastic Bin Packing problem. Song et al. [17] similarly applied a variant of the relaxed on-line bin packing model, which was shown to work well on a small-scale cluster. Maguluri et al. [18] further modelled VM consolidation using a stochastic model where jobs arrive according to a stochastic process, and described MaxWeight algorithms, a family of frame-based non-preemptive VM configuration policies to improve overall throughput. Compared to existing models and algorithms, MEGH makes no *a priori* assumption on the workload arriving pattern or load distribution, which may be adapted to various scenarios while requiring a small number of migration requests and thus having little impact on running workloads.

In the existing literature, the Minimum Migration Time (MMT) family of algorithms [6], [7] function without any assumption on the workload model like Megh and perform in real-time. Due to this general structure and online mode of operation, we have compared Megh's performance with them. These algorithms are heuristics designed for energy and performance efficient dynamic consolidation of VMs in Clouds. They start migrating a VM when its utilization crosses a certain threshold. The threshold can be fixed (for THR-MMT) or determined adaptively (for IQR-MMT, MAD-MMT, LR-MMT and LRR-MMT) from the summary statistics of workloads' history. The VM is migrated to a different host such that the migration time is minimum. These methods are greedy heuristics that suffer from high variation and instability like other heuristic-based algorithms, while Megh, being a learning algorithm, does not.

Reinforcement Learning Algorithms: Reinforcement learning (RL) [8] is a paradigm of machine learning where an agent aims at taking optimal decisions by developing an understanding of the constantly evolving environment around it. *Markov decision processes* (MDP) [19] are a model for RL. MDPs assume that it is sufficient to remember the present state of the system to decide the next decision or action, while rewards of state-action pairs carry the relevant information of system's history. Thus, the agent tries to take a policy or a sequence of decisions that will maximize the cumulative sum of rewards acquired by the agent.

[20], [21] apply Q-learning [22] for energy efficient management of cloud resources. [23] uses it for automatic reconfiguration of resource sharing VMs. Q-learning is an offline algorithm. We have to go through computationally expensive training periods of a few hundred iterations before using it in an online setup. But there is no reliable guarantee on the optimality of Q-learning for online learning setup for any approximated value function [24]. As our problem has an online setup and it may face a sudden variance in the workload on which it is trained due to change in user base or their applications, Q-learning has a high probability to break down or perform sub-optimally.We have done a comparative performance analysis with respect to Q-learning. We omit an elaborate description of that in this article due to Q-learning's dependence on offline training and presence of a recent, on-line approach called MadVM.

MadVM [12] models the energy-efficient dynamic resource management of VMs in a data center as an approximate MDP. It assumes no prior knowledge of workload and uses value iteration [25] algorithm to solve the problem. At each step, MadVM tries to select decisions that simultaneously maximize the expected cumulative rewards of each of the VMs. Their approach is indirect as they do not try to optimize directly over a policy space but rather rely exclusively on value function approximation, that hopefully returns a near-optimal policy. Due to the combinatorially large state space of the problem, they also face the curse of dimensionality of RL approach. Thus, MadVM connects the policy space and the value functions through a key state selection procedure. This dimensionality reduction mechanism, however, is computationally expensive. MadVM tries to simultaneously optimize the utility functions of each of the VMs. Thus, it has to bookkeep transition functions and update key states for each of them. This makes MadVM poorly scalable for real-time applications.

Unlike critic based MadVM, Megh relies on the *actor-critic* [26] approach of RL. The actor tries to estimate the policy as an incremental functional approximation problem. The critic leverages this estimated policy for approximating and updating value function using samples collected as-it-goes. This feedback ensures better convergence property and stability. In our paper, we use such an off-policy actor-critic framework of least-square policy iteration (LSPI) algorithm [27] as a skeleton. We utilize the projection based dimensionality reduction techniques and sparsity-based improved data structures described in Section V to construct our real-time learner Megh.

III. The Cloud Data Centre: System and Costs

In the following subsections, we describe the system model of a data centre used by Megh and formulate cost models for energy consumption and SLA violations.

A. System Model

In IaaS environments Cloud providers serve the users with virtualized computing resources over the Internet. In order to model such a system, we consider a data centre consisting of M heterogeneous physical machines (PMs) or hosts. Each of these PMs is characterized on the basis of the number of CPUs, the number of cores, the amount of RAM and the network bandwidth. Here, the performance of a CPU is defined in Millions Instructions Per Second (MIPS). In our paper, we consider all of the CPUs belonging to the same PM as a single-core CPU with the cumulative MIPS performance of all of them. Now, independent users submit requests for provisioning of computing resources to the Cloud and are assigned to N heterogeneous VMs hosted by M PMs. Each of the VMs is allocated CPU performance, memory size, RAM, and network bandwidth as per the users' requirements. We assume no *a priori* knowledge of the applications, workload dynamics and the time of provisioning of VMs. This allows us to deal with both general-purpose and specialised setting of mixed workloads with uncertain dynamics that utilize the resources of a PM concurrently.

Our learning algorithm, Megh, is implemented as a part of the global resource manager of the Cloud. This global manager acts as an interface between users' workloads and requirements, and the virtualization layer. The Virtual Machine Managers (VMMs) operating at each of the physical nodes act as the continuous monitoring systems. They send the workload dynamics of each VM and the resources utilized by them to the global manager. The global manager, which is acting as the learning agent in Megh, accumulates the information and allocates the resources in such a way that will minimize the energy consumption as well as the SLA violation. Following this, the decision is sent to VMMs as a resource map and VMs are migrated and consolidated accordingly. Megh may move the VMs allocated in an underloaded PM to another PM with potential capacity and put the first PM down to sleep. Similarly, if a PM gets overloaded, some of the VMs operating on it are migrated to another PM such that the expenditure for energy consumption and SLA violation remains minimal.

B. Energy Consumption Cost

Energy consumption cost of the Cloud data center can be considered as a function of time $C_p(t)$, such that

$$C_{\mathrm{p}}(t) = c_{\mathrm{p}} \int_0^t P(\theta) d\theta, \qquad \forall t \geq 0. \tag{1}$$

Here, c_{p} denotes the cost of consuming 1 Watt of power for 1 second. It is a fixed constant according to the place where the data center is built up, whereas $P(\theta)$ is the function representing the amount of power (in Watts) consumed by the data center at time θ (in seconds). This function does not only depend on the workload dynamics of VMs but also on the CPU

performance, memory size, disk storage and cooling system of the PMs installed in the data center [28]. Following the works by [7], we leverage the power consumption data provided by the SPECpower benchmark [29], [30] rather than moving our focus to precisely modelling $P(\theta)$. This benchmark provides energy consumption level y for a collection of servers with different CPU architectures under a workload of $x\%$ working on its CPU, as shown later in Table I. Now, if we assume the Cloud management system extracts the workload dynamics at a certain interval, say $\tau > 0$, we can model the cost of energy consumption up to time t as

$$C_{\mathrm{p}}(T) = c_{\mathrm{p}} \sum_{k=0}^{T} \sum_{i=1}^{M} y_i(k\tau)\tau, \qquad \forall T \geq 0 \tag{2}$$

where, $T \triangleq \lceil \frac{t}{\tau} \rceil$ represents the discretized version of time t, $y_i(k\tau)$ is the power consumed by the i^{th} PM at time $k\tau$ and M denotes the total number of PMs operating in the data center.

C. SLA Violation Cost

Though energy consumption covers the major part of the Cloud provider's expenditure, Quality of Service (QoS) provided by the Cloud is a concern from the user's side. Specifically, QoS is negotiated using a legal agreement between the user and the Cloud provider, called Service Level Agreement (SLA). SLAs provided by companies like Amazon, Microsoft and Google.confirm that service providers promise to pay the user certain monetary penalties if the QoS degrades below certain levels. We also observe this QoS is defined as the uptime percentage of the user. It means the percentage of total access time the user can utilize the Cloud services without any interruption. Though some of the Cloud providers do not consider any continuous downtime below 5 minutes as a degradation of QoS to provide the system privilege, in this paper, we have considered the exact downtime without such bias. Thus, the total SLA violation cost of a Cloud data center with M PMs and N VMs can be expressed as,

$$C_{\mathrm{v}}(t) = \sum_{j=1}^{N} c_{\mathrm{v}}^{j}(t), \qquad \forall t \geq 0 \tag{3}$$

where, $c_{\mathrm{v}}^{j}(t)$ is the SLA violation cost for VM j until time t. Following the aforementioned convention, it can be defined as

$$c_{\mathrm{v}}^{j}(t) = \begin{cases} cv_1, & \text{if downtime percentage up to } t \\ & \in (0.05\%, 0.10\%] \\ cv_2, & \text{if downtime percentage up to } t \\ & > 0.10\% \\ 0, & \text{otherwise} \end{cases}$$

as the system model considers each VM is used to virtually assign computing resources to each of the users.

As we try to allocate and manage the resources by migrating the VMs from one machine to another, we face two scenarios of QoS degradation. In the first case, when one or multiple VMs are allocated to a PM, it faces a sudden rise of workload. This PM gets overloaded. Overloading happens when VMs try to use more resources than the capacity of the host PM. This creates a scenario where we need to migrate VMs from that host to another. But due to discretized time of observations by the global learning agent and the inherent delay of the host system to react and adapt to the scenario, it takes some time before the migration decision is taken and executed. During this period the VMs working on that host remain suspended or their performance degrades substantially. This phenomenon introduces a downtime in each of the VMs working on that host and is termed as the overloading time. In this paper, we denote *overloading time* of host PM i at time t as $T_{\mathrm{o}_{it}}$. $T_{\mathrm{o}_{it}}$ represents the total time during which the host i has experienced the utilization of greater than $\beta\%$ leading to overloading. If the active time $T_{\mathrm{a}_{it}}$ of the host i is defined as the total time for which it is serving the users, we can define the *percentage of overloading time* as

$$O^{i}(t) \triangleq \frac{T_{\mathrm{o}_{it}}}{T_{\mathrm{a}_{it}}}. \tag{4}$$

Though the live migration transfers a VM from a host PM to another destination PM without suspending the running application, it still faces a downtime. The *migration time* is defined as the time required to copy all the pages of a VM from its present host memory to the destination memory. If M_{jt} is the amount of memory used by VM j right before initiating the migration at time t, and, B_{jt} is the available bandwidth of the network, expected migration time of VM j is expressed as $TM_{jt} \triangleq \frac{M_{jt}}{B_{jt}}$. Now, the downtime of VM j is estimated as the time for which the estimated CPU utilization $\widehat{u_j}(t)$ will be less than a certain threshold. This threshold can be defined as $\alpha\% > 0$ of the workload $u_j(t)$ trying to act on the VM from the user side. Thus, we can estimate the live migration downtime of VM j at time t as

$$T_{\mathrm{d}_{jt}} \triangleq \int_{t}^{t+TM_{jt}} \mathbb{1}\left(\widehat{u_j}(\theta) < \alpha u_j(\theta)\right) d\theta,$$

where $u_j(t)$ is the CPU utilization by VM j at that moment and $\mathbb{1}$ is the indicator function defined as

$$\mathbb{1}\left(\widehat{u_j}(t) < \alpha u_j(t)\right) \triangleq \begin{cases} 1, & \widehat{u_j}(t) < \alpha u_j(t) \\ 0, & \texttt{otherwise} \end{cases}, \qquad \forall t \geq 0.$$

If $T_{\mathrm{r}_{jt}}$ is the total active time requested by the VM j till the time t, we can estimate the *percentage of live migration downtime* of VM j as

$$D^{j}(t) \triangleq \frac{T_{\mathrm{d}_{jt}}}{T_{\mathrm{r}_{jt}}}, \tag{5}$$

So the total downtime percentage for VM j up to time t is defined as the sum of its downtime and overlap of its operation time on a PM and overloading time of the corresponding host up to time t.Thus, Equation (4) and (5) give us a concrete mathematical model to calculate the SLA violation cost for each of the VMs.

IV. Live Migration as a Learning Problem

In this section, we formulate the problem of energy- and performance-efficient resource management during live migration of VMs as a reinforcement learning problem.

Let us consider a Cloud data center with M PMs. Each of the PMs has homogeneous CPU capacity h. Each of the VMs is assigned to each of the users on the basis of their requests.

Though the workloads and requirements of users may differ, the maximum CPU capacity that can be allocated to a VM is a constant, say v. Thus, under the worst case scenario, when each of the VMs will ask for maximum CPU capacity, the maximum number of VMs n that can be allocated to a single host is $\frac{h}{v}$. Furthermore, the total number of VMs N that can be allocated to the data center or the maximum number of users that the Cloud can handle at any instance is Mn. The VMs are accessed by a large volume of users with diverse requirements and applications, and the dynamics of these workloads are also uncertain. This may cause a sudden change in workloads of one or multiple VMs and consequently overloading of hosts. Then one of the VMs working on the overloaded host has to be migrated to another destination PM such that cost for energy consumption and SLA violation remains minimal. But while doing so the system has to decide which VM to move to which destination host and when to start moving, so that the penalty will be minimum ensuring maximum profit of Cloud provider and also maximum QoS for users. In general, this problem of resource allocation during VM migration is NP-hard.

We can model this process of live migration as a Markov Decision Process [19]. In this model, the *state space* S is Cartesian product of $\mathcal{C}$, the set of all configurations of the VMs on the PMs, and W, the workloads operating on the VMs at any instance. At a certain instance, W is an array of N elements, where an element represents the CPU usage of a VM at that instance. Since W varies continuously, it makes the state space infinite dimensional and introduces uncertainty in state transitions. The *action space* A corresponds to migration of any of the VMs from one host to another depending on the operating workloads. Each action is represented by a pair (j, k), where j is the migrating VM, and k is the destination host. In order to capture the uncertainty of workloads, we define *transition function* $f : S \times A \to \mathcal{P}(S)$, where $\mathcal{P}$ is a probability measure over state space. Given the present state and an action, f returns the probability to reach another state. But in our problem, it is not known *a priori* and has to be learned. *The cost* of changing a configuration s_{t-1} of VMs to another configuration s_t is given by

$$C(s_{t-1}, s_t) = \triangle C_p(s_{t-1}, s_t) + \triangle C_v(s_{t-1}, s_t), t \in [1, T] \quad (6)$$

where, $\triangle C_p(s_{t-1}, s_t)$ and $\triangle C_v(s_{t-1}, s_t)$ are the costs of energy consumption and SLA violation in the interval $(t-1, t]$. Here, C_p and C_v are defined by Equations (2) and (3) respectively. We can observe $\triangle C_p(s_{t-1}, s_t)$ is always positive as the system will always consume some energy whether or not any migration happens, whereas $\triangle C_v(s_{t-1}, s_t) \geq 0$. The equality holds if and only if there is no SLA violation in that interval.

Thus, the problem reduces to finding the sequence of configurations that minimizes the sum of future per-stage costs. Unlike MadVM that assumes an average cost structure and computationally considers the effect of a migration is limited to a fixed future time horizon, we assume an infinite horizon [8] structure. This means an action will affect all the future states and actions of the system. But it makes the cumulative sum of future per-stage costs infinite. In order to circumvent this problem a *discount factor* $\gamma \in [0, 1)$ is introduced. Mathematically, it makes the cumulative sum of per-stage costs convergent. Physically, it means with each passing instance the effect of a past action decays by γ. Thus, it let the system give more importance to immediate costs than to costs distant in the future, which follows a practical intuition. Now, the problem translates into finding the sequence of configurations that minimizes a *discounted cumulative cost*. Under Markov assumption, a configuration change depends on its present state only. Given the current configuration and workloads, i.e the current state s_t, a policy $\pi\colon S \to A$ determines the next decision a_t. We define the *cost-to-go* function V^π for a policy π as

$$V^\pi(s) \triangleq \mathbb{E}_f \left[\sum_{t=1}^{\infty} \gamma^{t-1} C(s_{t-1}, s_t) \right] \quad (7)$$

such that the initial state $s_0 = s$, and s_t is the state reached from state s_{t-1} through an action $\pi(s_{t-1})$). The value of $V^\pi(s)$ represents the expected cumulative cost for following the policy π from the current configuration s. Thus, $V^\pi(s)$ allows us to optimize the long-term effect of migration decisions, unlike greedy MMT algorithms that try to minimize the present cost only. Let $\mathcal{U}$ be the set of all policies for the given set of VMs on the cluster of PMs. Now, our problem can be phrased as

$$\pi^* \triangleq \underset{\pi \in \mathcal{U}}{\operatorname{argmin}} V^\pi(s_0). \quad (8)$$

i.e, to find the optimal policy π^* that minimizes the expected cumulative cost.

V. MEGH: LEARN TO MIGRATE AS-YOU-GO

Depending on the cost model developed in Section III and the problem formulation in Section IV, here we propose an algorithm, Megh, to solve the problem of live migration. This problem consists of three basic questions: *when* to start migrating the VM, *which* VM to migrate, and *where* i.e, to which PM to migrate it.

Megh answers these questions by solving the minimization problem of Equation (8). This equation shows that optimal decision making is analogous to finding out the optimal function π^* that minimizes the cost-to-go function. We can consider this as a functional approximation problem over the space $\mathcal{U}$. In order to do so, we begin with an initial guess of the policy π_0. Following that as we gain more information about the configuration of VMs and also the dynamics of workload on them, we improve our approximation consequently such that it keeps the current estimation of cost-to-go function minimum. In RL literature, this strategy is known as *policy iteration* [8].

If the stochastic nature of workload W i.e, the transition function f is known *a priori*, we can apply Bellman's dynamic programming technique [31] to update the estimate of cost-to-go function at every time step using the following formula,

$$V^{\overline{\pi}_t}(s) = \mathbb{E}_f \left[C(s, s') + \gamma V^{\overline{\pi}_{t-1}}(s') \right]. \quad (9)$$

Thus, the updated policy would be $\pi_t = \operatorname{argmin}_{\overline{\pi}_t \in \mathcal{U}} V^{\overline{\pi}_t}(s)$. The algorithm terminates when there is no or very small change in the policy. Policy iteration has strong optimality and convergence properties [32].

In this problem policy iteration suffers from two main issues. Firstly, to update the cost-to-go function in Equation (9) and to find the optimal policy, we have to search through the whole state-action space. But in this case, the state space which consists of all possible configurations of VMs on all the PMs is combinatorially large. This high dimensionality of state space S makes the computation expensive and almost impossible to perform in real-time. This exponential blow-up due to huge state space is called the *curse of dimensionality* [32]. Besides this, it is also impossible to calculate the expectation in Equation (9). Because of the stochastic nature of workload, its correlation with VM configurations and their transitions are not known *a priori*. Also to conserve the robustness and universality of our method, we cannot restrict this workload dynamics to a specific model. Indeed that would narrow down the applications and the hardware architectures the algorithm can deal with.

In this work, we propose two methodologies to solve these issues. In order to solve the curse of dimensionality, which includes searching over a large state-action space to find out the estimate of the cost-to-go function, we project the state-action space to a $d = N \times M$ dimensional space X. X can be spanned with d basis vectors $\{\phi_{jk}\}_{j=0,k=0}^{N,M}$. Each of the basis ϕ_{jk} corresponds to an action (j, k) such that thejk^{th} element of it is one, and all others are zero. All the actions or configuration changes in the Cloud can be represented using these basis vectors or linear combinations of them. The basic rationale behind this projection is when transitioning from one state s to another state s' happens, we can reach only a certain part of the state space which is one action away from the present state s. Thus rather than searching over the whole state space in each and every step it is logical to search in a subspace X that contains all the states s' reachable from s by actions ϕ_{jk} or linear combinations of them. Now, we approximate the cost-to-go function as $V(s_{t+1}) = \theta^T \phi_{a_t}$, where $a_t = \pi_t(s_t)$ is the action taken at time t. Thus the whole state-action space of VM configurations, which is combinatorially explosive, is now projected down to a polynomial dimensional vector space with a sparse basis. This let us update the cost-to-go function effectively in real-time.

Theorem 1. *There exists a unique θ which approximates the value function as $V(s) = \theta^T \phi_{\pi(s)}$.*

Proof. See Appendix A for the details. □

Still the expectation of the cost-to-go function is not computable due to lack of prior knowledge of workload dynamics and how it affects the VM configurations and their transitions. In order to capture this notion, we create a stochastic operator T. It accumulates the possibility of using an action to move to another configuration from the present one depending on the nature of workload and the changes caused by them. In this work, we start with $T_0 = \frac{1}{\delta}\mathbb{I}_d$, where δ is a large positive number and $\mathbb{I}_d$ is an identity matrix of order d. Here, we have considered δ as d. It implies that initially, there is no bias and the system can migrate any of the VMs to any of the PMs equally probably. As the system extract information of the workload and VM configurations at each time step t, it decides

Algorithm 1

```
function MEGH(S, A, γ, ϵ, Temp_0)
    Initialize δ ← d, B_0 ← (1/δ) I_{d×d}, φ_0 ← 0_d,
    θ_0 ← 0_d, π(s_0) ← 0_d, z_0 ← 0_d, C_0 ← 0
    while t ≥ 1 do
        a_t ← argmax_{a∈A}  π_t(s_t)
        Take action a_t.
        Observe state s_{t+1}.
        C_{t+1} ← Calculate cost using Equation (6).
        B_{t+1} = T_{t+1}^{-1} update using Equation (10).
        z_{t+1} ← z_t + φ_{a_t} C_{t+1}
        θ_{t+1} ← B_{t+1} z_{t+1}
        π(s_{t+1}) ← PolicyCalculator(φ_{a_t}, θ_{t+1})
    end while
end function
```

an action a_t according to the policy π_t. Using this information, we can update the operator T as

$$T_{t+1} = T_t + \phi_{a_t}\left[\phi_{a_t} - \gamma\phi_{\pi_t(s_{t+1})}\right]^T. \tag{10}$$

where, $\phi_{\pi_t(s_{t+1})}$ represents the probable action at time $t+1$, if the policy π_t is followed at the next time instance. Thus, Equation (10) captures the effect of present state and action and its influence in future action with a discount γ.

In Megh, we plug in these two schemes of polynomial size projection space X and incremental update of the operator T to Least-Square Policy Iteration algorithm [27]. This algorithm acts in the actor-critic format [26] such that the algorithm first tries to find out an estimation of cost-to-go function by least-square estimation in the actor format and then to update the policy such that it maximizes the estimate in the critic format. The pseudo-code of Megh is depicted in Algorithm 1.

Theorem 2. *If for any policy π, there exists a vector θ such that $V_\pi(s) = \theta^T \phi_{\pi(s)}$ for any configuration s, Algorithm 1 will converge to an optimal policy.*

Proof. See Appendix B for the details. □

Algorithm 2

```
function POLICYCALCULATOR(φ_{a_t}, θ_{t+1})
    Temp_{t+1} ← Temp_t exp(−ϵ)
    for all i = 1, ..., d do
        Q(s_{t+1}, a_i) ← φ_{a_i}^T θ_{t+1}
    end for
    MIN_Q ← min_a Q(s_{t+1})
    for all i = 1, ..., d do
        π(s_{t+1})_i ← exp[ (−Q(s_{t+1}, a_i) + MIN_Q) / Temp_{t+1} ]
    end for
end function
```

Instead of greedily choosing the action with maximum $V^{\pi_t}(s_{t+1})$, we have used Boltzmann exploration. It is shown in Algorithm 2. This technique compares the goodness of an action with respect to the others and lets the algorithm explore more. Here, we have started with an initial temperature value $Temp_0$ and decay it consequently with a factor $\exp(-\epsilon)$. Initially, the large $Temp$ means rather than choosing the maximum greedily

Table I
POWER CONSUMPTION OF SERVERS IN WATTS FOR DIFFERENT LEVEL OF WORKLOAD [29], [30]

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

Table II
PERFORMANCE EVALUATION FOR PLANETLAB

Algorithms	THR-MMT	IQR-MMT	MAD-MMT	LR-MMT	LRR-MMT	Megh
Total cost (Dollar)	1347	1504	1367	1392	1392	1155
#VM migrations	325299	444624	331304	324079	324079	2309
#Active hosts	666	684	682	692	692	203
Execution time (ms)	2016	3077	2226	1924	2080	1426

Table III
PERFORMANCE EVALUATION FOR GOOGLE CLUSTER

Algortihm	THR-MMT	IQR-MMT	MAD-MMT	LR-MMT	LRR-MMT	Megh
Total cost (Dollar)	706	708	708	710	710	688
#VM migrations	299352	262185	266706	233172	233172	3104
#Active host	82	72	73	59	59	194
Execution time (ms)	2887	4030	4000	3889	3923	1945

it is trying to explore more. But with time as $Temp$ decreases, it turns into as same as the greedy selection of the maximum.

A. Managing the Complexity Bottleneck

Algorithm 1 has space complexity of $O(d^2)$ and time complexity of $O(d^3)$. Though it is computationally cheaper and faster than the actual combinatorially explosive problem scenario, still it can be slow enough for a real-time system operating over a large number of VMs and PMs. The bottleneck in this whole process is storing the $d \times d$ matrix B and finding out the inverse of the operator T to update B at each time-step, as shown in Line 9 in Algorithm 1. If we use the Gauss-Jordan elimination process [33] provided by linear algebra packages [34], it costs a time complexity of $O(d^3)$. In order to find out the inverse incrementally at every step, we use Sherman-Morrison Formula [35] on Equation (10) given by,

$$B_{t+1} = B_t - \frac{B_t \phi_{a_t} \left[\phi_{a_t} - \gamma \phi_{\pi_t(s_{t+1})}\right]^T B_t}{1 + \left[\phi_{a_t} - \gamma \phi_{\pi_t(s_{t+1})}\right]^T B_t \phi_{a_t}}. \tag{11}$$

It reduces the time complexity of every step to $O(d^2)$.

Now, we can reduce this complexity by leveraging the sparsity of the basis vectors ϕ_{a_i}'s. Since all the zero entries are redundant in the calculation of product, we can store only the non-zero entries of the matrix B and vector ϕ_{a_i} as a triplet (row number, column number, value). This reduces the initial storing size to $O(d)$. Because during initialization we start with a diagonal matrix of order d and d basis vectors each with single non-zero entry. The storing size increases at each step as per the number of migrations happened during the interval. Thus the multiplication in Equation (11) turns into simply choosing the non-zero terms in B_t according to the 1 entries in ϕ_{a_i}'s involved in the calculation and then adding or subtracting them. It reduces the time complexity of Line 9 in Algorithm 1 to $O(\#m)$, where $\#m$ is the number of migrations per step. The aforementioned use of online update and inversion technique, and also leveraging the sparsity of the basis vector reduces both the space and time complexity of Megh substantially. These techniques give Megh the speed-up to be a real-time system while keeping its structure and learn-as-you-go strategy intact.

VI. PERFORMANCE EVALUATION

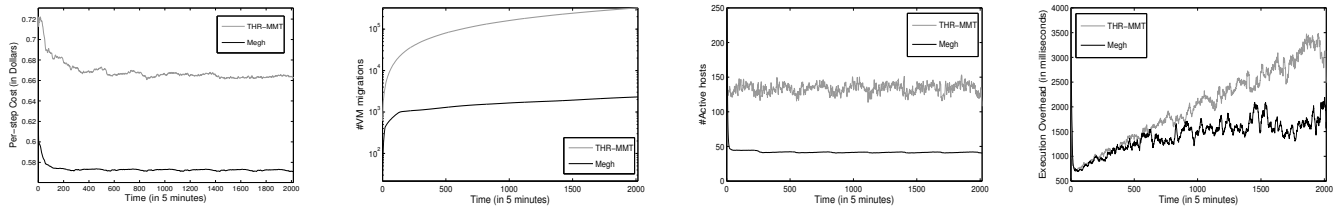
A. Experimental Setup

We perform our experiments using the CloudSim toolkit [9] as the simulation platform. In the power model, we use the standard price of the local power providers, 0.18675, SGD/kWh to calculate the energy consumption cost. We assume that the user has to pay 1.2 SGD per hour for using a VM instance. Though it is a bit costlier than reality, it does not harm the analysis. Following the model mentioned in Section III-C, we also assume that Cloud providers would pay back 16.7% and 33.3% of user's money depending on whether the performance degradation is less than or greater than 0.10%. In our set-up, we have considered $\beta = 70\%$ as the overloading threshold of the PMs and $\alpha = 30\%$ for the minimum CPU usage threshold by VMs during migration. The experiments are conducted on a server with AMD Opteron(TM) Processor 6272 CPU @ 2.1GHz and 128 GB memory. MMT algorithms are tested using the code embedded with the CloudSim toolkit, whereas Megh and MadVM are implemented and embedded in the CloudSim framework using Java. For both of them, the value of γ is 0.5. It let us allow 50:50 importance of both new and old information. $Temp_0$ and ϵ are set to 3 and 0.01 respectively for the experiments in Section VI-C and VI-D.We explicate such choice of parameters in Section VI-E. At each time-step, we allow a maximum 2% of VMs to be migrated by Megh.

B. Dataset and Workload

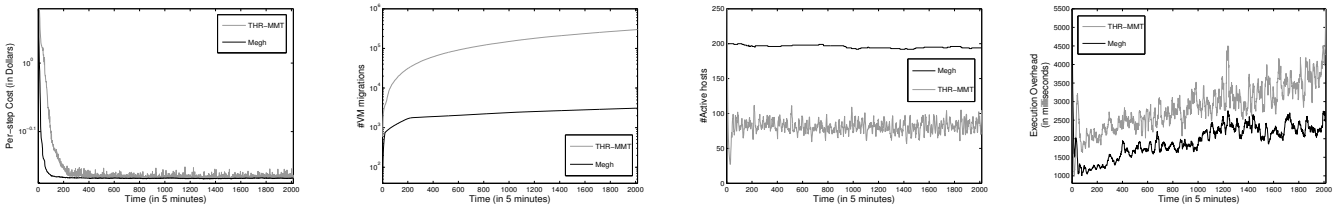
PlanetLab Dataset: CloudSim contains workloads extracted from the CoMoN project which was a monitoring infrastructure for PlanetLab [10]. Each of the workloads consists of CPU utilization data extracted at a regular interval of 5 minutes for a span of 7 days. Figure 1(a) shows the statistical nature of the workload and depicts inherent uncertainty in its dynamics. The workloads are working on a set of 800 heterogeneous physical machines (PMs). Half of these PMs are HP ProLiant ML110 G4 servers and the other half are HP ProLiant ML110 G5 servers. The power consumption characteristics of these two servers is obtained from SPECbenchmark and is shown in Table I. Though they follow different energy consumption models, each of them has a dual-core processor with 4GB RAM and are provided with 1 Gbps network bandwidth. There are a total of 1052 applications are running on this system. Each of the applications are allocated on a VM with 1 vcpu, 0.5-2.5GB RAM, 0.5-2.5 MIPS and 100 Mbps bandwidth.

Google Cluster Dataset: The Google Cluster trace represents dynamic tasks running on Google's Hadoop MapReduce clusters with 12,500 heterogeneous machines. It contains continuous information of 29 days with event records and sampled resource usage at an interval of 5 minutes. We select 500 machines as physical machines and the tasks scheduled on those machines as virtual machine workloads. We create 2000 virtual machines with each running an individual task to completion and switching to another. Unlike PlanetLab where all of the workloads are together varying intensely, the Google



(a) Per-step cost (in SGD) (b) Number of VM migrations (c) Number of active hosts (d) Execution time

Figure 2. Performance of Megh, THR-MMT and LRR-MMT algorithms for PlanetLab dataset



(a) Per-step cost (in SGD) (b) Number of VM migrations (c) Number of active hosts (d) Execution time

Figure 3. Performance of Megh, THR-MMT and LRR-MMT algorithms for Google Cluster dataset.

Cluster trace has tasks with varying durations, starting times, and obfuscated resource usages as shown in Figure 1(b).

PlanetLab is a huge geo-distributed computing platform consisting of hundreds of sites and more than one thousand nodes. It is hosted by organisations across the world. Users can access the computing resources by deploying applications to a subset of the nodes in the form of VMs. The trace is collected from PlanetLab to track the CPU usage of each VM's workload. The result can represent the typical workload running in a cloud environment. While the PlanetLab trace is mainly related to academic computation tasks, the Google Cluster trace records the events in Google's cluster. Google's trace can show the characteristics of workloads running in the widely available real cloud system. Evaluating our algorithm with the traces from both the community and the industry validates universality and robustness of our algorithm.

C. Comparative Performance Analysis

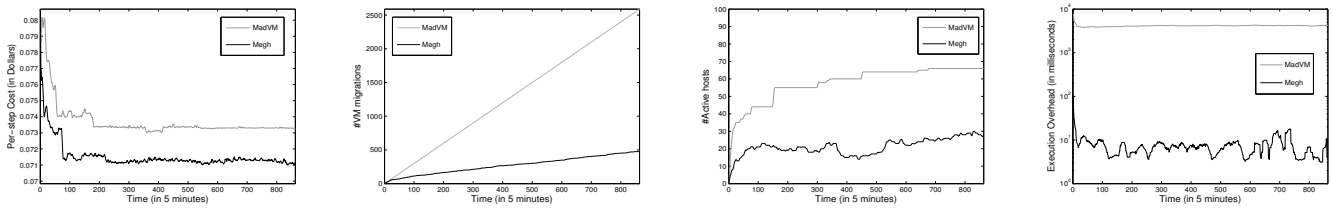
Megh vs MMT algorithms: Table II depicts the performance of Megh and the MMT algorithms on a week-long trace of PlanetLab. Table III summarizes the performance of aforementioned algorithms for the Google Cluster dataset. Total cost of operation of the data center (in SGD) that is obtained by adding the power consumption cost and SLA violation cost, the number of VM migrations, average number of active hosts and execution time (in milliseconds) of each iteration of the algorithms are used as the performance measures of these algorithms. As THR-MMT performs the best among the MMT algorithms, we show a comparison of Megh with THR-MMT in Figures 2 and 3.

We observe from Tables II and III after 7 days of operation Megh reduces the expenditure by 14.25% for PlanetLab and 2.5% for Google Cluster with respect to that of THR-MMT. Figures 2(a) and 3(a) show the per-step operation cost for Megh not only converges faster than the contending algorithms but also has less variance for both PlanetLab and Google. Here, the per-step operation cost includes both the energy consumption cost and the SLA violation cost in the 5 minutes interval between two observations. Due to the learn-as-you-go policy, Megh takes around 100 time-steps before reaching the almost stable cost per-step. We do not observe such a fast convergence for THR-MMT. Being a greedy heuristics, THR-MMT still faces high variance and instability even after initial convergence. It validates the robustness and stability of Megh for optimal resource management for a diverse set of workloads with respect to other heuristics.

In order to measure the performance of the system and its QoS, we use the number of VM migration as another metric. In our experiments, we consider that during the course of migration the CPU capacity allocated to a VM on the destination node is same as that of the present host. This means that each migration may cause some SLA violation; therefore, it is crucial to minimize the number of VM migrations. The total number of VM migrations for THR-MMT is almost 140 times and 97 times more than that of Megh for PlanetLab and Google respectively. Figures 2(b) and 3(b) report the evolution of the cumulative number of VM migrations over the span of 7 days. As the total number of VM migrations up to an instance for Megh is much less than that of the THR-MMT, it shows at any instance Megh performs significantly better.

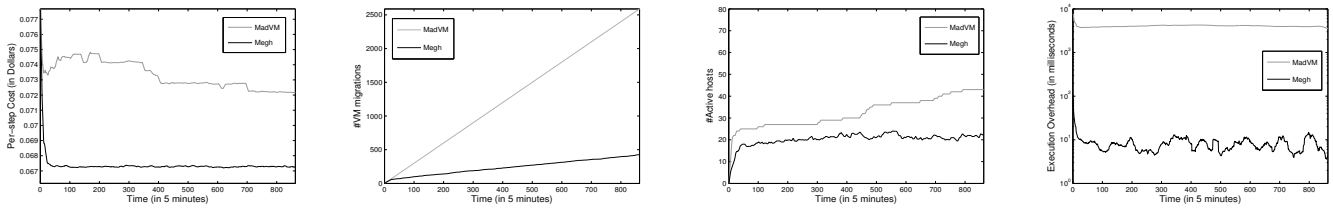
Decreasing the number of active hosts also decreases the power consumption. Thus, the number of active hosts is also used as a performance metric for resource management algorithms. Though reducing the number of active hosts is the approach taken by VM consolidation algorithms, it may prove not to be a perfect metric. Because keeping a larger number of hosts at very low utilization level can cause less power consumption than keeping a few hosts at very high utilization level. We can see this dilemma from Figures 2(c) and 3(c). For PlanetLab, Megh keeps fewer hosts active than other MMT algorithms, whereas for Google it keeps more active VMs while incurring the least per-step cost for both datasets.

While the results above establish Megh's effectiveness to solve the live migration decisions with less expenditure and better QoS, it has to fulfil another criterion to be a real-time system: a small execution time. From Figures 2(d) and 3(d), we observe Megh is running faster than that of the heuristic based online algorithms. As shown in Tables II and III, Megh speeds up the decision making by 1.41 and 1.48 times with respect



(a) Per-step cost (in SGD) (b) Number of VM migrations (c) Number of active hosts (d) Execution time

Figure 4. Performance of Megh and MadVM for a dataset of 100 PMs and 150 VMs extracted from PlanetLab trace.



(a) Per-step cost (in SGD) (b) Number of VM migrations (c) Number of active hosts (d) Execution time

Figure 5. Performance of Megh and MadVM for a dataset of 100 PMs and 150 VMs extracted from Google Cluster trace.

to THR-MMT for PlanetLab and Google respectively. Since migration time of a VM is in the order of a few seconds, speed up of Megh with respect to the state-of-the-art can help the system to make decisions and to execute them with significantly less overhead or downtime to the process of migration. This, in turn, improves the QoS of the system too. This empirically proves the efficiency of Megh not only as an effective learning algorithm but also as an eligible real-time resource management system in Clouds.

Megh vs MadVM: MadVM fails to scale-up for the complete PlanetLab or Google Cluster in our experimental facilities. Thus, in order to compare the performance of our algorithm with MadVM, we have chosen two random sets of 150 workloads running on 100 PMs for 3 days from PlanetLab and Google Cluster traces. At the beginning, all these workloads are allocated uniformly at random to each of the PMs, such that there is no initial bias for the learning and the robustness of both the algorithms can be tested. The 50:50 ratio of two type of servers is still maintained.

From Figures 4(a) and 5(a), we see Megh incurs less cost (4.3% and 8.8%) than MadVM at every time step. Figures 4(b) and 5(b) show Megh causes significantly less (5.5 and 6.1 times) migrations than MadVM. Figures 4(c) and 5(c) depict at every time step MadVM (average $\sim$58 and 34) keeps more hosts active than our method (average $\sim$21 and 20). But the main factor where MadVM stumbles is the execution time. It takes on an average 4143ms and 4057ms to execute a single iteration for a system of 100 PMs and 150 VMs, which is almost the same as the migration time of a VM of 0.5 GB RAM in the PlanetLab set-up. As the RL algorithms face the curse of dimensionality and have a huge transition matrix for bookkeeping at each time step, it makes RL algorithms slower for a real-time system. Though authors of MadVM tries to handle such scenario, Figures 4(d) and 5(d) depict its inability to scale in real-time for large data centers. But since Megh leverages the sparsity-based projection technique, along with the specialised data structure, it takes the same migration decision in approximately 7ms and 8ms respectively for PlanetLab and Google systems. Thus, the experiments show though Megh uses the RL approach for learning the workload dynamics and to make migration decisions, it is significantly more efficient and faster than the latest state-of-art RL algorithm for dynamic VM management.

D. Scalability Analysis

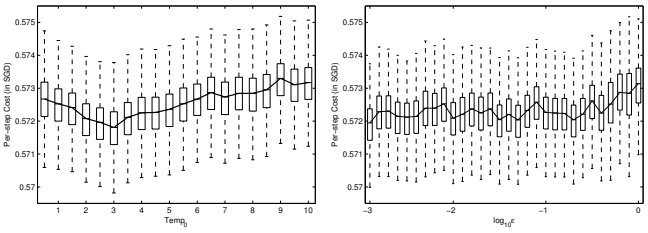
Scalability is an important issue that an algorithm has to achieve in order to perform for a large Cloud data center. Thus, we show a comparative analysis of scalability of Megh and THR-MMT in Figures 7(a) and 7(b). In order to conduct such experiments, we have randomly chosen m and n number of PMs and VMs from the PlanetLab data. Here, both m and n can take values $\{100, 200, 300, 400, 500, 600, 700, 800\}$. For each value of m and n, we have done 25 such experiments with 25 randomly chosen set of PMs and VMs.

We observe from Figures 7(a) and 7(b) as the number of PMs and VMs increase, the execution time per step increases for both THR-MMT and Megh. With the increase of number of PMs and VMs, the decision making algorithm has to choose among larger set of options and has to face an increased uncertainty in workload dynamics. Thus, this increase in execution time is intuitive and natural. For Megh the rise in execution time is much smaller than that of THR-MMT. This significant difference in per-step execution time shows that Megh scales up better than THR-MMT. It makes Megh more effective as a real-time decision maker for large Clouds.

As MadVM is not scalable after 100$\sim$150 PMs, we cannot conduct such a comparative study with it.

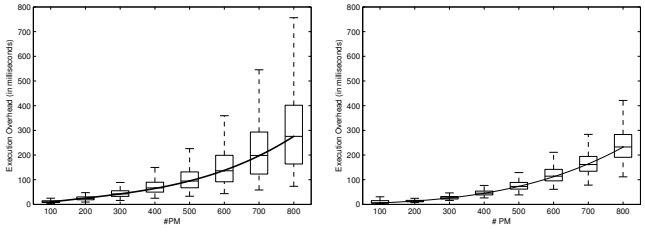
E. Parameter Sensitivity

$Temp_0$ and ϵ are used as parameters to tune the exploration-exploitation trade-off of Megh. We test and analyze Megh's performance on different values of the parameters. We vary $Temp_0$ from 0.5 to 10 with a granularity of 0.5 while keeping $\epsilon = 0.001$. We run experiments on 30 distinct values of ϵ, which belong to the interval $\left[10^{-3}, 10^{0}\right]$ and are at a logarithmic (base 10) distance of 0.1. In this case, $Temp_0$ is fixed to 1.



(a) $Temp_0$-sensitivity (b) ϵ-sensitivity

Figure 6. Sensitivity of per-step cost (in SGD) on $Temp_0$ and ϵ.



(a) THR-MMT (b) Megh

Figure 7. Scalability analysis of THR-MMT (left) and Megh (right).

For each value of $Temp_0$ and ϵ, Megh is tested 25 times on the PlanetLab dataset described in Section VI-B.

Figures 6(a) and 6(b) show boxplots of per-step cost (in SGD) of Megh for each of the values of the parameter. These boxplots depict the median and 90 percentile distribution of the per-step cost. We observe that the median cost decreases first as the $Temp_0$ increases. But as it crosses $Temp_0 = 3$, the cost rises. Though for ϵ this change in per-step cost is a bit sporadic, we see the variance and the median both reach a local minimum at $\epsilon = 0.001$.

Since use of $Temp$ in Algorithm 2 let Megh explore more rather than direct exploitation, increase in $Temp_0$ would increase the initial exploration. We see till $Temp_0 = 5$ this increase in exploration is decreasing the median cost. Because increased exploration stops agent from getting stuck at local minima and take decisions more globally. But after that point, we see the adverse effect of too much exploration. As $Temp_0$ increases after 5, the algorithm cannot benefit enough from exploitation. Thus, the curve instantiate the exploration-exploitation trade-off in case of Megh.

ϵ controls decay of $Temp_0$ with time. As $Temp_0$ decays, the exploratory nature turns dormant and exploitative nature begins to dominate. Thus, increase in ϵ would cause faster decay of $Temp$. Though we expect to see similar nature as that of the variation of $Temp_0$ but here we see a bit of sporadic nature where it is hard to detect a single tipping point for exploration-exploitation trade-off. Hence, we make our choice empirically from observation.

VII. Conclusion

In this paper, we address the problem of energy- and performance-efficient resource management during live migration of VMs in a data center. Cloud providers leverage live migration of VMs to reduce energy consumption and allocate resources efficiently in data centers. Each migration decision depends on three questions: when to move a VM, which VM to move and where, i.e, to which PM to move it? The uncertain, dynamic and heterogeneous nature of workloads make these decision-makings difficult. Knowledge-based and heuristics-based algorithms are commonly used to tackle this problem. Knowledge-based algorithms are restricted to the specifics of the targeted Cloud architectures and applications. Heuristics-based algorithms, such as MMT algorithms, are formulated to get rid of such limitations. They use constant or adaptive thresholds to make corresponding decisions. Though they are immune to the specificity of knowledge-base, they suffer from high variance and poor convergence because of their greedy approach. Thus, we propose a reinforcement learning approach. This methodology does not require any prior knowledge. Instead, it learns the workload dynamics and adapts with it. Our approach models the decision-making issues as a Markov decision process. Motivated by this aspect, several authors have proposed learning algorithms to solve this problem. The curse of dimensionality restrains their usage and applicability. Thus, they need an elaborate offline training like Q-learning based algorithms. Even if they are online like MadVM, they fail to be scalable in real-time because of the costly operations to restrict the state space. Use of critic-only approaches, like value iteration, also causes slow convergence. We propose Megh from an actor-critic approach to overcome this deficiency. In order to overcome the curse of dimensionality, Megh projects the combinatorially explosive state-action space to a polynomial dimensional space with sparse basis. Megh updates the transition operator T incrementally without using any prior knowledge of workload dynamics. Through this update, Megh learns the uncertainty and dynamics of workload as-it-goes. We leverage a data structure based on the sparsity of the basis for fast and scalable real-time updates and learning. Megh incurs the smallest cost and the least execution overhead with respect to its contenders both on PlanetLab and Google Cluster workloads. This validates Megh's claim as a cost-effective, time-efficient and robust algorithm. We also produce a comparative scalability analysis of Megh and THR-MMT. It demonstrates that Megh has better scalability than the competing algorithm. We explicate our choices of parameters controlling the exploration-exploitation trade-off through a sensitivity analysis of Megh.

We have conducted additional experiments that show effects of energy and SLA costs on the performance of Megh. We did not present any detailed analysis of them due to space constraints. We are currently investigating the opportunity to take advantage of additional knowledge about the workload, such as periodicity, and also to leverage knowledge of the network topology like fat-trees [36]. Finally, following previous works on energy efficient live migration of Clouds, we have considered only CPU utilization data. We are confident that our solution is able to seamlessly account for network and memory sharing. We are studying the necessary extension to the cost model to such settings in order to apply the Megh strategy.

APPENDIX A
PROOF OF THEOREM 1

Proof. As we project the state-action space $S \times A$ to the space X spanned by d dimensional basis vectors $\{\phi_j\}_{j=0}^{d}$, we reduce our search space from whole state-action space to a subspace S^t. S^t is the set of all the states reachable through one migration action from the present state $s_t \in S$. Suppose $S^t = \{s^1, s^2, \ldots, s^d\}$. Note that we use superscripts to denote the ordering of elements in S^t.

Thus, at each time-step t, we update the value functions of the only reachable states in S^t. Let $\boldsymbol{V} = (V(s))_{s\in S^t}^{T}$ and M be a $d \times d$ matrix such that

$$\Psi_{i,j} = \phi_{\pi(s^t)}[j] \quad \forall j = 1, \ldots, d$$

where, s^i is the state reachable from s^t using action $\pi(s^t)$. Let $\boldsymbol{\theta}$ be a $|S|$-dimension column vector such that $\Psi\theta = \boldsymbol{V}$. If Ψ is invertible, $\theta = \Psi^{-1}\boldsymbol{V}$ and Theorem 1 holds.

We claim that Ψ is invertible and its inverse is the matrix Ω such that,

$$\Omega_{i,j} = (-1)^{|s^i|-|s^j|}\Psi_{i,j}.$$

In order to establish this construction, let us consider the i, j^{th} element of the matrix obtained by multiplying Ψ and Ω. If $s_i \sim s_j$ means state s_j is reachable from state s_i through one of the d migration actions, then

$$\begin{aligned}(\Omega\Psi)_{i,j} &= \sum_{1\le k\le |S^t|} (-1)^{|s^i|-|s^k|}\Psi_{i,k}\Psi_{k,j} \\ &= \sum_{s_j\sim s_k\sim s_i} (-1)^{|s^i|-|s^k|}.\end{aligned}$$

Therefore $(\Omega\Psi)_{i,j} = 1$ if and only if $i = j$. Thus, Ψ is invertible and there exists a unique projection for given basis vectors. □

APPENDIX B
PROOF OF THEOREM 2

Proof. Let us denote the set of all possible value functions V^π obtained using policy $\pi \in \mathcal{U}$. Without loss of generality, we can assume $\mathcal{V}$: $S \to \mathbb{R}$ be a set of bounded, real-valued functions. Then $\mathcal{V}$ is a Banach space with the norm $\|v\| = \|v\|_\infty = \sup |v(s)|$ for any $v \in \mathcal{V}$.

Now, if we rephrase our problem of Equation 8 by including the projection, we obtain,

$$\underset{\pi\in\mathcal{U}}{\operatorname{argmin}}\, \mathbb{E}_f \left[\sum_{t=1}^{\infty} \gamma^{t-1} C(s_{t-1}, s_t)\right] \tag{12}$$

such that, $s_t \in S_{t-1}$ i.e, s_t is reachable from s_{t-1} through one of the d migrations. Then Algorithm 1 is analogous to LSPI over the reduced search space X. For this new problem given by Equation (12), Algorithm 1 converges to a unique cost-to-go function, say $\tilde{V} \in \mathcal{V}$. We need to show that the cost-to-go function estimated by Algorithm 1 is the optimal one i.e, $V^* = \tilde{V}$.

Let us define the process of updating policy as a mapping $\mathcal{M}$: $\mathcal{V} \to \mathcal{V}$. Now using Equation 9, we can formulate $\mathcal{M}$ as

$$\mathcal{M}v(s) = \min_{s'\in S_s} \mathbb{E}_f \left[C(s, s') + \gamma v(s')\right].$$

For a given state s, let

$$a_s^*(v) = \underset{s'\in S_s}{\operatorname{argmin}} \left(C(s, s') + \gamma v(s')\right).$$

Let us assume that $\mathcal{M}v(s) \ge \mathcal{M}u(s)$ If $s^*(v)$ is the state obtained by following optimal policy π^* from value function v and state s, then

$$\begin{aligned}0 &\le \mathcal{M}v(s) - \mathcal{M}u(s) \\ &= \mathbb{E}\left[C(s, s^*(v)) + \gamma v(s^*(v))\right] \\ &\quad - \mathbb{E}\left[C(s, s^*(u)) + \gamma u(s^*(u))\right] \\ &\le \mathbb{E}\left[C(s, s^*(u)) + \gamma v(s^*(u))\right] \\ &\quad - \mathbb{E}\left[C(s, s^*(u)) + \gamma u(s^*(u))\right] \\ &= \gamma\mathbb{E}\left[v(s^*(u)) - u(s^*(u))\right] \\ &\le \gamma\mathbb{E}\left[\|v - u\|\right] = \gamma\|v - u\|.\end{aligned}$$

This result states that if $\mathcal{M}v(s) \ge \mathcal{M}u(s)$, then

$$\mathcal{M}v(s) - \mathcal{M}u(s) \le \gamma|v(s) - u(s)|.$$

If we assume that $\mathcal{M}v(s) \le \mathcal{M}u(s)$, the same reasoning produces

$$\mathcal{M}v(s) - \mathcal{M}u(s) \ge -\gamma|v(s) - u(s)|.$$

Thus we can conclude, $|\mathcal{M}v(s) - \mathcal{M}u(s)| \le \gamma|v(s) - u(s)|$ for all configuration $s \in S$. From the definition of our norm, we can write

$$\begin{aligned}\sup_{s\in S} |\mathcal{M}v(s) - \mathcal{M}u(s)| &= \|\mathcal{M}v - \mathcal{M}u\| \\ &\le \gamma\|v - u\|.\end{aligned}$$

This means for $0 \le \gamma < 1$, $\mathcal{M}$ is a contraction mapping. Following [27, Proposition 3.10.2], there exists a unique v^* such that $\mathcal{M}v^* = v^*$. Thus, for an arbitrary initial value function v^0, the sequence v^n generated by $v^{n+1} = \mathcal{M}v^n$ converges to v^*. By the property of convergence of LSPI [27], $v^* = \tilde{V}$. As the cost function C is a positive and monotonically increasing function, the optimal cost-to-go function V^* also satisfies $\mathcal{M}V^* = V^*$. Hence $V^* = \tilde{V}$ and the property of convergence of LSPI is preserved in Algorithm 1. □

REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[2] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, 2005, pp. 273–286.

[3] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, 2005, pp. 25–25.

[4] C. L. Belady, "In the data center, power and cooling costs more than the it equipment it supports," 2007.

[5] P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, *Service level agreements for cloud computing*. Springer Science & Business Media, 2011.

[6] 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," *Concurr. Comput. : Pract. Exper.*, vol. 24, no. 13, pp. 1397–1420, Sep. 2012.

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

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. 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*, vol. 41, no. 1, pp. 23–50, 2011.

[10] K. Park and V. S. Pai, "Comon: a mostly-scalable monitoring system for planetlab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65–74, 2006.

[11] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, pp. 1–14, 2011.

[12] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. M. Lau, "Dynamic virtual machine management via approximate markov decision process," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[13] R. Nathuji and K. Schwan, "Virtualpower: coordinated power management in virtualized enterprise systems," in *Proce. SOSP*, 2007, pp. 265–278.

[14] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. NSDI*, 2007, pp. 11–13.

[15] Z. Á. Mann, "Allocation of virtual machines in cloud data centers: A survey of problem models and optimization algorithms," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 11, 2015.

[16] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *INFOCOM, 2011 Proceedings IEEE*, 2011, pp. 71–75.

[17] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *Computers, IEEE Transactions on*, vol. 63, no. 11, pp. 2647–2660, 2014.

[18] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *INFOCOM, 2012 Proceedings IEEE*, 2012, pp. 702–710.

[19] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[20] F. Farahnakian, P. Liljeberg, and J. Plosila, "Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, Feb 2014, pp. 500–507.

[21] S. S. Masoumzadeh and H. Hlavacs, "Integrating vm selection criteria in distributed dynamic vm consolidation using fuzzy q-learning," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, Oct 2013, pp. 332–338.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "Vconf: A reinforcement learning approach to virtual machines auto-configuration," in *Proceedings of the 6th International Conference on Autonomic Computing*, ser. ICAC '09. New York, NY, USA: ACM, 2009, pp. 137–146.

[24] L. Baird *et al.*, "Residual algorithms: Reinforcement learning with function approximation," in *Proceedings of the twelfth international conference on machine learning*, 1995, pp. 30–37.

[25] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[26] I. Grondman, L. Busoniu, G. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 42, no. 6, pp. 1291–1307, Nov 2012.

[27] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *The Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[28] L. Minas and B. Ellison, *Energy efficiency for information technology: How to reduce power consumption in servers and data centers*. Intel Press, 2009.

[29] K. Huppler, K.-D. Lange, and J. Beckett, "Spec: Enabling efficiency measurement," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, 2012, pp. 257–258.

[30] K.-D. Lange, "Identifying shades of green: The specpower benchmarks," *Computer*, vol. 42, no. 3, pp. 95–97, 2009.

[31] R. Bellman and R. E. Kalaba, *Dynamic programming and modern control theory*. Citeseer, 1965, vol. 81.

[32] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley-Interscience, 2007.

[33] K. E. Atkinson, *An introduction to numerical analysis*. John Wiley & Sons, 2008.

[34] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' guide*. Siam, 1999, vol. 9.

[35] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *Annals of Mathematical Statistics*, vol. 20, p. 317, 1949.

[36] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, Oct. 1985.