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

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

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 an online reinforcement learning algorithm called Megh. Megh does not require prior knowledge of the workload rather learns the dynamics of workloads as-it-goes. Megh models the problem of energy- and performance-efficient resource management during live migration as a Markov decision process and solves it using a functional approximation scheme. While several reinforcement 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. These algorithms often incur execution overheads which are comparable with the migration time of a VM. Megh overcomes 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 without incurring significant execution overhead. 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, converges faster, incurs smaller execution overhead and is more scalable than MadVM and MMT. An empirical sensitivity analysis explicates the choice of parameters in experiments.

Index Terms—Cloud computing, Reinforcement learning, Virtual machine, Live migration, Online algorithms, Markov decision process.

1 Introduction

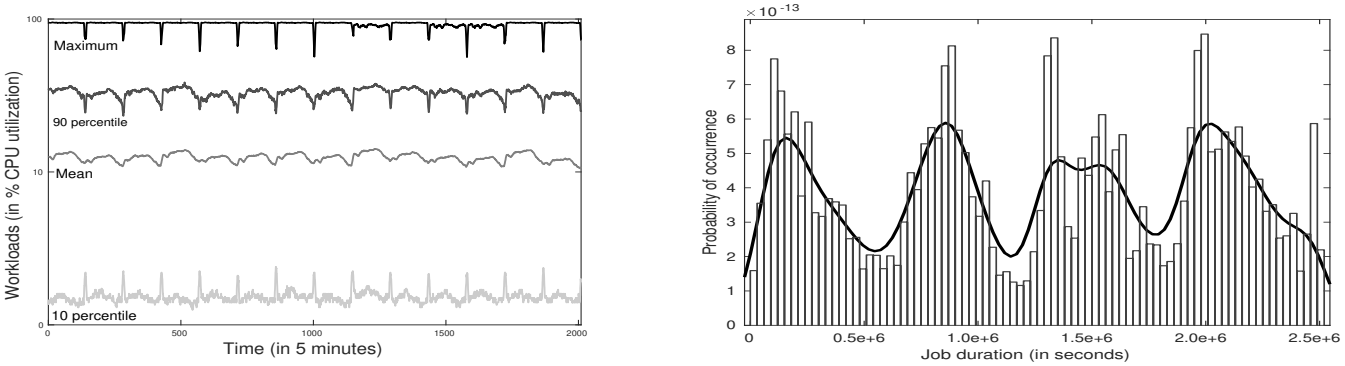
Infrastructure as a Service (IaaS) environments of Cloud computing leverage virtualization technology [1] to provide a shared platform of resources accessible at any time and from anywhere through the Internet. Cloud providers allocate Virtual Machine instances (VM) on a cluster of Physical Machines (PM). VMs allow users to share physical resources concurrently. Therefore, VMs enhance utilization of resources and increase return on investment for Cloud providers.

Making such an optimal allocation of resources is challenging not only in general-purpose IaaS Clouds [2] but also in Clouds with specialised features like scientific computing [3] or online transaction. A large number of users accessing the Cloud, the diversity of applications, and the heterogeneity of hardware yield significant variations in performance. Furthermore, the uncertain dynamics of workloads creates abrupt and unpredictable changes in resource utilization. Thus, dynamic allocation of VMs in Clouds is indispensable. In order to avoid disruption due to dynamic allocation, [4] and [5] 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. If properly carried out, live migration causes 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 centers. As energy consumption contributes almost 75% of the operation cost of a data center [**?**], 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) [6]. 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.

Energy- and performance-efficient resource management in Cloud data centers is difficult as the workloads running on the corresponding VMs are uncertain, dynamic and heterogeneous. Figures 1(a) and 1(b) reasserts this nature of the workloads in Cloud data centers. While Figure 1(a) illustrates the workloads to have long duration but high variance, Figure 1(b) depicts workloads to have wide range of durations that does not follow a standard probability distribution. These observations emphasize the need of

- *D. Basu and S. Bressan are with the Department of Computer Science, School of Computing, National University of Singapore, Singapore. E-mail: see http://www.comp.nus.edu.sg/∼deb-basu/contact.html*
- *X. Wang, Y. Hong and H. Chen are with the Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University, Shanghai, China.*



(a) PlanetLab Workload

(b) Google Cluster workload

Figure 1. Dynamics of PlanetLab workloads and distribution of task durations in Google Cluster.

learning the workload on-the-go than estimating them with a specific model in order to formulate a generic algorithm.

Knowledge-based and heuristics-based algorithms are applied to solve the resource management problem. Knowledge-based algorithms, such as MaxWeight scheduling algorithms [7] or [8] for video streaming data centers, 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 [9], [10] do not use such specific knowledge base. They save the power by greedily accumulating a majority of VMs on a smaller number of servers. Heuristics-based algorithms 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 and may make suboptimal decisions due to their myopic and greedy nature.

The shortcomings of knowledge-based and heuristics-based algorithms has motivated us to look into reinforcement learning (RL) [11]. RL is a paradigm of machine learning. In RL, an agent operating in an uncertain environment takes optimal decisions by learning more about the dynamics of its surroundings as-it-goes. If we consider the Cloud administrator *system* as a learning agent and the user workloads operating on the Cloud with corresponding resource distribution as the uncertain environment, our problem manifests as an RL problem. The system takes *optimal* live migration decisions as-it-goes by learning the dynamics of the workload and adapting accordingly. A *policy* or a sequence of decisions made by RL is optimal if it does live migration and resource management of data center with minimum operation cost. RL achieves such optimality by predicting as-it-goes the optimal decisions based on immediate costs. 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 problem of exploding state space is called curse of dimensionality in RL. Curse of dimensionality restricts the applicability of recently proposed learning algorithms in real-life scenarios. These algorithms are either not scalable in real-time, as it is the case of MadVM [12], or need an elaborate offline training, as it is the case of Q-learning [11]. We propose an online RL algorithm, called Megh, to solve this problem as-it-goes. Megh uses a functional approximation framework that uses a set of sparse basis functions to efficiently and effectively estimate the long-term effect of a migration decision. Megh projects the state space into a smaller vector space spanned by sparse basis functions and learns the dynamics of the workloads without assuming any model or prior knowledge. Megh is a robust algorithm to learn the uncertainty and diversity of workloads as-it-goes. 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.

Following the experimental setup of [9], [10], we evaluate the performance of Megh by simulating it using the CloudSim toolkit [13] over workload data extracted from PlanetLab [14] and Google Cluster [15]. 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 [9], [10]. 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 ($\sim$ 5 ms) than MMT heuristics ($\sim$ 4000 ms) even for large data center configurations. The results validate the robustness, efficiency, fast convergence, and real-time execution of Megh to cost-effectively decide live migrations under uncertain workload dynamics. A comparative scalability analysis also demonstrates Megh's better scalability than THR-MMT. Experiments also show the Q-table of Megh to grow sublinearly with the number of PMs and VMs. A sensitivity analysis empirically explicate our choices of parameters controlling the exploration-exploitation trade-off of Megh.

Our contribution. Here, we summarise the main contributions of this paper.

– We propose an online reinforcement learning algorithm, Megh, to solve the problem of energy- and performance-efficient live VM migration where the workload dynamics is not known a priori, or modelled explicitly.

- We develop a sparse projection scheme that approximates the value function uniquely (Theorem 1). While the projection scheme reduces the complexity of Megh and practically resolves the curse of dimensionality, Megh asymptotically converges to the optimal policy (Theorem 2).
- The projection scheme and the proposed online transition operator update induce two significant improvements in Megh's performance. Firstly, Megh is oblivious to the training phase. Megh learns the workload dynamics on-the-go while optimizing the decisions simultaneously. Secondly, each iteration of Megh incurs small execution time proportional to the number of VM migrations happening at that iteration.
- In order to verify these outcomes, we discuss a system model and the cost model inspired by [9], [12] (Section 3). Though Megh is applicable for even complex cost models, we use this model for further experimentation.
- We experimentally verify the performance of Megh for workload traces of PlanetLab and Google Cluster that differ significantly in nature and dynamics. Comparative performance evaluation validates that Megh reduces 14% and 8% operational cost with respect to THR-MMT and MadVM respectively, while Megh's execution time is 86% and 0.0001% of that of the THR-MMT and MadVM.

Structure of the paper. The rest of this paper is organised as follows. In Section 2, we review the related work. In Section 3, 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 4. Following that in Section 5, we propose an algorithm Megh to solve it in real-time. In Section 6, 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 7.

2 Related Works

While Megh tries to perform *energy and performance efficient live VM migrations for resource management*, the form of the problem it solves and the way it solves are based on *reinforcement learning*. Here, we review the related works in these two areas.

2.1 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 [16] and to eliminate hotspots [17]. 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. [18] recently presented an extensive survey of the problem models and optimization algorithms. Wang et al. [19] consider the dynamic network bandwidth demand for real workloads and model the VM consolidation into a Stochastic Bin Packing problem. Song et al. [20] 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. [7] 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-pre-emptive 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 [9], [10] 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.

2.2 Reinforcement Learning Algorithms

Reinforcement learning (RL) [11] is a paradigm of machine learning. In RL, an agent aims at taking optimal decisions by developing an understanding of the constantly evolving environment around it. *Markov decision processes* (MDP) [21] 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. The agent tries to fix a policy or a sequence of decisions that will maximize the cumulative sum of rewards acquired.

[22], [23] apply Q-learning [24] for energy–efficient resource management in Clouds. [25] 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 like the one addressed. But there is no reliable guarantee on the optimality of Q-learning for online learning setup for any approximated value function [26]. The general efficient VM migration problem may consist of cases where the algorithm encounters a significant variance in the real-life workload than the training one due to change in user base or their applications. Under such conditions, 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 that performs better than the Q-learning in testing phase.

MadVM [12] models the energy-efficient dynamic resource management of VMs in a data center as an approximate MDP. This algorithm assumes no prior knowledge of workload and uses value iteration [27] 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. This algorithm is indirect as it does 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, MadVM also faces the curse of dimensionality of RL approach. This leads to a key state selection procedure to connect the policy space and the value functions. This procedure for dimensionality reduction, however, is computationally expensive. MadVM tries to simultaneously optimize the utility functions of each of the VMs. Simultaneous optimization requires bookkeeping of transition functions and evaluation of key states for each of them. This computational burden makes MadVM poorly scalable for real-time applications.

Furthermore, MadVM is a critic based RL algorithm whereas Q-learning is an actor based RL algorithm. Actor based algorithms suffer from high variance due to its sensitivity to the estimates of the gradient. Critic based algorithms are stable but usually needs a discretized version of the state-action space. Discretization may lead to suboptimal results. Megh relies on the *actor-critic* [28] framework of RL and a functional approximation scheme for computation. 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 [29] as the skeleton. We utilize the projection based dimensionality reduction techniques and sparsity-based improved data structures described in Section 5 to construct our real-time learner Megh.

3 The Cloud Data Center: System and Cost Models

In the following subsections, we describe the system model of a data center used by Megh and formulate cost models for energy consumption and SLA violations. We use these cost models in Section 4 in the problem formulation, and in Section 6 for further experimentation.

3.1 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 center 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. 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. In some of the research works [30], [31], [32], [33], authors assume the distribution of incoming jobs in a workload to be a Poisson distribution and model the allocation of resources to jobs in the form of VMs as a queueing system. They are proved to be useful for systems where network dependent constraints dictate the relation between VM allocation and incoming workload though there is no discussion about effect of such workload model on energy efficiency. Since our main goal is to investigate energy efficiency of VM migration, and we do not want to assume such a distribution for the workload, or any specific network model rather we want to learn the workload dynamics and its effect on live VM migration on-the-go.

The proposed reinforcement 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 acts as the learning agent in Megh. The global manager accumulates the information and allocates the resources such that the energy consumption as well as the SLA violation will be minimized. Following this, the decision is sent to VMMs as a resource map and VMs are migrated and consolidated accordingly. Megh may migrate 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.

Following previous works on energy–efficient live migration in Clouds [9], [10], [12], we consider CPU utilization data as the key metric of characterizing the workloads. We are aware of the importance of bandwidth and memory as resources and research works [34], [35] accounting available bandwidth and network traffic as principal decision variables for VM migration. One can build cost models for these resources and add them as additional modules in the cost calculation without modifying Megh algorithmically.

3.2 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 [36]. Following the works by [10], we leverage the power consumption data provided by the SPECpower_ssj®2008 benchmark [37], [38] rather than moving our focus to precisely modelling $P(\theta)$. This is a certified industry-standard benchmark to evaluate the power and performance characteristics of server-class computer equipments. SPECpower_ssj®2008 is tested on a wide variety of operating systems and hardware architectures to remove extensive dependence on data center infrastructure for power–performance characteristics calculations. This benchmark spec2014 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 1. Now, if we assume that 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.

3.3 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 users certain monetary penalties if the QoS degrades below certain levels. We also observe that QoS is defined as the uptime percentage of the user. *Uptime* is the percentage of total access time for which the user can utilize the Cloud services without any interruption. *Downtime* is the percentage of total access time for which the user cannot utilize the Cloud services due to the interruption. 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 consider the exact downtime without such bias. Thus, *SLA violation cost* at time t for a Cloud with M PMs and N VMs can be expressed as,

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

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

$$c_{\mathrm{v}}^{j}(t) = \begin{cases} cv_1, & \text{if user's downtime percentage up to } t \\ & \in (0.05\%, 0.10\%] \\ cv_2, & \text{if user's downtime percentage up to } t \\ & > 0.10\% \\ 0, & \text{otherwise} \end{cases}$$

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

As we allocate and manage the resources by migrating the VMs from one machine to another, we face two cases of QoS degradation. In the first case, when one or multiple VMs are allocated to a PM, it faces a sudden rise of workload. The PM gets overloaded. Overloading happens when VMs try to use more resources than the capacity of the host PM. Overloading creates a scenario where we need to migrate VMs from that host to another. 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, some time is lost before the migration decision is made 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 the *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 PM i is the total time for which it is serving the users, we define the *percentage of overloading time* as

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

In the second case, the downtime is caused by the live migration process itself. Though the live migration transfers a VM from a host PM to another destination PM without suspending the running application, it still causes 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 $TM_{jt} \triangleq \frac{M_{jt}}{B_{jt}}$. Thus, the downtime of VM j during live migration is estimated as the time for which its estimated CPU utilization $\widehat{u_j}(t)$ will be less than a certain threshold. This threshold is introduced as a given $\alpha\% > 0$ of the workload $u_j(t)$ that is demanded from the VM by the user. Thus, we 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 estimate the *percentage of live migration downtime* of VM j as

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

Thus, the total downtime percentage for VM j up to time t is defined as the sum of its downtime due to live migration

and the overloading time of the PMs, which got overloaded while the VM was operating on it. Equations (4) and (5) give us a concrete mathematical model to calculate the SLA violation cost for each of the VMs. Though we develop and use the aforementioned cost model for SLA violation, it can be replaced with other cost models considering varying market prices and various subtle factors [39] without further modifying Megh.

4 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. Thus, the maximum number of users that the Cloud can handle is the maximum number of VMs it can allocate. 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. 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 PM is $\frac{h}{v}$. Furthermore, the total number of VMs N that can be allocated to the data center 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. 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.

[40] proves that optimal scheduling of tasks in a multiprocessor system is impossible in the absence of any prior knowledge of the deadline and the request distribution. [41] states that resource allocation among even soft real-time tasks under fully stochastic environment is analytically intractable. Thus, online allocation of tasks in a data center with unknown job request distribution and unknown job durations is intractable, and learning the stochastic nature of workload is essential for taking optimal decisions.

We model the process of live migration with uncertain workloads as a Markov Decision Process [21]. In this model, a state is a configuration of the VMs, with certain workloads, operating on the PMs. Thus, the *state space* S is Cartesian product of the set of all configurations of the VMs on the PMs $\mathcal{C}$, and the workloads operating on the VMs at any instance W. 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 and stochastically, 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 PM 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 PM. 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. In the problem addressed in this paper, 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 observe $\triangle C_p(s_{t-1}, s_t)$ is always positive as the system will always consume some energy whether any migration happens or not, 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.

This formulation reduces the problem 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 [11] formulation of MDP. Infinite horizon means an action will affect all the future states and actions of the system. This formulation 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, γ makes the cumulative sum of per-stage costs convergent. Physically, γ let the effect of a past action decay with each passing instance. The discount factor inclines the system to 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, the problem can be phrased as

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

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

5 Megh: Learn to Migrate As-you-go

Depending on the cost model developed in Section 3 and the problem formulation in Section 4, we propose in thi section an online reinforcement algorithm, Megh. Megh answers three basic questions of the VM migration problem: *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 computing the optimal function π^* that minimizes the cost-to-go function. This is a sequential functional approximation problem over the space of policies $\mathcal{U}$. 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 workloads on them, we improve our approximation consequently such that the current estimation of cost-to-go function remains minimal. In RL literature, this strategy is known as *policy iteration* [11].

If transition function f i.e, the stochastic nature of workload and its effect on migration, is known *a priori*, we can apply Bellman's dynamic programming technique [42] to update the estimate of cost-to-go function at every time step using the following formula,

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

Thus, the updated policy would be $\pi_t = \text{argmin}_{\overline{\pi}_t \in \mathcal{U}} V^{\overline{\pi}_t}(s)$. The algorithm terminates when there is no or very small change in the policy. Policy iteration has strong optimality and convergence properties [43].

In live VM migration 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. The state space consists of all possible configurations of VMs on all the PMs and is combinatorially large. As computation of an estimate of the cost-to-go function involves searching through the state space S, high dimensionality of S makes the policy update expensive and almost impossible to perform in real-time. This exponential blow-up in computation due to the huge state space is called the *curse of dimensionality* [43]. Secondly, the expectation in Equation (9) is not computable as the stochastic nature of workload, its correlation with VM configurations, and transition of configurations are not known *a priori*. In order to conserve the robustness and universality of Megh, we do not restrict this workload dynamics to a specific model. Indeed that would narrow down the applications and the hardware architectures the algorithm can deal with. Megh solves both the issues.

In order to solve the curse of dimensionality, Megh projects the state-action space to a $d = N \times M$ dimensional space X. X is 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 the jk^{th} element of it is one, and all other elements are zero. All the actions or configuration changes in the Cloud are represented using these basis vectors or linear combinations of them. The basic rationale behind this projection is during transition from a state to another the accessible subspace is constructed by the states which are one action away from the present state. Instead of 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. Thus, the combinatorially explosive state-action space of VM configurations is projected to a polynomial dimensional vector space with a sparse basis. Hence Megh approximates 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. This enable Megh to update the cost-to-go function effectively in real-time. Theorem 1 proves that this design of basis vectors is not ad hoc rather leads to a unique projection vector to approximate the cost-to-go function.

Algorithm 1

function MEGH(S, A, γ, ϵ, $Temp_0$)
  Initialize $\delta \leftarrow d, B_0 \leftarrow \frac{1}{\delta}\mathbb{I}_{d\times d}, \phi_0 \leftarrow \mathbf{0}_d$,
  $\theta_0 \leftarrow \mathbf{0}_d, \pi(s_0) \leftarrow \mathbf{0}_d, z_0 \leftarrow \mathbf{0}_d, C_0 \leftarrow 0$
  while $t \geq 1$ **do**
    $a_t \leftarrow \text{argmax}_{a\in A} \quad \pi_t(s_t)$
    Take action a_t.
    Observe state s_{t+1}.
    $C_{t+1} \leftarrow$ Calculate cost using Equation (6).
    $B_{t+1} = T_{t+1}^{-1}$ update using Equation (10).
    $z_{t+1} \leftarrow z_t + \phi_{a_t} C_{t+1}$
    $\theta_{t+1} \leftarrow B_{t+1} z_{t+1}$
    $\pi(s_{t+1}) \leftarrow$ *PolicyCalculator*$(\phi_{a_t}, \theta_{t+1})$
  end while
end function

Theorem 1. *There exists a unique projection vector $\theta \in \mathbb{R}^d$ that approximates the cost-to-go function as $V(s) = \theta^T \phi_{\pi(s)}$ for all states $s \in S$ and policy $\pi \in \mathcal{U}$.*

While the projection scheme resolves curse of dimensionality, the expectation of cost-to-go function is still not computable due to lack of prior knowledge about workload dynamics, and how it affects the VM configurations and their transitions. In order to capture this notion, we create a stochastic operator T. T is updated on-the-go in a frequentist fashion. T 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 begin 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 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 [29]. As described in Section 2.2, Least-Square Policy Iteration is a functional approximation algorithm that implements in an actor-critic framework. Megh 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. We begin with a random policy that allocates equal probability to all possible

Table 1
Power Consumption of servers in Watts for different level of workload [37], [38]

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

actions. The cost of any action is initiated from 0, and the transition matrix is initiated as a diagonal matrix with elements $\frac{1}{d}$.

Theorem 2 shows that Algorithm 1 asymptotically converges to an optimal policy π^* that minimises the cost-to-go function while learning as-it-goes.

Theorem 2. *If there exists a unique vector θ such that $V_\pi(s) = \theta^T \phi_{\pi(s)}$ for any configuration s and for any policy $\pi \in \mathcal{U}$, Algorithm 1 asymptotically converges to an optimal policy.*

5.1 Inducing Exploration in Action Selection

Instead of greedily choosing the action with minimum $V^{\pi_t}(s_{t+1})$, we use Boltzmann exploration [44] as the on-policy mechanism [11]. Adaptation of Boltzmann mechanism with decreasing temperature parameter for Megh is shown in Algorithm 2. Boltzmann exploration compares the goodness of an action with respect to the others by assigning exponentially weight to each action. It allows the off-policy algorithm explore more with a bias towards the actions yielding less cost. The temperature parameter controls the trade-off between the bias towards the actions with less cost and the exploration of other actions. 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 it is trying to explore more. As $Temp$ decreases with time, *PolicyCalculator* becomes the greedy selection of the minimum. Thus, Boltzmann exploration allows to adapt the exploration and greedy selection of actions with time.

Algorithm 2

1: **function** POLICYCALCULATOR(ϕ_{a_t}, θ_{t+1})
2: $Temp_{t+1} \leftarrow Temp_t \exp(-\epsilon)$
3: **for all** $i = 1, \ldots, d$ **do**
4: $Q(s_{t+1}, a_i) \leftarrow \phi_{a_i}^T \theta_{t+1}$
5: **end for**
6: $MIN\_Q \leftarrow \min_a Q(s_{t+1})$
7: **for all** $i = 1, \ldots, d$ **do**
8: $\pi(s_{t+1})_i \leftarrow \exp\left[\frac{-Q(s_{t+1},a_i)+MIN\_Q}{Temp_{t+1}}\right]$
9: **end for**
10: **end function**

5.2 Managing the Complexity Bottleneck

Algorithm 1 has space complexity of $O(d^2)$ and time complexity of $O(d^3)$. Though this algorithm 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 space complexity bottleneck is storing the $d \times d$ matrix B. The time complexity bottleneck is computing 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 [45] provided by linear algebra packages [46], inversion of T costs time complexity of $O(d^3)$. In order to compute the inverse incrementally at every step, we use Sherman-Morrison Formula [47] on Equation (10) given by,

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

Thus, the time complexity of every step is reduced to $O(d^2)$.

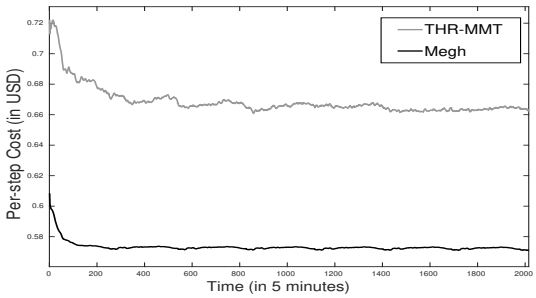
We reduce the complexity further by leveraging the sparsity of the basis vectors ϕ_{a_i}'s. Since all the zero entries are redundant in the calculation of product, we store only the non-zero entries of the matrix B and vector ϕ_{a_i} as a triplet (row number, column number, value). This reduces the initial memory storage 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 memory storage 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 non-zero 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 for live VM migration while keeping its structure and learn-as-you-go strategy intact.

6 Performance Evaluation

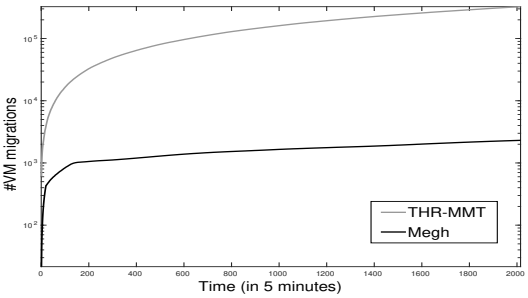
6.1 Experimental Setup

We perform experiments using the CloudSim toolkit [13] as the simulation platform. CloudSim uses CPU utilization as the key metric to characterize the workloads. We follow this characterization throughout our experiments. In the power model, we use the standard price of the local power providers, 0.18675 USD/kWh, to calculate the energy consumption cost. We assume that the user has to pay 1.2 USD 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 3.3, 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%. We consider $\beta = 70\%$ 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 two AMD Opteron(TM) Processor 6272 CPUs. Each CPU has eight cores, 128 GB memory and clock rate of 2.1GHz. Each core has two threads.

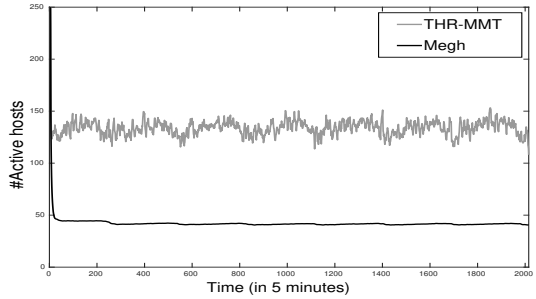
MMT algorithms are tested using the code embedded with the CloudSim toolkit, whereas Megh and MadVM are implemented and embedded in the CloudSim framework using Java. For both of them, the value of γ is set to 0.5. $\gamma = 0.5$ imposes 50:50 importance of both new and old



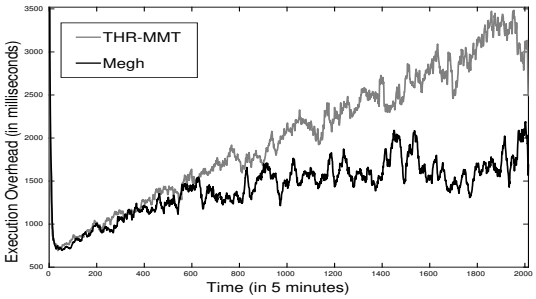
(a) Per-step cost (in USD)



(b) Number of VM migrations



(c) Number of active hosts



(d) Execution time

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

Table 2
Performance Evaluation for PlanetLab

Algorithms	THR-MMT	IQR-MMT	MAD-MMT	LR-MMT	LRR-MMT	Megh
Total cost (USD)	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 3
Performance Evaluation for Google Cluster

Algortihm	THR-MMT	IQR-MMT	MAD-MMT	LR-MMT	LRR-MMT	Megh
Total cost (USD)	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

information. $Temp_0$ and ϵ are set to 3 and 0.01 respectively for the experiments in Section 6.3 and 6.4.We explicate such choice of parameters in Section 6.5. At each time-step, we allow a maximum 2% of VMs to be migrated by Megh.

6.2 Dataset and Workload

PlanetLab Dataset

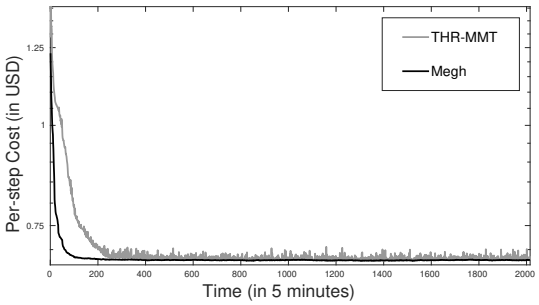
CloudSim contains workloads extracted from the CoMoN project which was a monitoring infrastructure for PlanetLab [14]. Each of the workloads consists of CPU utilization data extracted at a regular interval of 5 minutes for a span of 7 days. Figure 1(a) shows the statistical nature of the workload and depicts inherent uncertainty in its dynamics. All the workloads operate continuously on the PlanetLab system for the 7 days. The average workload operating on a VM is $\sim 12\%$ and the standard deviation of the workload is $\sim 34\%$. At any moment, the maximum and minimum workload levels vary from ~ 90 to $\sim 5\%$. This demonstrates the diversity of workload dynamics and presency of heavy workloads in the system.

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 1. 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.

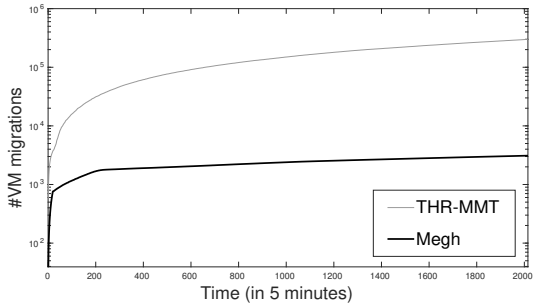
Google Cluster Dataset

The Google Cluster trace represents dynamic tasks running on Google's Hadoop MapReduce clusters with 12,500 heterogeneous machines [2]. The trace 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 Cluster trace has tasks with varying durations, starting times, and obfuscated resource usages as shown in Figure 1(b). Figure 1(b) also shows that the durations of the tasks do not follow any standard distribution, and vary in a wide range from the order of 10^1 to 10^6 seconds. These observations demonstrates need of a prior bias free learning algorithm to perform efficiently for both the datasets.

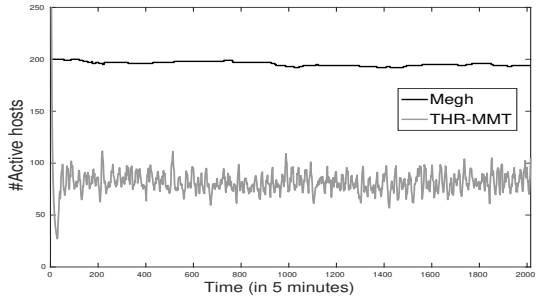
PlanetLab is a huge geo-distributed computing platform consisting of hundreds of sites and more than one thousand nodes [14]. It is hosted by organisations across the world. Users can access the computing resources by deploying applications to a subset of the nodes in the form of VMs. The trace is collected from PlanetLab to track the CPU usage of each VM's workload. The result represents the typical work-



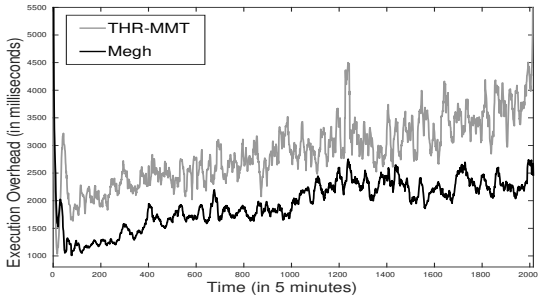
(a) Per-step cost (in USD)



(b) Number of VM migrations



(c) Number of active hosts



(d) Execution time

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

load running in an enterprise Cloud environment. While the PlanetLab trace is mainly related to academic and other organisational computation tasks, the Google Cluster trace records the events in Google's Hadoop MapReduce clusters. Google's trace shows the characteristics of workloads running in the publicly available Cloud systems [2]. In order to confirm, we plotted Cullen and Frey graph [48] for the workloads of both the datasets. They did not match with any of the standard parmateric distributions. This shows the need of learning them without imposing a prior assumption. Evaluating Megh with the traces from both the community and the industry validates its universality and robustness.

6.3 Comparative Performance Analysis

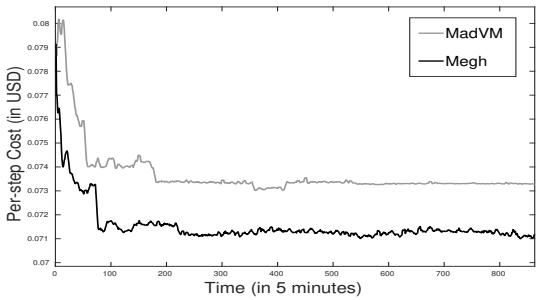
Megh vs MMT algorithms

Table 2 depicts the performance of Megh and the MMT algorithms on a week-long trace of PlanetLab. Table 3 summarizes the performance of the aforementioned algorithms for the Google Cluster dataset. Total cost of operation of the data center (in USD) 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 the 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.

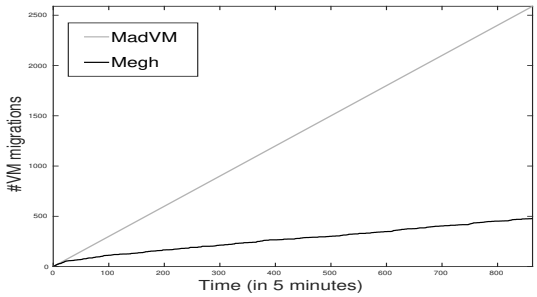
We observe from Tables 2 and 3 after 7 days of operation Megh reduces the expenditure by 14.25% for PlanetLab and 2.5% for Google Cluster with respect to that of THR-MMT. Figures 2(a) and 3(a) show the per-step operation cost for Megh not only converges faster than the contending algorithms but also 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. Megh takes around 100 time-steps before converging to almost stable cost per-step for both PlanetLab and Google Cluster datasets. We do not observe such a fast convergence for THR-MMT. THR-MMT takes around 600 and 300 time-steps in order to converge for PlanetLab and Google Cluster datasets, respectively. Being a greedy heuristics, THR-MMT still faces high variance and instability even after initial convergence. The comparatively fast convergence speed and less variance in per-step cost after convergence validate robustness and stability of Megh 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 that at any instance Megh performs significantly better.

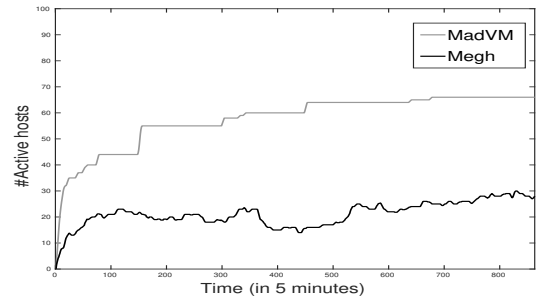
Decreasing the number of active hosts also decreases the power consumption. Thus, the number of active hosts is also used as a performance metric for resource management algorithms [12]. Though reducing the number of active hosts is the approach taken by VM consolidation algorithms, it may prove not to be a perfect metric. Because keeping a larger number of hosts at very low utilization level may cause less power consumption than keeping a few hosts at



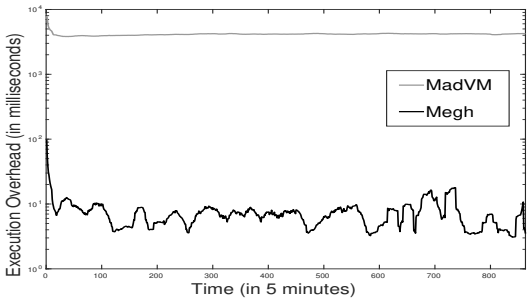
(a) Per-step cost (in USD)



(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.

very high utilization level. We observe 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. Figures 2(c) and 3(c) interestingly indicates towards a subtle balance between the number of active hosts and the feature of corresponding workloads. For the data centers with VMs running for long enough with heavy workloads, such as PlanetLab, the overloading and thus migration is unavoidable. Thus, consolidating VMs on smaller number of hosts than equally distributing them is intuitive as we see in VM consolidation literature. In contrary, we observe that if the VMs have very low workload operating for small duration, such as the Google Cluster, the VMs are distributed over larger number of hosts. As each host has less workload, the probability of overloading and hence that of the migration reduces significantly. This reduces the number of migrations and the cost due to degradation of performance but maintains more number of active hosts. This phenomenon is counter-intuitive with respect to the VM consolidation literature.

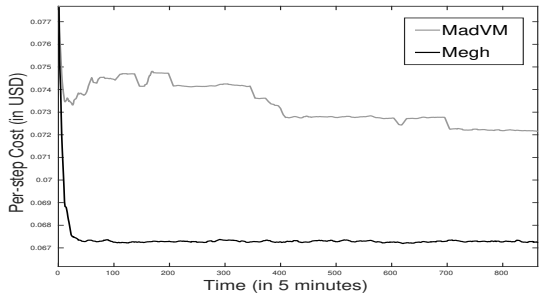
While the results establish Megh's effectiveness to solve the live migration decisions with less expenditure and better QoS, Megh has to fulfil another criterion to be a real-time system: a small execution time. From Figures 2(d) and 3(d), we observe Megh is running faster than that of the heuristic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

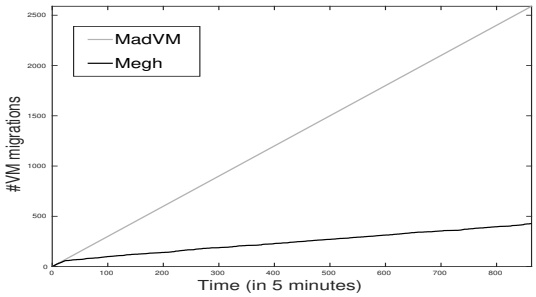
MadVM fails to scale-up for the complete PlanetLab or Google Cluster in our experimental facilities. Thus, in order to compare the performance of Megh with MadVM, we have chosen two random sets of 150 workloads running on 100 PMs for 3 days from PlanetLab and Google Cluster traces. In the beginning, 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 observe that Megh incurs less cost (4.3% and 8.8%) than MadVM at every time step. Additionally, Megh converges faster than MadVM. Figures 4(b) and 5(b) show Megh causes significantly less number (5.5 and 6.1 times) of 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 Megh (average $\sim$21 and 20). Figures 4(a) and 5(a) show that Megh takes 100 and 40 time-steps to converge whereas MadVM takes 200 and 700 steps to converge for PlanetLab and Google Cluster respectively.

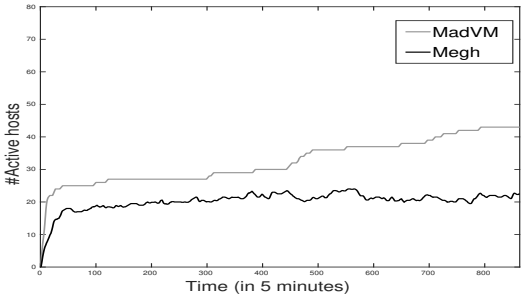
The main factor where MadVM stumbles is the execution time. MadVM takes on an average 4143ms and 4057ms to execute a single iteration for a system of 100 PMs and 150 VMs. In PlanetLab set-up, the migration time of a VM of 0.5 GB RAM is at least 4000ms. Thus, MadVM incurs a large execution overhead that disrupts VM migration to be 'live'. In contrary to MadVM, Megh incurs $\frac{1}{1000}$th of the execution overhead of MadVM that allows the live VM migration to happen without additional delay. As the RL algorithms face the curse of dimensionality and have a huge



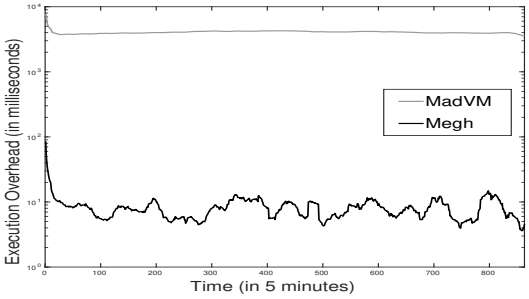
(a) Per-step cost (in USD)



(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.

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. Since Megh leverages the sparsity-based projection technique (Theorem 1), along with the specialised data structure (Section 5.2), it takes the same migration decisions in approximately 7ms and 8ms respectively for PlanetLab and Google datasets.

The experiments validate that though Megh uses the RL framework, it is significantly more efficient and faster than the latest state-of-art RL algorithm for live VM migration.

6.4 Scalability Analysis

Scalability is an important issue that an algorithm has to achieve in order to perform for a large-scale Cloud data center. We show a comparative analysis of scalability of Megh and THR-MMT in Figures 6(a) and 6(b). In order to conduct such experiments, we randomly choose m and n number of PMs and VMs from the PlanetLab data. Here, both m and n take values in $\{100, 200, 300, 400, 500, 600, 700, 800\}$. For each value of m and n, we conduct 25 experiments with 25 randomly chosen set of PMs and VMs.

We observe from Figures 6(a) and 6(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 actions 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. This scalability establishes Megh more effective as a real-time decision maker for large-scale Clouds.

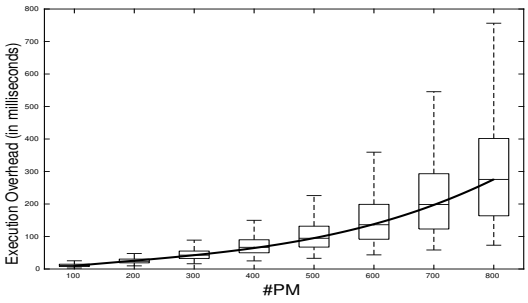
In Figure 7, we report the growth of the number of non-zero elements in the Q-table of Megh with time and the number of physical machines. We assume the number of VMs to be equal to the number of PMs for these experiments. We observe that the Q-table grows linearly with time, and shifts by certain constant factors due to increase in the number of PMs. These observations empirically proves constant increment in complexity of every iteration of Megh with time. Figure 6.4 also show that the vertical shift in the growth of Q-table for Megh is linear with respect to the number of PMs with a proportionality constant around 0.3.

As MadVM takes execution time more than the migration time of a VM even for 100$\sim$150 PMs, it is not realistic to use it for live VM migration of a larger number of PMs and VMs. Additionally, MadVM is not scalable beyond this setup for our experimental resources. Thus, we cannot conduct such a comparative study of scalability with MadVM.

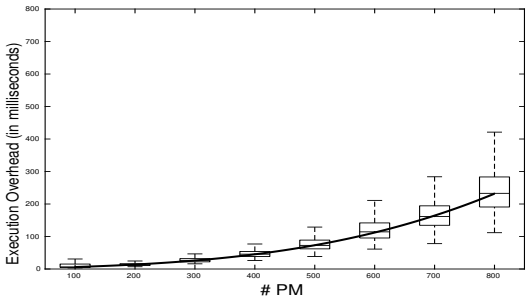
6.5 Parameter Sensitivity

$Temp_0$ 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. For each value of $Temp_0$ and ϵ, Megh is tested 25 times on the PlanetLab dataset described in Section 6.2.

Figures 8(a) and 8(b) show boxplots of per-step cost (in USD) 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 the cost rises as $Temp_0$ becomes greater than 3. Though for ϵ this change in per-step cost is a bit sporadic, we empirically observe that the



(a) THR-MMT



(b) Megh

Figure 6. Scalability analysis of THR-MMT (left) and Megh (right).

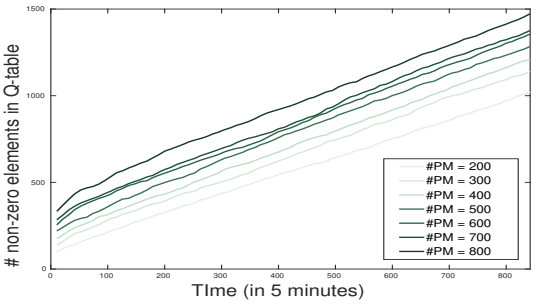


Figure 7. Increase in the number of non-zero elements in the Q-table with time and number of PMs.

variance and the median both reach a local minimum at $\epsilon = 0.001$.

Since use of $Temp$ in Algorithm 2 allows Megh to explore more rather than direct exploitation, increase in $Temp_0$ would increase the initial exploration. We observe till $Temp_0 = 3$ 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. After that point, we see the adverse effect of too much exploration. As $Temp_0$ increases after 3, the algorithm cannot benefit enough from exploitation. Thus, the curve instantiate the exploration-exploitation trade-off in case of Megh.

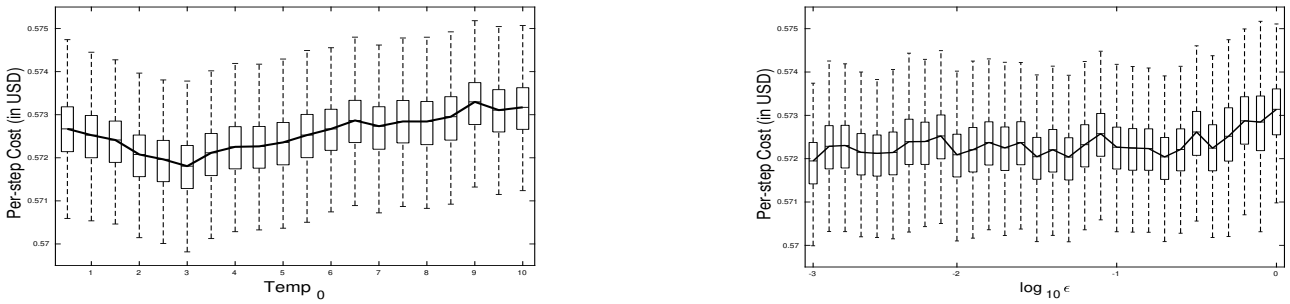
ϵ controls decay of $Temp_0$ with time. As $Temp_0$ decays, the exploratory nature turns dormant and exploitative nature begins to dominate. Thus, increase in ϵ would cause faster decay of $Temp$. Though we expect to observe similar nature as that of the variation of $Temp_0$, here we find out a bit 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.

We have conducted additional experiments that show effects of energy and SLA costs on the performance of Megh. We do not present any detailed analysis of them due to space constraints.

7 Conclusion and Future Directions

This work addresses the problem of energy– and performance–efficient resource management during live migration of VMs in a Cloud data center. Uncertain dynamics and diversity of workloads as well as the heterogeneous Cloud hardware demand for a generic algorithm to solve the efficient VM migration problem under uncertainty. Reinforcement learning provides a general framework to learn as-you-go and to take decisions under uncertainty. Thus, we propose an online reinforcement learning algorithm, Megh, that works irrespective of application and hardware heterogeneity while learning the uncertain dynamics. State-of-the-art reinforcement learning algorithms encounter curse of dimensionality and unavailability of a model for workload's uncertainty. These issues make such algorithms not scalable in real-time and asks for extensive training respectively. Megh dissolves both of the issues in real-time. It is scalable, operates in real-time with small execution overhead, and does not require a training phase. 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 incrementally without using any prior knowledge of workload dynamics. Through this update, Megh learns the uncertainty and dynamics of workload as-it-goes. Megh uses these two schemes to develop the sequential functional approximation framework with asymptotic convergence guarantee. 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. The comparative scalability analysis of Megh and THR-MMT 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 are currently investigating the opportunity to take advantage of additional knowledge about the workload, such as periodicity or a queueing model representing the dynamics of incoming workload [31], [32], [33], and also to leverage knowledge of the network topology like fat-trees [49]. We are confident that network and memory sharing can be seamlessly accommodated without modifying our solution algorithmically. Megh can be used with some other cost model till the MDP formulation of the problem is kept intact. We are studying the necessary extensions of the cost model to such settings in order to apply Megh. Though this paper majorly focuses on theoretical study and validating it on a simulation platform like [10], [12], we are also planning to extend this research and study performance of Megh in real-life large-scale Cloud data center.



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

Figure 8. Sensitivity of per-step cost (in USD) on $Temp_0$ and ϵ.

ACKNOWLEDGMENTS

We thank Prof. Pierre Senellart for his valuable feedbacks on this work. This work is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme and by the National University of Singapore Institute for Data Science project WATCHA: WATer CHallenges Analytics.

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] P. Li, S. Guo, T. Miyazaki, X. Liao, H. Jin, A. Y. Zomaya, and K. Wang, "Traffic-aware geo-distributed big data analytics with predictable job completion time," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 6, pp. 1785–1796, June 2017.

[3] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 931–945, June 2011.

[4] 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.

[5] 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*. USENIX Association, 2005, pp. 25–25.

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

[7] 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.

[8] H.-W. Tseng, T.-T. Yang, K.-C. Yang, and P.-S. Chen, "An energy efficient vm management scheme with power-law characteristic in video streaming data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 2, pp. 297–311, 2018.

[9] 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.

[10] 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.

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

[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. 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.

[14] 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.

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

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

[17] 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.

[18] Z. A. Mann, "Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 11:1–11:34, September 2015.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

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

[25] 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.

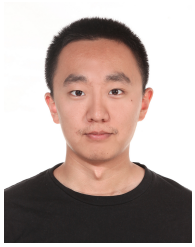
[26] L. Baird *et al.*, "Residual algorithms: Reinforcement learning with function approximation," in *Proceedings of the twelfth international conference on machine learning*, 1995, pp. 30–37.

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

[28] 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.
[29] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *The Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.
[30] K. R. Babu, A. A. Joy, and P. Samuel, "Load balancing of tasks in cloud computing environment based on bee colony algorithm," in *Advances in Computing and Communications (ICACC), 2015 Fifth International Conference on*. IEEE, 2015, pp. 89–93.
[31] H. Khazaei, J. Misic, and V. B. Misic, "Performance of an iaas cloud with live migration of virtual machines," in *Global Communications Conference (GLOBECOM), 2013 IEEE*. IEEE, 2013, pp. 2289–2293.
[32] C. Zhu, B. Han, Y. Zhao, and B. Liu, "A queueing-theory-based bandwidth allocation algorithm for live virtual machine migration," in *Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1065–1072.
[33] H. Lu, C. Xu, C. Cheng, R. Kompella, and D. Xu, "vhaul: Towards optimal scheduling of live multi-vm migration for multi-tier applications," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 453–460.
[34] D. G. Lago, E. R. Madeira, and D. Medhi, "Energy-aware virtual machine scheduling on data centers with heterogeneous bandwidths," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 1, pp. 83–98, 2018.
[35] R. Yu, G. Xue, X. Zhang, and D. Li, "Survivable and bandwidth-guaranteed embedding of virtual clusters in cloud data centers," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 2017, pp. 1–9.
[36] L. Minas and B. Ellison, *Energy efficiency for information technology: How to reduce power consumption in servers and data centers*. Intel Press, 2009.
[37] 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.
[38] S. P. Committee *et al.*, "Spec power and performance benchmark methodology," *Standard Performance Evaluation Corporation, Tech. Rep. Version*, vol. 2, 2014.
[39] A. Alsarhan, A. Itradat, A. Y. Al-Dubai, A. Y. Zomaya, and G. Min, "Adaptive resource allocation and provisioning in multi-service cloud environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 1, pp. 31–42, 2018.
[40] M. L. Dertouzos and A. K. Mok, "Multiprocessor online scheduling of hard-real-time tasks," *IEEE Transactions on software engineering*, vol. 15, no. 12, pp. 1497–1506, 1989.
[41] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-time systems*, vol. 28, no. 2-3, pp. 101–155, 2004.
[42] R. Bellman and R. E. Kalaba, *Dynamic programming and modern control theory*, 1965.
[43] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley-Interscience, 2007.
[44] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári, "Convergence results for single-step on-policy reinforcement-learning algorithms," *Machine learning*, vol. 38, no. 3, pp. 287–308, 2000.
[45] K. E. Atkinson, *An introduction to numerical analysis*. John Wiley & Sons, 2008.
[46] 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.
[47] 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.
[48] A. C. Cullen, H. C. Frey, and C. H. Frey, *Probabilistic techniques in exposure assessment: a handbook for dealing with variability and uncertainty in models and inputs*. Springer Science & Business Media, 1999.
[49] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, Oct. 1985.

Debabrota Basu Debabrota Basu received his Bachelor of Engineering degree in Electronics and Telecommunication Engineering from Jadavpur University, India. He has recently defended his PhD degree in Computer Science from School of Computing, National University of Singapore. He is going to join Chalmers University of Technology as a postdoctoral research fellow in adversarial machine learning. Basu's research interests include reinforcement learning, statistical learning theory, information theory, differential privacy, and their applications in real-world systems.



Xiayang Wang Xiayang Wang received his BS degree in software engineering from Fudan University, China, in 2014. He is now working toward the PhD degree at the School of Software, Shanghai Jiao Tong University. His research interests include program analysis and software security.

Yang Hong Yang Hong received the BS degree in software engineering from Shanghai Jiao Tong University, China, in 2013. He is now working toward the PhD degree at the School of Software, Shanghai Jiao Tong University. His research interests include parallel systems and networked systems.

Haibo Chen Haibo Chen received the PhD degree in computer science from Fudan University, in 2009. He is currently a tenured full Professor in the School of Software, Shanghai Jiao Tong University. His research interests include operating systems and parallel & distributed systems. He is a senior member of the IEEE.

Stéphane Bressan Stéphane Bressan is Associate Professor in the Department of Computer Science of the School of Computing of the National University of Singapore. Stéphane graduated in 1992 with a PhD in Computer Science from the University of Lille (France). In 1990, Stéphane joined the European Computer-industry Research Centre of Bull, ICL, and Siemens in Munich (Germany). From 1996 to 1998, he was Research Associate at the Sloan School of Management of the Massachusetts Institute of Technology (United States of America). Stéphane's research interests include the integration, management and analysis of data from heterogeneous, disparate and distributed sources.

APPENDIX

SUPPLEMENTARY MATERIAL FOR SECTION 5 (MEGH: LEARN TO MIGRATE AS-YOU-GO)

Proof of Theorem 1

Theorem 1. *There exists a unique projection vector $\theta \in \mathbb{R}^d$ that approximates the cost-to-go function as $V(s) = \theta^T \phi_{\pi(s)}$ for all states $s \in S$ and policy $\pi \in \mathcal{U}$.*

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. □

Proof of Theorem 2

Theorem 2. *There exists a unique projection vector $\theta \in \mathbb{R}^d$ that approximates the cost-to-go function as $V(s) = \theta^T \phi_{\pi(s)}$ for all states $s \in S$ and policy $\pi \in \mathcal{U}$.*

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 $\mathcal{V}\colon\ S \to \mathbb{R}$ 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 $\mathcal{M}\colon\ \mathcal{V} \to \mathcal{V}$. 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 $\mathcal{M}v(s) \geq \mathcal{M}u(s)$ If $s^*(v)$ is the state obtained by following optimal policy π^* from value function v and state s, then

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

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

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

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

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

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

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

This means for $0 \leq \gamma < 1$, $\mathcal{M}$ is a contraction mapping. Following [29, 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 [29], $v^* = \tilde{V}$. 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. □