2017 IEEE 37th International Conference on Distributed Computing Systems

A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning

Ning Liu*, Zhe Li*, Jielong Xu*, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, Yanzhi Wang
Department of Electrical Engineering and Computer Science,
Syracuse University, Syracuse, NY 13244, USA
{nliu03, zli89, zxu105, jxu21, shlin, qiqiu, jtang02, ywang393}@syr.edu

***Abstract*—Automatic decision-making approaches, such as reinforcement learning (RL), have been applied to (partially) solve the resource allocation problem adaptively in the cloud computing system. However, a complete cloud resource allocation framework exhibits high dimensions in state and action spaces, which prohibit the usefulness of traditional RL techniques. In addition, high power consumption has become one of the critical concerns in design and control of cloud computing systems, which degrades system reliability and increases cooling cost. An effective dynamic power management (DPM) policy should minimize power consumption while maintaining performance degradation within an acceptable level. Thus, a joint virtual machine (VM) resource allocation and power management framework is critical to the overall cloud computing system. Moreover, novel solution framework is necessary to address the even higher dimensions in state and action spaces.**

In this paper, we propose a novel hierarchical framework for solving the overall resource allocation and power management problem in cloud computing systems. The proposed hierarchical framework comprises a global tier for VM resource allocation to the servers and a local tier for distributed power management of local servers. The emerging deep reinforcement learning (DRL) technique, which can deal with complicated control problems with large state space, is adopted to solve the global tier problem. Furthermore, an autoencoder and a novel weight sharing structure are adopted to handle the high-dimensional state space and accelerate the convergence speed. On the other hand, the local tier of distributed server power managements comprises an LSTM based workload predictor and a model-free RL based power manager, operating in a distributed manner. Experiment results using actual Google cluster traces show that our proposed hierarchical framework significantly saves the power consumption and energy usage than the baseline while achieving no severe latency degradation. Meanwhile, the proposed framework can achieve the best trade-off between latency and power/energy consumption in a server cluster.

***Keywords*—Deep reinforcement learning; hierarchical framework; resource allocation; distributed algorithm**

I. Introduction

Cloud computing has emerged as the most popular computing paradigm in today's computer industry. Cloud computing with virtualization technology enables computational resources (including CPU, memory, disk, communication bandwidth, etc.) in data centers or server clusters to be shared by allocating virtual machines (VMs) in an on-demand manner. The effective adoption of VMs in data centers is one of the key enablers of the large-scale cloud computing paradigm. To support such a feature, an efficient and robust scheme of VM allocation and management is critical. Due to the time-variance of workloads [1,2], it is desirable to perform cloud resource allocation and management in an online adaptive manner. Automatic decision-making approaches, such as *reinforcement learning* (RL) [3], have been applied to (partially) solve the resource allocation problem in the cloud computing system [4–6]. Key properties of RL-based methods are suitable for the cloud computing systems because they do not require *a priori* modeling of state transition, workload, and power/performance of the underlying system, i.e., they are essentially *model-free*. Instead, the RL-based agents learn the optimal resource allocation decision and control the system operation in an online fashion as the system runs.

However, a complete resource allocation framework in the cloud computing systems exhibits high dimensions in state and action spaces. For example, a *state* in the state space may be the Cartesian product of characteristics and current resource utilization level of each server (for hundreds of servers) as well as current workload level (number and characteristics of VMs for allocation). An *action* in the action space may be the allocation of VMs to the servers (a.k.a. physical machines) and allocating resources in the servers for VM execution. The high dimensions in state and action spaces prohibit the usefulness of traditional RL techniques to the overall cloud computing system in that the convergence speed of traditional RL techniques is in general proportional to the number of state-action pairs [3] and will be impractically long with high state and action dimensions. As a result, previous works only used RL to dynamically adjust the power status of a single physical server [7] or the number of homogeneous VMs to an application [5,8], in order to restrict the state and action spaces.

In addition, power consumption has become one of the critical concerns in design and control of cloud computing systems. High power consumption degrades system reliability and increases the cooling cost for high-performance systems. Dynamic power management (DPM), defined as the selective shut-off or slow-down of system components that are idle

*Ning Liu, Zhe Li and Jielong Xu contributed equally to this work

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.123

IEEE computer society

or underutilized, has proven to be an effective technique for reducing power dissipation at system level [9]. An effective DPM policy should minimize power consumption while maintaining performance degradation to an acceptable level. The design of such DPM policies has been an active research area, while a variety of adaptive power management techniques including machine learning have been applied [9–11]. The DPM policies have mainly been applied to the server level, and therefore depend on the results of VM resource allocations. Therefore, a joint VM resource allocation and power management framework, which targets at the whole data center or server cluster, would be critical to the overall cloud computing system. And obviously, the joint management framework exhibits even higher dimensions in state and action spaces and requires a novel solution framework.

Recent breakthroughs of *deep reinforcement learning* (DRL) in AlphaGo and playing Atari set a good example in handling large state space of complicated control problems [12,13], and could be potentially utilized to solve the overall cloud resource allocation and power management problem. The convolutional neural network, one example of *deep neural networks* (DNN), was used to effectively extract useful information from high-dimensional image input and build a correlation between each state-action pair (s, a) and the associated *value function* $Q(s, a)$, which is the expected accumulative rewards (or costs) that the agent aims to maximize (or minimize) in RL. For online operations, a deep Q-learning framework was also proposed to derive the optimal action a at each state s in order to maximize (or minimize) the corresponding $Q(s, a)$ value. Although promising, both the DNN and the deep Q-learning framework need to be modified for applications in cloud resource allocation and power management. Moreover, the DRL framework requires a relatively low-dimensional action space [13] because in each decision epoch the DRL agent needs to enumerate all possible actions at current state and perform inference using DNN to derive the optimal $Q(s, a)$ value estimate, which implies that the action space in cloud resource allocation and power management needs to be significantly reduced.

To fulfill the above objectives, in this paper we propose a novel *hierarchical framework* for solving the overall resource allocation and power management problem in cloud computing systems. The proposed hierarchical framework comprises a *global tier* for VM resource allocation to the servers and a *local tier* for power management of local servers. Besides the enhanced scalability and reduced state/action space dimensions, the proposed hierarchical framework enables to perform the local power managements of servers *in an online and distributed manner*, which further enhances the parallelism degree and reduces the online computational complexity.

The global tier of VM resource allocation exhibits large state and action spaces, and thus the emerging DRL technique is adopted to solve the global tier problem. In order to significantly reduce the action space, we adopt a continuous-time and event-driven decision framework in which each decision epoch coincides with the arrival time of a new VM request. In this way the action at each decision epoch is simply the target server for VM allocation, which ensures that the available actions are enumerable. Furthermore, an *autoencoder* [14] and a novel *weight sharing structure* are adopted to handle the high-dimensional state space and accelerate the convergence speed, making use of the specific characteristics of cloud computing systems. On the other hand, the local tier of server power managements comprises a *workload predictor* and a *power manager*. The workload predictor is responsible for providing future workload predictions to facilitate the power management algorithm, and we adopt the *long short-term memory* (LSTM) network [15] due to its ability to capture long-term dependencies in time-series prediction. Based on the workload prediction results and the current information, the power manager adopts the model-free RL technique to adaptively determine the suitable action for turning ON/OFF of the servers in order for simultaneous reductions of power/energy consumption and job (VM) latency.

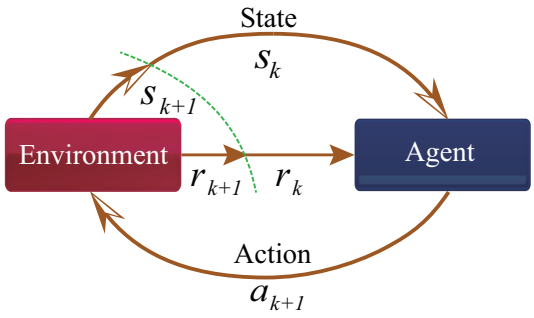


Fig. 1. Illustration of the agent-environment interaction system.

Experiment results using actual Google cluster traces [16] show that proposed hierarchical framework significantly save the power consumption/energy usage than the baseline while achieves similar average latency. In a 30-server cluster, with $95,000$ job requests, the proposed hierarchical framework can save 53.97% power and energy consumptions. Meanwhile, the proposed framework can achieve the best trade-off between latency and power/energy consumption in a server cluster. In the same case, the proposed framework gives the average per-job latency saving with the same energy usage up to 16.16%, and the average power/energy saving with the same latency up to 16.20%.

II. Background of the Agent-Environment Interaction System and Continuous-Time Q-Learning

A. Agent-Environment Interaction System

As shown in Fig. 1, the general agent-environment interaction modeling (of both traditional RL and the emerging DRL) consists of an agent, an environment, a finite state space S, a set of available actions A, and a reward function: $S \times A \to R$. The decision maker is called the *agent*, and should be trained as the interaction system runs. The agent needs to interact with the outside, which is called the *environment*. The interaction between the agent and the environment is a continual process. At each decision epoch k, the agent will make decision a_k based on the current state s_k of the environment. Once the

decision is being made, the environment would receive the decision and make corresponding changes, and the updated new state s_{k+1} of the environment would be presented to the agent for making future decisions. The environment also provides reward r_k to the agent depending on the decision a_k, and the agent tries to maximize some notion of the cumulative rewards over time. This simple reward feedback mechanism is required for the agent to learn its optimal behavior and policy.

B. Continuous-Time Q-Learning for Semi-Markov Decision Process (SMDP)

In the Q-learning procedure [17], a representative algorithm in general RL, the agent aims to maximize a *value function* $Q(s,a)$, which is the expected accumulated (with discounts) reward function when system starts at state s and follows action a (and certain policy thereafter). $Q(s,a)$ is given as:

$$Q(s,a) = \mathbf{E}\left[\int_{t_0}^{\infty} e^{-\beta(t-t_0)} r(t) dt \middle| s_0 = s, a_0 = a\right] \quad (1)$$

for continuous-time systems where $r(t)$ is the reward rate function and β is the discount rate. The $Q(s,a)$ for discrete-time systems can be defined similarly.

The Q-learning for SMDP is an online adaptive RL technique that operates in continuous time domain in an event-driven manner [18], which could reduce the overheads associated with periodic value updates in discrete-time RL techniques. Please note that the name of the technique has the term "SMDP" ONLY because it is proven to achieve the optimal policy under SMDP environment. In fact, it could be utilized to non-stationary environments as well with excellent results [19]. In Q-learning for SMDP, the definition of value function $Q(s,a)$ is continuous-time based and given in Eqn. (1). At each decision epoch t_k, the RL agent selects the action a_k using certain policy, e.g., the ϵ-greedy policy [20], similar to discrete-time RL techniques. At the next decision epoch t_{k+1} triggered by state transition, the value updating rule (from the k-th estimate to the $(k+1)$-th estimate) is given by the following:

$$Q^{(k+1)}(s_k,a_k) \leftarrow Q^{(k)}(s_k,a_k) + \alpha \cdot \Big(\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k,a_k) + \max_{a'} e^{-\beta\tau_k} Q^{(k)}(s_{k+1},a') - Q^{(k)}(s_k,a_k)\Big) \quad (2)$$

where $Q^{(k)}(s_k,a_k)$ is the value estimate at decision epoch t_k, $r(s_k,a_k)$ is the reward function, τ_k is the sojourn time that system remains in state s_k before a transition occurs, $\alpha \leq 1$ is the *learning rate*, and β is *discount rate*.

III. System Model and Problem Statement

We consider a server cluster with M physical servers that offer D types of resources with respect to the cloud resource allocation and power management framework in this paper. Usually, a server can be in active mode or sleep mode for power saving. We denote $\mathcal{M}$ as the set of physical servers, and $\mathcal{D}$ as the set of resources. Fig. 2 provides an illustration of the

This work can be generally applied to multiple server clusters or an overall data center.

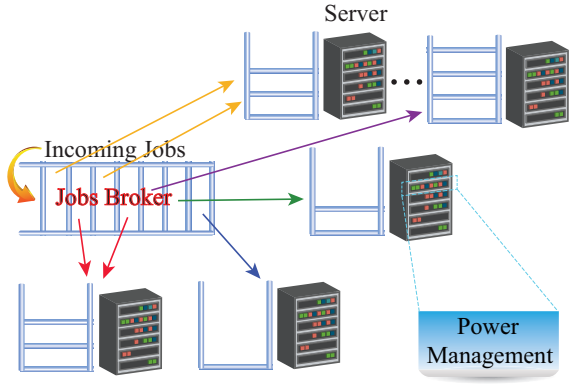


Fig. 2. Cloud resource allocation and power management framework, comprising both the global tier and the local tier.

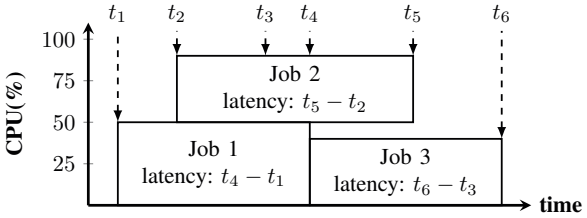


Fig. 3. An example of job execution on a server when the server is active.

hierarchical cloud resource allocation and power management framework, comprising a global tier and a local tier. A job broker, controlled by the global tier in the proposed hierarchical framework, dispatches jobs that request resource for processing to one of the servers in the cluster at its arrival time, as shown in Fig. 2. Each server queues all assigned jobs and allocates resources for them in a *first-come-first-serve (FCFS)* manner. If a server has insufficient resource to process a job, it waits until sufficient resource is released by completed jobs. On the other hand, the local tier performs power management and turns ON/OFF of each server, in a distributed manner. Both the job scheduling and the local power management will significantly affect the overall power consumption and performance of the server cluster.

We demonstrate how jobs are executed on an active server with only CPU resource usage in Fig. 3 as an example. Job 1 consumes 50% of CPU, while each of job 2 and 3 requires 40%. Job 1, 2 and 3 arrive at t_1, t_2 and t_3, respectively, and complete at t_4, t_5 and t_6, respectively. We assume that the server is in active mode at time 0. When job 1 and 2 arrive, there are enough CPU resources so their requirements are satisfied immediately. When job 3 arrives, it waits until the job 1 is completed, and the waiting time is $t_4 - t_3$. The *job latency* is defined as the duration between the job arrival and completion. Therefore the latency of job 3 is $t_6 - t_3$, which is longer than the job duration. To reduce the job latency, the job broker should avoid overloading servers. A scheduling scheme must be developed for dynamically assigning the jobs to servers and allocating resources in each server.

When a job is assigned to a server in sleep mode, it takes $T_{\mathbf{on}}$ time to change the server into the active mode. Similarly,

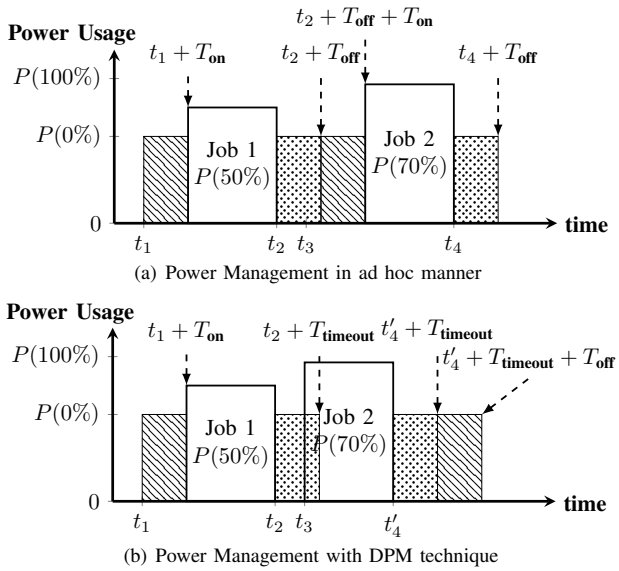


Fig. 4. Illustration of the effectiveness of server power management in the local tier.

it takes T_{off} time to change the server back to the sleep mode, which decision is made by the local tier of power manager (in a distributed manner). We assume the power consumption of a server in the sleep mode is zero, and the power consumption of a server at time t in the active mode is a function of the CPU utilization [21],

$$P(x_t) = P(0\%) + (P(100\%) - P(0\%))(2x_t - x_t^{1.4}) \quad (3)$$

where x_t denotes the CPU utilization of the server at time t, $P(0\%)$ denotes the power consumption of the server in the idle mode, and $P(100\%)$ denotes the power concumption of the server in full load. In general, the power consumption of the server during the sleep to active transition is higher than $P(0\%)$ [21,22].

We explain the effectiveness of power management in the local tier by Fig. 4. Assume that at time 0 the server is in the sleep mode, and the arrival times and CPU resource utilizations of Jobs 1 and 2 are $< t_1$, $50\% >$ and $< t_3$, $70\% >$, respectively. Job 1's arrival triggers the server to switch to the active mode and start serving job 1 from $t_1 + T_{on}$ to t_2. Fig. 4 (a) illustrates the case when power management is performed in an ad hoc manner. In this case, the server turns back to the sleep mode from t_2, and the expected completion time of power mode switching is $t_2 + T_{off}$, which is later than the arrival time of job 2 (t_3). Thus the server switches back to active mode immediately after $t_2 + T_{off}$. At $t_2 + T_{off} + T_{on}$, the server starts serving job 2 and completes at time t_4. In this case, a portion of power consumption is wasted on the frequent turning ON/OFF of the server, and also extra latency is incurred on job 2. On the other hand, Fig. 4 (b) illustrates the case when effective DPM technique is in place. In this case, an effective timeout is set after time t_2 and the server will stay in the idle state. If job 2 arrives before the timeout expires, the server will process job 2 immediately and finish at an earlier time t'_4, with $t'_4 < t_4$.

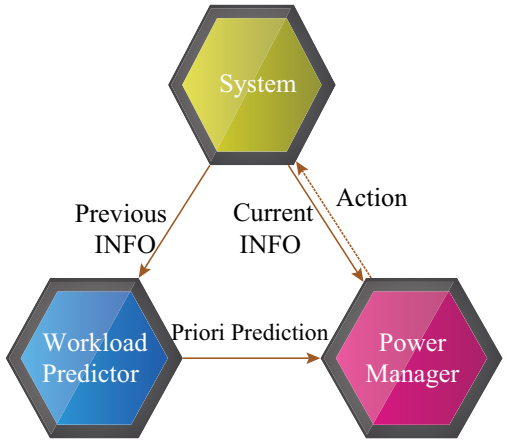


Fig. 5. Power management at each local server, comprising the workload predictor and the power manager.

A simultaneous reduction in power/energy consumption and average job latency could be achieved through the effective DPM technique at the server level, and the DPM technique for each server could be performed in a distributed manner.

The effective DPM technique in the local tier properly determines the most desirable timeout values in an online adaptive manner, based on the VM allocation results from the global tier. The DPM framework at the local server level is illustrated in Fig. 5. A *workload predictor* is incorporated in the DPM framework, which is critical to the performance of DPM by providing predictions of future workloads for the *power manager*. The prediction together with the current information such as the number of pending jobs in the queue is fed into the power manager, and serve as the state of the power manager for selecting corresponding actions and learning in the observation domain of the system.

The DPM framework of local servers relies heavily on a confident workload prediction and a properly designed power manager. In [19], a Naïve Bayes classifier is adopted to perform the workload prediction. In this paper, we want to perform a more accurate (time-series) prediction with continuous values, and thus a *recurrent neural network* (RNN) [23] or even a *long short-term memory* (LSTM) network [15] becomes a good candidate for the workload predictor. With accurate prediction and current information of the system under management, the power manager has to derive the most appropriate actions (timeout values) to help simultaneously reduce the power consumption of the server and the job latency, and the model-free RL technique [19,24] serves as a good candidate for the adaptive power management algorithm.

In Sections V and VI, we will describe the global and local tiers of the overall cloud resource allocation and power management framework. The global tier of cloud resource (VM) allocation exhibits high dimensions in state and action spaces, which prohibit the usefulness of traditional RL techniques. This is because the convergence speed of traditional RL techniques is in general proportional to the number of state-action pairs [3]. Therefore, we adopt the emerging DRL technique [13], which has the potential of handling large state

space of complicated control problems, to solve the global tier problem. For the local tier of power management, we adopt a model-free RL-based DPM technique integrated with an effective workload predictor using the LSTM network.

IV. Overview of Deep Reinforcement Learning

In this section, we present a generalized form of DRL technique compared with the prior work, which could be utilized for resource allocation and other problems as well. The DRL technique is comprised of an offline DNN construction phase and an online deep Q-learning phase [13,25]. The offline phase adopts DNN to derive the correlation between each state-action pair (s, a) of the system under control and its value function $Q(s, a)$. The offline DNN construction phase needs to accumulate enough samples of $Q(s, a)$ value estimates and corresponding (s, a) for constructing an enough accurate DNN, which can be a model-based procedure or from actual measurement data [12,26]. For the game playing applications [13], this procedure includes the pre-processing of game playing profiles/replays and obtaining the state transitioning profile and $Q(s, a)$ value estimates (e.g., win/lose or the score achieved). For the cloud resource allocation applications, we will make use of the real job arrival traces to obtain enough state transitioning profiles and $Q(s, a)$ value estimates, which can be a composite of power consumption, performance (job latency), and resiliency metrics, for the DNN construction. Arbitrary policy and gradually refined policy can be applied in this procedure. The transition profiles are stored in an experience memory $\mathcal{D}$ with capacity $N_{\mathcal{D}}$. The use of experience memory can smooth out learning and avoid oscillations or divergence in the parameters. [13]. Based on the stored state transition profiles and $Q(s, a)$ value estimates, the DNN is constructed with weight set θ trained using standard training algorithms [27].

We summarize the key steps in the offline procedure, which are shown in line 1-4 in Algorithm 1.

The deep Q-learning technique is adopted for the online control based on the offline-trained DNN. More specifically, at each decision epoch t_k of an execution sequence, the system under control is in a state s_k. The DRL agent performs inference using the DNN to derive the $Q(s_k, a)$ estimate of each state-action pair (s_k, a), and uses ϵ-greedy policy to derive the action with the highest $Q(s_k, a)$ with probability $1 - \epsilon$ and choose the other actions randomly with total probability ϵ. The chosen action is denoted by a_k. At the next decision epoch t_{k+1}, the DRL agent performs Q-value updating based on the total reward (or cost) $r_k(s_k, a_k)$ observed during this time period $[t_k, t_{k+1})$. At the end of the execution sequence, the DRL agent updates the DNN using the newly observed Q-value estimates, and the updated DNN will be utilized in the next execution sequence. More detailed procedures are shown in Algorithm 1.

It can be observed from the above procedure that the DRL framework is highly scalable for large state space and could deal with the case of continuous state space, which is distinct from traditional RL techniques. On the other hand, the DRL framework requires a relatively low-dimensional action space because in each decision epoch the DRL agent needs to enumerate all possible actions at current state and perform inference using DNN to derive the optimal $Q(s, a)$ value estimate, which implies that the action space in the cloud resource allocation framework needs to be reduced.

Algorithm 1 Illustration of the General DRL Framework

Ensure:

1: Extract real data profiles using certain control policies and obtain the corresponding state transition profiles and $Q(s, a)$ value estimates;
2: Store the state transition profiles and $Q(s, a)$ value estimates in experience memory $\mathcal{D}$ with capacity $N_{\mathcal{D}}$;
3: Iterations may be needed in the above procedure;
4: Pre-train a DNN with features (s, a) and outcome $Q(s, a)$;

Require:

5: **for** each execution sequence **do**
6: **for** at each decision epoch t_k **do**
7: With probability ϵ select a random action, otherwise $a_k = argmax_a Q(s_k, a)$, in which $Q(s_k, a)$ is derived (estimated) from DNN;
8: Perform system control using the chosen action;
9: Observe state transition at next decision epoch t_{k+1} with new state s_{k+1}, receive reward $r_k(s_k, a_k)$ during time period $[t_k, t_{k+1})$;
10: Store transition (s_k, a_k, r_k, s_{k+1}) in $\mathcal{D}$;
11: Updating $Q(s_k, a_k)$ based on $r_k(s_k, a_k)$ and $\max_{a'} Q(s_{k+1}, a')$ based on Q-learning updating rule;
12: **end for**
13: Update DNN parameters θ using new Q-value estimates;
14: **end for**

V. The Global Tier of the Hierarchical Framework – DRL-Based Cloud Resource Allocation

In this paper, we develop a scalable hierarchical framework for the overall cloud resource allocation and power management problem, comprising a global tier of cloud VM resource allocation and a local tier of power managements. The global tier adopts the emerging DRL technique to handle the high-dimensional state space in the VM resource allocation framework. In order to significantly reduce the action space, we adopt a continuous-time and event-driven decision framework in which each decision epoch coincides with the arrival time of a new VM (job) request. In this way the action at each decision epoch is simply the target server for VM allocation, which ensures that the available actions are enumerable at each epoch. The continuous-time Q-learning for SMDP [18] is chosen as the underlying RL technique in the DRL framework.

In the following we present the proposed DRL-based global tier of cloud resource allocation including novel aspects both in the offline and online phases.

A. DRL-based Global Tier of Resource Allocation

In the DRL-based global tier of cloud resource allocation, the job broker is controlled by the DRL agent, and the server

cluster is the environment. The DRL-based cloud resource allocation framework is continuous-time based and event-driven, in that the DRL agent selects an action at the time of each VM (job) request arrival, i.e., a decision epoch. The state space, action space, and reward function of the DRL-based global tier of resource allocation are defined as follows:

State Space: In the DRL-based resource allocation, we define the state at job j's arrival time t_j, s^{t_j}, as the union of the server cluster state at job j's arrival time $s_c^{t_j}$ and the job j's state s_j, i.e., $s^{t_j} = s_c^{t_j} \cup s_j$. All the M servers can be equally divided into K groups, $G_1, \cdots, G_K$. We define the state of servers in group G_k at time t as g_k^t. We also define the utilization requirement of resource type p of job j by u_{jp}, and the utilization level of server m at time t as u_{mp}^t. Therefore, the system state s^{t_j} of the DRL-based cloud resource allocation tier can be represented using u_{jp}'s and $u_{mp}^{t_j}$'s as follows:

$$\begin{aligned} s^{t_j} &= \left[s_c^{t_j}, s_j\right] = \left[g_1^{t_j}, \cdots, g_K^{t_j}, s_j\right] \\ &= [u_{11}^{t_j}, \cdots, u_{1|D|}^{t_j}, \cdots, u_{|M||D|}^{t_j}, u_{j1}, \cdots, u_{j|D|}, d_j], \end{aligned}$$

where d_j is the (estimated) job duration. The state space consists of all possible states and has a high dimension.

Action space: The action of the DRL agent for cloud resource allocation is defined as the index of server for VM (job) allocation. The action space for a cluster with M servers is defined as follows.

$$\mathcal{A} = \{a | a \in \{1, 2, \cdots, |M|\}\}$$

It can be observed that the action space is significantly reduced (to the same size as the total number of servers) by using an event-driven and continuous-time DRL-based decision framework.

Reward: The overall profit of the server cluster equals to the total revenue of processing all the incoming jobs minus the total energy cost and the reliability penalty. The income achieved by processing a job decreases with the increase in job latency, including both waiting time in the queue and processing time in the server. *Hot spot avoidance* is employed in physical servers because the overloading situations can easily lead to resource shortage and affect hardware lifetime, and thereby undermining data center reliability. Similarly, for the sake of reliability, a cloud provider can introduce *anti co-location* objectives to ensure spatial distances between VMs and the use of disjoint routing paths in the data center, so as to prevent a single failure from affecting multiple VMs belonging to the same cloud customer. It is clear that both load balancing and anti-colocation objectives are partially in conflict with power saving (and maybe job latency), as they actually try to avoid the usage of high VM consolidation ratios. Through a joint consideration of power consumption, job latency, and reliability issues, we define the reward function $r(t)$ that the agent of the global tier receives as follows:

$$\begin{aligned} r(t) = &- w_1 \cdot Total_Power(t) \\ &- w_2 \cdot Number_VMs(t) - w_3 \cdot Reli_Obj(t), \end{aligned} \tag{4}$$

where the three terms are the negatively weighted values of instantaneous total power consumption, number of VMs in the system, and reliability objective function value, respectively. Please note that according to the Little's Theorem [28], the average number of VMs pending in the system is proportional to the average VM (job) latency. Therefore, the DRL agent of the global tier optimizes a linear combination of total power consumption, VM latency, and reliability metrics when using the above instantaneous reward function.

Offline DNN Construction: The DRL-based global tier of cloud resource allocation comprises an offline DNN construction phase and an online deep Q-learning phase. The offline DNN construction phase derives the correlation between Q value estimates with each state-action pair (s^{t_j}, a). A straightforward approach is to use a conventional feed-forward neural network to directly output Q value estimates. This works well with problems with relatively small state and action spaces (e.g., in [13], the number of actions ranges from 2 to 18). Alternatively, we can train multiple neural networks in which each of them estimates the Q value for a subset of actions, for example, the Q values for assigning the job j to one subset of servers G_k. However, this procedure will significantly increase the training time of the neural network, by up to a factor of K, compared with the single network solution. Moreover, this simple multiple neural networks method cannot well stress the importance of the related servers of a target action, which slows down the training process.

In order to address these issues, we harness the power of *representation learning* and *weight sharing* for DNN construction, with basic procedure shown in Fig. 6. Specifically, we first use an autoencoder to extract a lower-dimensional high-level representation of server group state $g_k^{t_j}$ for each possible k value, denoted by $\bar{g}_k^{t_j}$. Next, for estimating the Q value of the action of allocating VM to servers in G_k, the neural network **Sub-Q**$_k$ takes $g_k^{t_j}$, s_j and all $\bar{g}_{k'}^{t_j}$ $(k' \neq k)$ as input features. The dimension difference between $g_k^{t_j}$ and $\bar{g}_{k'}^{t_j}$ $(k' \neq k)$ reflects the importance of the targeting server group's own state compared with the other server groups, which determines the degree of reduction in the state space.

In addition, we introduce weight sharing among all K autoencoders, as well as all **Sub-Q**$_k$'s. Weight sharing has been successfully applied to convolutional neural networks and image processing [29]. It is used in our model due to following reasons: 1) With weight sharing, any training samples can be used to train the **Sub-Q**$_k$'s and autoencoders, compared to the case without weight sharing, where only the training samples allocated to a server in G_k can be used to train **Sub-Q**$_k$. This usually leads to higher degree of scalability. 2) Weight sharing reduces the total number of required parameters and the training time.

Online Deep Q-Learning: The online deep Q-learning phase, which is an integration of the DRL discussed in Section III and the Q-learning for SMDP, is utilized for action selection and value function/DNN updating. At each decision epoch t_j, it performs inference using the DNN (with autoencoder incorporated) to derive the $Q(s^{t_j}, a)$ value estimate of each state-action pair (s^{t_j}, a), and use ϵ-greedy policy for action selection. At the next decision epoch t_{j+1}, it performs value

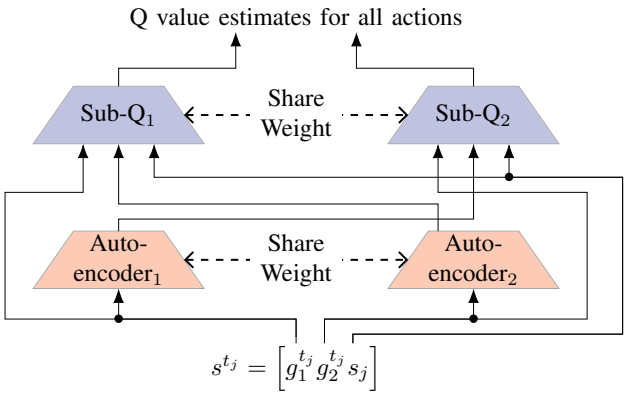


Fig. 6. Deep neural network for Q-value estimating with autoencoder and weight sharing scheme (K = 2).

updating using Eqn. (2) in Q-learning for SMDP. At the end of each execution sequence, it updates the DNN with autoencoders based on the updated Q value estimates.

B. Convergence and Computational Complexity Analysis of the Global Tier

It has been proven that the Q-learning technique will gradually converge to the optimal policy under stationary Markov decision process (MDP) environment and sufficiently small learning rate [17]. Hence, the proposed DRL-based resource allocation framework will converge to the optimal policy when (i) the environment evolves as a stationary, memoryless SMDP and (ii) the DNN is sufficiently accurate to return the action associated with the optimal $Q(s, a)$ estimate. In reality, the stationary property is difficult to satisfy as actual cloud workloads are time-variant. However, simulation results will demonstrate the effectiveness of the DRL-based global tier in realistic, non-stationary cloud environments.

The proposed DRL-based global tier of resource allocation exhibits low online computational complexity, i.e., the computational complexity is proportional to the number of actions (number of servers) at each decision epoch (upon arrival of each VM), which is insignificant for cloud computing systems.

VI. The Local Tier of the Hierarchical Framework – RL-Based Power Management for Servers

In this section, we describe the proposed local tier of power management in the hierarchical framework, which is responsible for controlling the turning ON/OFF of local servers in order to simultaneously reduce the power consumption and the average job latency. The local tier includes the workload predictor using the LSTM network and the adaptive power manager based on the model-free, continuous-time Q-learning for SMDP. Details will be described next.

A. Workload Predictor Using the Long Short-Term Memory (LSTM) Network

The workload predictor is responsible for providing partial observation of the actual future workload characteristics (i.e., inter-arrival times of jobs) to the power manager, which will be utilized for defining system states in the RL algorithm.

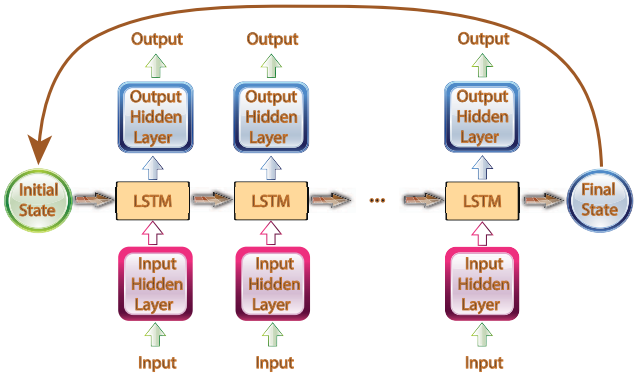


Fig. 7. Unrolled LSTM neural network for workload prediction at the local server.

The workload characteristics of servers are in fact the result of the global tier of VM resource allocation. Previous works on workload prediction in [30,31] assume that a linear combination of previous idle times (or request inter-arrival times) may be used to infer the future ones, which is not always true. For example, one very long inter-arrival time can ruin a set of subsequent predictions. In order to overcome the above effect and achieve much higher prediction accuracy, in this work we adopt the LSTM neural network [15] for workload prediction, which is a recurrent neural network architecture and can be applied for prediction of time series sequences. The LSTM network can eliminate the vanishing gradient problem and capture the long-term dependencies in the time series, and works efficiently where *back propagation through time* (BPTT) technique is adopted [32].

The LSTM network structure is shown in Fig. 7. The network has three layers, input hidden layer, LSTM cell layer and output hidden layer. In the proposed LSTM network for workload prediction, we predict the next job inter-arrival time based on the past 35 inter-arrival times as the look-back time steps. The size of both input and output of an LSTM cell should be 1 according to the dimension of the measured trace. We set 30 hidden units in each LSTM cell, and all LSTM cells have shared weights. In our LSTM-based workload prediction model, we discretize the output inter-arrival time prediction by setting n predefined categories, corresponding to n different states in the RL algorithm utilized in the power manager.

In the training process, first we initialize the weights for the input layer and output layer as a normal distribution with a mean value of 0 and standard deviation of 1. The bias for both layers is set as a constant value 0.1. The initial state of LSTM cell is set as 0 for all cells. In response to the back propagated errors, the network is updated by adopting *Adam optimization* [27], a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates from estimates of the first and second moments of the gradients [33]. The state of the LSTM cell and weights will be trained for minimizing the propagated errors.

B. Distributed Dynamic Power Management for Local Servers

In this subsection, we describe the proposed adaptive power manager for local servers using model-free RL, based on effective workload prediction results. The continuous-time Q-learning for SMDP is adopted as the RL technique in order to reduce the decision frequency and overheads. The power management for local servers is performed in a distributed manner. First, we provide a number of key definitions and notations. Let t_j^G $(j = 1, 2, 3...)$ denote the time when the j-th job (VM) arrives in the local machine. Suppose that we are currently at time t with $t_j^G \leq t < t_{j+1}^G$. Note that the power manager has no exact information on t_{j+1}^G at the current time t. Instead, it only receives an estimation of inter-arrival time from the LSTM-based workload predictor.

The power manager monitors the following *state parameters* of the power managed system (server):

1) The machine power state $M(t)$, which can be active, idle, sleep, etc.
2) The estimated next job inter-arrival time from the workload predictor.

To apply RL techniques to DPM frameworks, we define *decision epochs*, i.e., when new decisions are made and updates for the RL algorithm are executed. In our case, the decision epochs coincide with one of the following three cases:

1) The machine enters the idle state (usage = 0) and $JQ(t) = 0$ (no waiting job in the buffer queue).
2) The machine is in the idle state and a new job arrives.
3) The machine is in the sleep state and a new job arrives.

The proposed RL-based DPM operates as follows. At each decision epoch, the power manager finds itself in one of the three aforesaid conditions and it will make a decision. If it finds itself in case (1), it will use the RL-based timeout policy. A list of timeout periods, including the immediate shutdown (timeout value = 0), serve as the action set $\mathcal{A}$ in this case, and the power manager learns to choose the optimal action by using RL technique. If it is in case (2), it will start executing the new job and turn from idle to active. If it is in case (3), the server will turn active from the sleep state and start the new job as soon as it is ready. Updates are no need to perform in cases (2) and (3) because there is just one action to choose. As pointed out in reference [34], the optimal policy when the machine is idle for non-Markov environments is often a timeout policy, wherein the machine is turned off to sleep if it is idle for more than a specified timeout period.

In this local RL-based DPM framework, we use the reward rate defined as follows:

$$r(t) = -wP(t) - (1 - w)JQ(t) \tag{5}$$

The reward rate function is the negative of a linearly-weighted combination of power consumption $P(t)$ of server and the number of jobs $JQ(t)$ buffered in the service queue, where w is the weighting factor. This is a reasonable reward rate because as authors in [35] has pointed out, the average number of buffered service requests/jobs is proportional to the average latency for each job, which is defined as the average time for each job between the moment it is generated and the moment that the server finishes processing it, i.e., it includes the queueing time plus execution time. The above observation is based on the Little's Law [36]. In this way, the value function $Q(s, a)$ for each state-action pair (s, a) is a combination of the expected total discounted energy consumption and total latency experienced by all jobs. Since the total number of jobs and the total execution time are fixed, the value function is equivalent to a linear combination of the average power consumption and average per-job latency. The relative weight w between the average power consumption and latency can be adjusted to obtain the power-latency trade-off curve. The detailed procedure of the RL algorithm is in Algorithm 2.

Algorithm 2 The RL-based DPM framework in the local tier.

1: At each decision epoch t^D, denote the RL state as $s(t^D)$ for the power-managed system.
2: **for** each decision epoch t_k^D **do**
3: With probability ϵ select a random action from action set $\mathcal{A}$ (timeout values), otherwise $a_k = argmax_a Q(s(t_k^D), a)$.
4: Apply the chosen timeout value a_k.
5: If job arrives during the timeout period, turn active to process the job until the job queue is empty. Otherwise turn sleep until the next job arrives.
6: Then we arrive at the next decision epoch t_{k+1}^D.
7: Observe state transition at next decision epoch t_{k+1}^D with new state $s(t_{k+1}^D)$, and calculate reward rate during time period $[t_k^D, t_{k+1}^D)$.
8: Updating $Q(s(t_k^D), a_k)$ based on reward rate and $\max_{a'} Q(s(t_{k+1}^D), a')$ based on the updating rule of Q-learning for SMDP (Eqn. (2)).
9: **end for**

VII. Experimental Results

In this section, we first describe the simulation setup. Then, from perspectives of power consumption and job latency we compare our proposed hierarchical framework for resource allocation and power management with the DRL-based framework for resource allocation ONLY, and the baseline round-robin VM allocation policy. Finally, we evaluate our proposed framework by investigating the optimal trade-off between the power consumption and latency.

A. Simulation Setups

Without loss of generality, in this paper, we assume a homogeneous server cluster. The peak power of each server is $P(100\%) = 145$W, and the idle power consumption is $P(0\%) = 87$W [21]. We set the server power mode transition times $T_{\textbf{on}} = 30$s and $T_{\textbf{off}} = 30$s. The number of machines in each cluster M is set as 30, 40 respectively. Please note that our proposed framework is applicable to the case with more servers as well.

We use real data center workload traces from Google cluster-usage traces [16], which provide the server cluster usage data over a month-long period in May 2011. The extracted job traces include job arrival time (absolute time value), job duration, and resource requests of each job, which

include CPU, memory and disk requirements (normalized by the resource of one server). All the extracted jobs are with a duration between 1 minute and 2 hours, ordered increasingly based on their arrival time. To simulate the workload on a 30-40 machines cluster, we split the traces into 200 segments, and each segment contains about 100,000 jobs, corresponding to the workload for a M-machine cluster in a week.

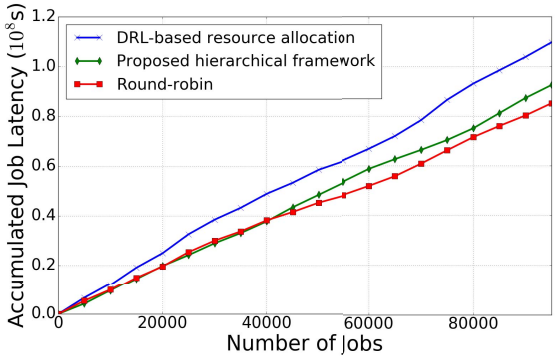
In this work, for the global tier, we perform offline training, including the experience memory initialization and training of the autoencoder, using the whole Google cluster traces. To obtain DNN model in the global tier of the proposed framework, we use workload traces for five different M-machine clusters. We generate four new state transition profiles using the ϵ-greedy policy [20] and store the transition profiles in the memory before sampling the minibatch for training the DNN. In addition, we clip the gradients to make their norm values less than or equal to 10. The gradient clipping method has been introduced to DRL in [37].

In DNN construction, we use two layers of fully-connected Exponential Linear Units (ELUs) with 30 and 15 neurons, respectively, to build an autoencoder. Each **Sub-Q**$_k$ mentioned in Section V contains a single fully-connected hidden layer with 128 ELUs, and a fully-connected linear layer with a single output for each valid action in the group. The number of groups varies between 2 and 4. We use $\beta = 0.5$ in simulations for Q-learning discount rate.

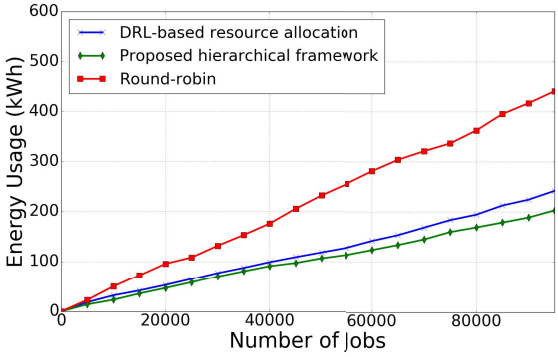
B. Comparison with Baselines

We simulate five different one-week job traces using the proposed hierarchical framework with different parameters and compare the performance against the DRL-based resource allocation framework and the round-robin allocation policy (denoted as the baseline) in terms of the (accumulative) power consumption and the accumulated job latency.

Fig. 8 shows the experimental results when the number of machines in the cluster $M = 30$. Compared with the round-robin method, the hierarchical framework and DRL-based resource allocation result in longer latency than the baseline round-robin method shown in Fig. 8(a). This is because in the round-robin method jobs are dispatched evenly to each machine, and generally, the jobs do not need to wait in each machine job queue. However, from the perspective of energy (accumulative power consumption) usage shown in Fig. 8(b), the round robin method gives a larger increase rate than the hierarchical framework or DRL-based resource allocation, which implies the round robin method has a larger power consumption. On the other hand, compared with the DRL-based resource allocation, the proposed hierarchical framework shows a reduced job latency shown in Fig. 8(a) as well as a lower energy usage shown in Fig. 8(b). From Fig. 8(b), we can also observe the energy chart for the proposed framework is always lower than that for DRL-based resource allocation or round robin baseline, which means our proposed hierarchical framework achieves the lowest power consumption among three. Shown in Table I, given the number of jobs of $95,000$, compared with the round-robin method, our proposed hierarchical framework saves 53.97% power and energy. It also saves



(a) Accumulated job latency versus the number of jobs



(b) Energy usage versus the number of jobs

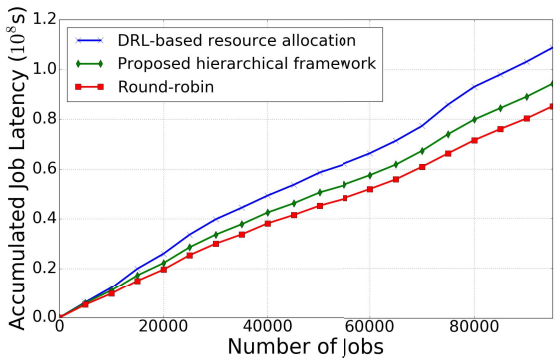
Fig. 8. Comparison among the proposed hierarchical framework, the DRL-based resource allocation framework, and the round-robin baseline in the case of $M = 30$.

16.12% power/energy and 16.67% latency compared with the DRL-based resource allocation framework.

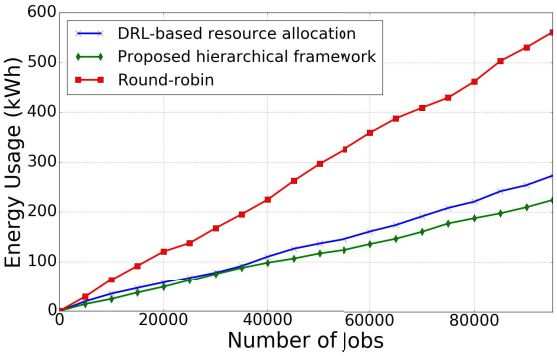
Similarly, in the case of 40 machines in the cluster as shown in Table I, given the number of jobs of $95,000$, the proposed hierarchical framework consumes 59.99% less power and energy compared with round-robin baseline, as well as 17.89% less power and energy compared with the DRL-based resource allocation framework. The latency of the hierarchical framework reduces by 13.32% compared with the DRL-based resource allocation framework. From Fig. 8(a), 9(a), we can observe that the corresponding hierarchical frameworks' latency increase rates in two figures differ very little and so as to the corresponding DRL-based resource allocation frameworks. In Fig. 8(b) and Fig. 9(b), the trend of energy usage of cor-

TABLE I
SUMMARY OF THE SERVER CLUSTER PERFORMANCE METRICS (ACCUMULATED ENERGY, ACCUMULATED LATENCY, AND AVERAGE POWER CONSUMPTION) WITH JOB NUMBER = 95000

Round-Robin	Energy (kWh)	Latency(10^6s)	Power (W)
$M = 30$	441.47	85.20	2627.79
$M = 40$	561.13	85.20	3340.06
DRL-based	Energy (kWh)	Latency(10^6s)	Power (W)
$M = 30$	242.25	109.73	1441.96
$M = 40$	273.41	108.76	1627.44
Hierarchical	Energy (kWh)	Latency(10^6s)	Power (W)
$M = 30$	203.21	92.53	1209.58
$M = 40$	224.51	94.26	1336.37



(a) Accumulated job latency versus the number of jobs



(b) Energy usage versus the number of jobs

Fig. 9. Comparison among the proposed hierarchical framework, the DRL-based resource allocation framework, and the round-robin baseline in the case of $M = 40$.

responding hierarchical frameworks and DRL-based resource allocation frameworks remains close which means their power consumption remains as the number of machines increases. However, the increase rate of energy usage (power) for round-robin methods becomes larger as the number of machines M increases. Thus, when the number of jobs increases, in a larger-size server cluster, the round robin baseline system consumes an unacceptable amount of power/energy. These facts imply that the DRL-based resource allocation framework is capable of managing a large-size server cluster with an enormous number of jobs. With the help of local and distributed power manager, the proposed hierarchical framework achieves shorter latency and less power/energy consumption.

C. Trade-off of Power/Energy Consumption and Average Latency

Next, we explore the power (energy) and latency trade-off curves for the proposed framework as shown in Fig. 10. The latency and energy usage are averaged for each job. We first set three baselines with the DRL-based resource allocation tier and the local tier with different fixed timeout values, which is discussed in Section VI-B. The timeout values of the fixed timeout baselines are set to be $30s$, $60s$, and $90s$, respectively. Please note when the timeout value is fixed, the baseline system cannot reach every point in the energy and latency space, so that the curves for baselines are not complete. We can observe the proposed hierarchical framework can achieve

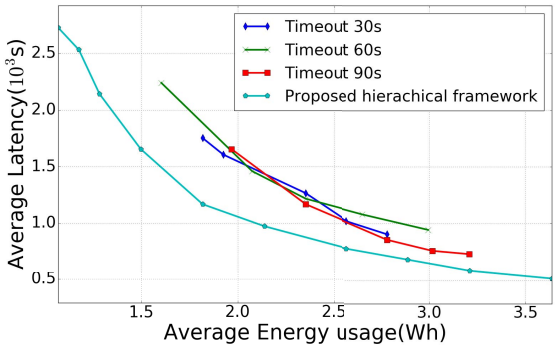


Fig. 10. Trade-off curves between average latency and average energy consumption of the proposed hierarchical framework and baseline systems.

the smallest area against the axes of power and latency, which denotes that our proposed hierarchical framework gives the best trade-off than any of those with a fixed timeout value. For instance, compared with a baseline with fixed timeout value of 60, the maximum average latency saving with the same energy usage is 14.37%, while the maximum power/energy saving with the same average latency is 16.13%; compared with a baseline with fixed timeout value of 90, the maximum average latency saving with the same energy usage is 16.16%, while the maximum average power/energy saving with the same latency is 16.20%

VIII. Conclusion

In this paper, a hierarchical framework is proposed to solve the resource allocation problem and power management problem in the cloud computing. The proposed hierarchical framework comprises a global tier for VM resource allocation to the servers and a local tier for power management of local servers. Besides the enhanced scalability and reduced state/action space dimensions, the proposed hierarchical framework enables to perform the local power managements of servers in an online and distributed manner, which further enhances the parallelism degree and reduces the online computational complexity. The emerging DRL technique is adopted to solve the global tier problem while an autoencoder and a novel weight sharing structure are adopted for acceleration. For local tier, an LSTM based workload predictor helps a model-free RL based power manager to determine the suitable action of the servers. Experiment results using actual Google cluster traces show that proposed hierarchical framework significantly save the power consumption/energy usage than the baseline while achieves similar average latency. Meanwhile, the proposed framework can achieve the best trade-off between latency and power/energy consumption in a server cluster.

References

[1] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from google compute clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[2] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *2012 IEEE Network Operations and Management Symposium*. IEEE, 2012, pp. 1287–1294.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1.

[4] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow," in *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, 2011, pp. 67–74.

[5] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[6] A. Galstyan, K. Czajkowski, and K. Lerman, "Resource allocation in the grid using reinforcement learning," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*. IEEE Computer Society, 2004, pp. 1314–1315.

[7] G. Tesauro, R. Das, H. Chan, J. Kephart, D. Levine, F. Rawson, and C. Lefurgy, "Managing power consumption and performance of computing systems using reinforcement learning," in *Advances in Neural Information Processing Systems*, 2007, pp. 1497–1504.

[8] X. Lin, Y. Wang, and M. Pedram, "A reinforcement learning-based power management framework for green computing data centers," in *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2016, pp. 135–138.

[9] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 8, no. 3, pp. 299–316, 2000.

[10] G. Dhiman and T. S. Rosing, "Dynamic power management using machine learning," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 2006, pp. 747–754.

[11] H. Jung and M. Pedram, "Dynamic power management under uncertain information," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[14] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., Mountain View, CA, USA, Technical Report, Nov. 2011, revised 2012.03.20. Posted at url-http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.

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

[18] S. J. Duff and O. Bradtke Michael, "Reinforcement learning methods for continuous-time markov decision problems," *Adv Neural Inf Process Syst*, vol. 7, p. 393, 1995.

[19] Y. Wang, Q. Xie, A. Ammari, and M. Pedram, "Deriving a near-optimal power management policy using model-free reinforcement learning and bayesian classification," in *Proceedings of the 48th Design Automation Conference*. ACM, 2011, pp. 41–46.

[20] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," *Advances in neural information processing systems*, pp. 1038–1044, 1996.

[21] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," vol. 35, no. 2, pp. 13–23, 2007.

[22] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *ACM Sigplan Notices*, vol. 44, no. 3. ACM, 2009, pp. 205–216.

[23] J. T. Connor, R. D. Martin, and L. E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 240–254, 1994.

[24] Y. Tan, W. Liu, and Q. Qiu, "Adaptive power management using reinforcement learning," in *Proceedings of the 2009 International Conference on Computer-Aided Design*. ACM, 2009, pp. 461–467.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[26] 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*. ACM, 2009, pp. 137–146.

[27] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[28] D. Gross, *Fundamentals of queueing theory*. John Wiley & Sons, 2008.

[29] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[30] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 4, no. 1, pp. 42–55, Mar. 1996. [Online]. Available: http://dx.doi.org/10.1109/92.486080

[31] C.-H. Hwang and A. C. H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *1997 Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, Nov 1997, pp. 28–32.

[32] N. Kalchbrenner, I. Danihelka, and A. Graves, "Grid long short-term memory," *arXiv preprint arXiv:1507.01526*, 2015.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[34] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli, "Event-driven power management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 7, pp. 840–857, Jul 2001.

[35] Q. Qiu, Y. Tan, and Q. Wu, "Stochastic modeling and optimization for robust power management in a partially observable system," in *Proceedings of the conference on Design, automation and test in Europe*. EDA Consortium, 2007, pp. 779–784.

[36] J. D. Little and S. C. Graves, "Little's law," in *Building intuition*. Springer, 2008, pp. 81–100.

[37] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.