DRL-Cloud: Deep Reinforcement Learning-Based Resource Provisioning and Task Scheduling for Cloud Service Providers

Mingxi Cheng*, Ji Li†, and Shahin Nazarian†
*Duke University, Durham, NC, USA (mingxi.cheng@duke.edu)
†University of Southern California, Los Angeles, CA, USA ({jli724, shahin.nazarian}@usc.edu)

***Abstract*—Cloud computing has become an attractive computing paradigm in both academia and industry. Through virtualization technology, Cloud Service Providers (CSPs) that own data centers can structure physical servers into Virtual Machines (VMs) to provide services, resources, and infrastructures to users. Profit-driven CSPs charge users for service access and VM rental, and reduce power consumption and electric bills so as to increase profit margin. The key challenge faced by CSPs is data center energy cost minimization. Prior works proposed various algorithms to reduce energy cost through Resource Provisioning (RP) and/or Task Scheduling (TS). However, they have scalability issues or do not consider TS with task dependencies, which is a crucial factor that ensures correct parallel execution of tasks. This paper presents DRL-Cloud, a novel Deep Reinforcement Learning (DRL)-based RP and TS system, to minimize energy cost for large-scale CSPs with very large number of servers that receive enormous numbers of user requests per day. A deep Q-learning-based two-stage RP-TS processor is designed to automatically generate the best long-term decisions by learning from the changing environment such as user request patterns and realistic electric price. With training techniques such as target network, experience replay, and exploration and exploitation, the proposed DRL-Cloud achieves remarkably high energy cost efficiency, low reject rate as well as low runtime with fast convergence. Compared with one of the state-of-the-art energy efficient algorithms, the proposed DRL-Cloud achieves up to 320% energy cost efficiency improvement while maintaining lower reject rate on average. For an example CSP setup with $5,000$ servers and $200,000$ tasks, compared to a fast round-robin baseline, the proposed DRL-Cloud achieves up to 144% runtime reduction.**

***Keywords*—Deep reinforcement learning, deep Q-learning, resource provisioning, task scheduling, cloud resource management.**

I. Introduction

Cloud computing has emerged as a cogent and powerful paradigm that delivers omnipresent and on-demand access to a shared pool of configurable computing resources as a service through the Internet [1]. Virtualization is the fundamental technology of cloud computing, which enables multiple operating systems to run on the same physical platform, and structures servers into Virtual Machines (VMs) [2]. VMs are used by Cloud Service Providers (CSPs) to provide infrastructures, platforms, and resources (e.g., CPU, memory, storage, etc.). In the cloud computing paradigm, CSPs are incentivized by the benefit of charging users for cloud service access, resource utilization and VM rental, whereas users are attracted by the opportunity of eliminating expenditure of implementing computational, time and power consuming applications on cloud based on their own requirements [3].

Despite the success of many well-known CSPs such as Google App Engine (GAE) and Amazon Elastic Compute Cloud (EC2), the tremendous energy costs in terms of electricity consumed by data centers is a serious challenge. Data center electricity consumption is projected to be roughly 140 billion kilowatt-hours annually by 2020, which costs 13 billion US dollars annually in electric bills [4]. Hence, in order to increase the profit margin and as well, reduce the carbon footprint for sustainable development and abstemious economical society, it is imperative to minimize the data center electricity consumption for large-scale CSPs.

According to [5], energy usage of data centers has two important features: (i) servers tend to be more energy inefficient under low utilization rate (with the optimal power efficient utilization rate of most servers ranging between 70% and 80%), and (ii) servers may consume a considerable amount of power in idle mode. Therefore, server consolidation and load balancing can be applied to improve the overall energy efficiency through selectively shutting down idle servers and improving the utilization levels in active servers. Meanwhile, the agreements in the Service-Level Agreement (SLA) should be consistently met, which is negotiated by the CSP and users regarding privacy, security, availability, and compensation [6].

Energy consumption and electric cost reduction become challenging for CSPs, and the reasons are twofold: First, scalability of expenditure control is critical due to the large-scale server farms and enormous numbers of incoming requests per day, and both of which are still growing. Second, as user request patterns can change both in short-term (within a day) and long-term (from month/year to month/year), the adaptability and self-learning capacity of the energy and electric cost reduction method are required.

Many approaches have been proposed in the literature to improve energy efficiency of data centers owned by CSPs through Resource Provisioning (RP) and/or Task Scheduling (TS) [3], [7–9]. Nevertheless, these prior works [3], [7–9] have scalability issues and their offline algorithms have difficulties in dealing with the large size of inputs and adapt to changes, e.g., dealing with different user request patterns. The recently proposed Deep Reinforcement Learning (DRL) technique, which has been shown successful in playing Atari and Go games [10–12], has outstanding ability in handling complicated control problems with high-dimensional state spaces and low-dimensional action spaces by utilizing deep neural networks [13–17]. Inspired by this, N. Liu et al. applied DRL to (partially) solve the resource allocating problem in cloud computing [18], without detailed scheduling for tasks with data dependencies, which is critical to guarantee tasks are executed correctly [19].

To comprehensively solve the energy cost reduction problem, we propose the DRL-Cloud framework, which is the first DRL-based highly scalable and adaptable RP and TS system with capability to handle large-scale data centers and changing user requests. In this paper, a general type of realistic pricing policy comprised of time-of-use pricing (TOUP) and real-time pricing (RTP) [20], [21] is used. In addition, Pay-As-You-Go billing agreement (as in GAE and EC2) is used. All deadlines are hard deadlines and a task will be rejected if the hard deadline is violated. DRL-Cloud is comprised of two major parts: i) user request acceptance and decoupling into a job queue and a task ready queue; ii) energy cost minimization by our DRL-based two-stage RP-TS processor, and fast convergence is guaranteed by training techniques in

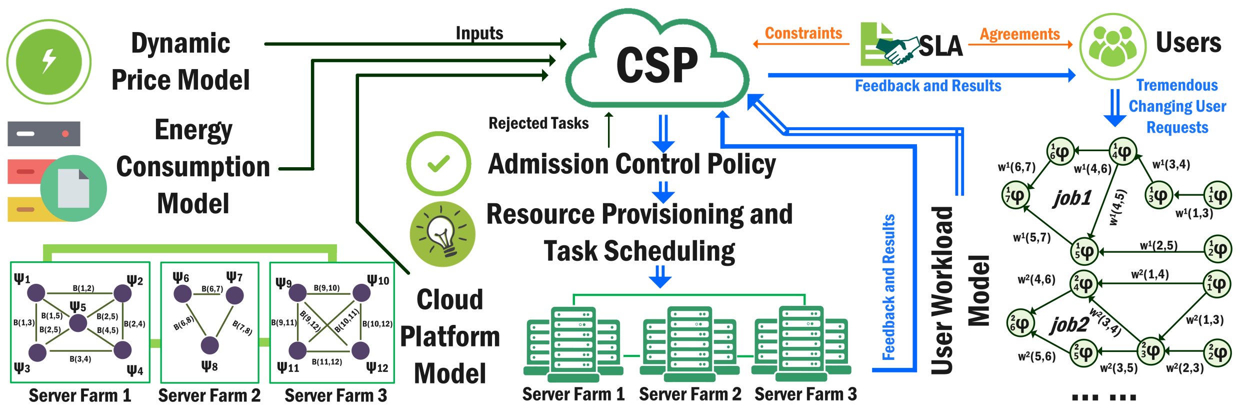

Fig. 1. System model of the cloud platform and structure for DRL-Cloud. The system model is defined in Section II. The structure and algorithm of the proposed DRL-Cloud are described in Section III and Section IV.

deep Q-learning such as target network and experience replay.

The contributions of this work are as follows:

- **Applying DRL to RP and TS.** To the best of our knowledge, this is the first paper to present a DRL-based RP and TS system to minimize energy cost for CSPs with large-scale data centers and large amounts of user requests with dependencies. A two-stage RP-TS processor based on DRL is designed to automatically generate the best actions to obtain minimum energy cost in long-term by learning from changing environment, and its multi-stage structure gives the proposed DRL-Cloud high efficiency and high scalability.
- **Semi-Markov Decision Process (SMDP) formulation.** Cloud resource allocation and energy cost minimization problem is formulated based on a semi-Markov decision process, because DRL-Cloud receives user requests that can raise randomness and data centers' resource utilization status can be formulated as MDP. State space and action space are defined in both stages of RP-TS processor, both of which are large but finite.
- **Fast convergence and high adaptability.** The proposed DRL-Cloud is fully parallelizable to training algorithm, which empowers our system with robustness, efficiency and ability of steady evolvement. Utilization of training techniques such as experience replay and target network, makes DRL-Cloud converge in less than 0.5 second, and gives it high adaptability and low runtime.
- **Remarkably low runtime and low energy cost.** Compared to FERPTS, one of the state-of-the-art methods that considers historical resource allocation and current server utilization, DRL-Cloud achieves up to 3X energy cost efficiency improvement while maintaining up to 2X lower user request reject rate (hard deadline violation rate) and up to 92X runtime reduction. And compared to the Round-robin method, which is known for the remarkably low runtime, DRL-Cloud achieves up to 12X runtime reduction, 2X energy cost efficiency improvement and rejects up to 15X less user requests.

II. System Model for DRL-Based Energy Cost Minimization in Cloud Computing

The system model (as shown in Figure 1) is introduced in this section, which includes user workload model, cloud platform model, energy consumption model, and price model.

A. *User Workload Model*

In this paper, the entire user workload is composed of a number of *jobs* (i.e., user requests), each of which contains several *tasks* with dependencies.

1) Job Characteristics: Directed Acyclic Graphs (DAGs) are used to model jobs. The entire user workload of U jobs is represented as a collection of U disjoint DAGs: $\{G_1(N_1, W_1), ..., G_U(N_U, W_U)\}$. A DAG $G_u(N_u, W_u)$ $(u \in [1, U])$ contains N_u vertexes and W_u edges. Each vertex ${}^u_n\phi$ $(n \in [1, N_u])$ represents a single task, and each edge ${}^u w(i, j)$ represents the amount of data that needs to be delivered from parent task ${}^u_i\phi$ to child task ${}^u_j\phi$. Figure 1 presents an example of user workload model with multiple jobs. Other cloud frameworks that use similar task graph based workload model include Nephele [22] and Dryad [19].

2) Task Characteristics: For each task ${}^u_n\phi$, the requested VM types is denoted by ${}^u_nK$, and its estimated executable time is represented by ${}^u_nL$, which can be derived from Nephele, an approximation method [22]. Besides, user specifies hard deadline ${}^u_nT_{ddl}$ for task ${}^u_n\phi$, and the scheduled start time by CSP is ${}^u_nT_{start}$. Based on Admission Control Policy, if a task cannot be completed before given deadline by using infinite resource, i.e., a tight deadline is given, then this task should be rejected immediately. Also, according to SLA, one prerequisite should be met: ${}^u_nT_{start} + {}^u_nL \leq {}^u_nT_{ddl}$. The CSP supports V types of VMs $\{VM_1, ..., VM_V\}$. Each type VM_v $(v \in [1, V])$ is associated with a two-tuple parameter set $\{R^v_{CPU}, R^v_{MEM}\}$, which represents the required amount of CPU and memory, respectively. Similarly, each task ${}^u_n\phi$ is associated with a two-tuple parameter set $\{{}^u_nD_{CPU}, {}^u_nD_{MEM}\}$, which represents the required amount of CPU and memory by this task, respectively. Successful task execution requires sufficient resource. Hence, if task ${}^u_n\phi$ is allocated to VM_v, two prerequisites should be met: $R^v_{CPU} \geq {}^u_nD_{CPU}$ and $R^v_{MEM} \geq {}^u_nD_{MEM}$.

B. *Cloud Platform Model*

As shown in Figure 1, a CSP owns M servers $\{\psi_1, \psi_2, ..., \psi_M\}$, and nearby servers are clustered in a server farm. The number of server farms owned by CSP is denoted by F. Server farm F_f has M_f servers, i.e., $\sum_{f=1}^{F} M_f = M$. Servers farms are connected with each other through two-way high-speed channels, whereas servers within one server farm are connected through local channels. The cloud platform is modeled as an undirected graph, where each vertex represents

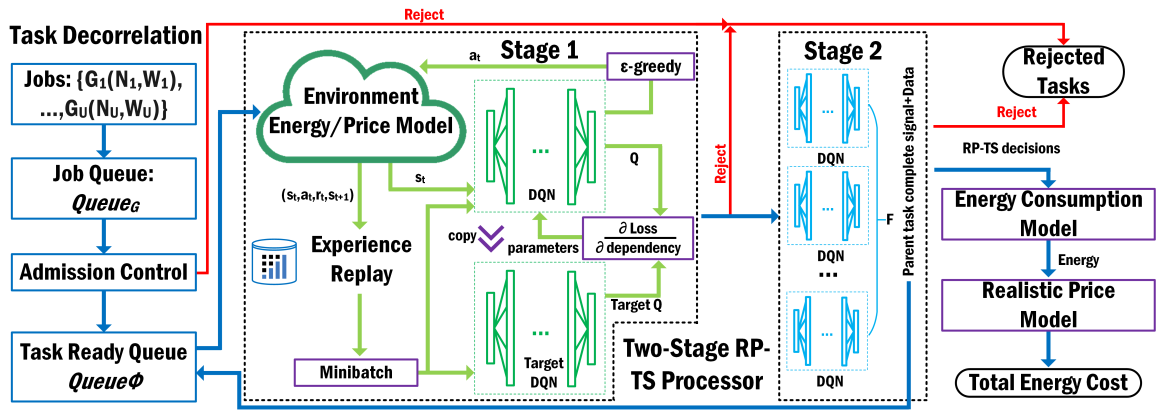

Fig. 2. The structure of the DRL-Cloud framework: the details of task decorrelation is described in Section III-A, and the details of the two-stage RP-TS processor is described in Section III and Section IV.

a server and each edge embodies the communication channel [23]. The bandwidth of the channel between server ψ_m and server $\psi_{m'}$ is represented by the weight of the edge $B(m, m')$ $(1 \le m, m' \le M)$. Note that data exchange in the same server does not incur extra delay, i.e., $B(m, m) = \infty$. Similar to tasks, each server ψ_m has a two-tuple parameter set: $\{C^m_{CPU}, C^m_{MEM}\}$, which represents the available amount of CPU and memory on server ψ_m. During operation, the state of server ψ_m is represented as $\{\lambda^m_1(t), \lambda^m_2(t), ..., \lambda^m_V(t)\}$, where $\lambda^m_v(t)$ is the number of v type VMs that are hosted on server ψ_m at time t. One prerequisite for VM allocation is that the total utilized computing resource (CPU, memory) must be less than the total available resource on each server during the entire operating process: $\forall t, \forall m, \sum_{v=1}^{V} \lambda^m_v(t) \cdot R^v_{CPU} \le C^m_{CPU}$ and $\sum_{v=1}^{V} \lambda^m_v(t) \cdot R^v_{MEM} \le C^m_{MEM}$.

C. Energy Consumption Model

The total power of server ψ_m at time t is composed of static power $Pwr^m_{st}(t)$ and dynamic power $Pwr^m_{dy}(t)$. Both static and dynamic power of server ψ_m are dependent on the CPU utilization rate $Ur^m(t)$ at time t, which is calculated as

$$Ur^m(t) = \frac{\sum_{v=1}^{V} \lambda^m_v(t) \cdot R^v_{CPU}}{C^m_{CPU}}. \tag{1}$$

$Pwr^m_{st}(t)$ is constant when $Ur^m(t) > 0$ and zero otherwise. $Pwr^m_{dy}(t)$ is linearly increased when $Ur^m(t)$ is under the optimal utilization rate Ur^m_{Opt}, and non-linearly increased otherwise [3], [9]. Additionally, different server may have different energy efficiency even under exactly identical utilization rate, this is captured by the parameter α_m, and β_m measures the power consumption increase of ψ_m. In this paper, we adopt the dynamic power model from [3] and $Pwr^m_{dy}(t)$ is calculated as

$$\begin{cases} Ur^m(t) \cdot \alpha_m, & Ur^m(t) < Ur^m_{Opt} \\ Ur^m_{Opt} \cdot \alpha_m + \left(Ur^m(t) - Ur^m_{Opt}\right)^2 \cdot \beta_m, & Ur^m(t) \ge Ur^m_{Opt}. \end{cases} \tag{2}$$

Note that the proposed method can accommodate other energy consumption models as well. The total power at time t is $Pwr_{ttl}(t) = \sum_{m=1}^{M} \left(Pwr^m_{st}(t) + Pwr^m_{dy}(t)\right)$.

D. Realistic Price Model

In this paper, we consider a realistic non-flat price model $Price\big(t, Pwr_{ttl}(t)\big)$ that is composed of a time-of-use-pricing (TOUP) component and a real-time pricing (RTP) component with inclining block rates (IBR) [20], [24–26]. The TOUP $TOUP(t)$ is dependent on the time of the day and is usually higher in peak usage time periods than off-peak time, in order to incentivize users to shift loads towards off-peak periods. The RTP price with IBR $RTP\big(Pwr_{ttl}(t)\big)$ increases by the total quantity consumed, which is calculated as

$$RTP\big(Pwr_{ttl}(t)\big) = \begin{cases} RTP^l(t), & Pwr_{ttl}(t) < \theta(t) \\ RTP^h(t), & Pwr_{ttl}(t) \ge \theta(t). \end{cases} \tag{3}$$

where $\theta(t)$ is a threshold, and $RTP^l(t)$ and $RTP^h(t)$ are the wholesale prices set by utility company on-the-fly. The total energy cost during whole RP and TS process is:

$$TotalCost = \sum_{t=1}^{T} Price\big(t, Pwr_{ttl}(t)\big). \tag{4}$$

III. DRL-Cloud: DRL-Based Cloud Resource Provisioning and Task Scheduling System

In this section, we present DRL-Cloud that minimizes power consumption and electric bills for CSPs with large-scale data centers. As shown in Figure 2, the proposed system first decorrelates the dependencies among tasks, then the decoupled tasks are sent to the two-stage RP-TS processor. After this process, final energy cost is calculated by using the realistic price model (as proposed in Section II-D).

A. Task Decorrelation

A CSP holds all jobs sent from users in *job queue* $Queue_G$. Within one job, child tasks are dependent on parent tasks with dependencies, and tasks that do not depend on other parent tasks or whose dependencies are satisfied are *ready tasks*. $Queue_G$ takes feedback (*parent task complete signal* and data that need to be delivered to child task) from the two-stage RP-TS processor as shown in task decorrelation part of Figure 2, which makes parent task become ready task. Job queue provides ready tasks from each job as input to *task ready queue* $Queue_\Phi$. Then CSP pops tasks from $Queue_\Phi$ and sends them to the two-stage RP-TS processor.

B. Two-Stage RP-TS Processor Based on Deep Q-Learning

The admission control policy takes place to filter the jobs that cannot be accomplished before hard deadline even with infinite computing resources. Otherwise, the first stage ($Stage_1$) of the two-stage RP-TS processor will allocate task to one of F server farms, and determine task start time ${}^u_n T_{start}$, and then continue processing or drop the job if SLA described in Section II-A2 is violated.

After allocating task into server farm F_f ($f \in [1, F]$), the second stage ($Stage_2$) of the processor takes responsibility of choosing exact server ψ_m to run task ${}^u_n\phi$. When task is completed, $Stage_2$ sends parent task complete signal and data to job queue (as feedback as described in Section III-A). Setups of the proposed deep Q-learning-based two-stage RP-TS processor is described as follows:

1) Action Space: For server farm level, DQN in $Stage_1$ is responsible for choosing a server farm from F farms, and determines start time ${}^u_nT_{start}$ to be one of the possible time, so action space for $Stage_1$'s DQN can be represented as $A_{Stage_1} = \{F_1^{T_1}, ..., F_F^{T_T}\}$. For server level, DQN in $Stage_2$ selects exact server within server farm F_f, so action space for $Stage_2$'s DQN can be represented as $A_{Stage_2} = \{\psi_1, ..., \psi_{M_f}\}$.

2) State Space: The optimal action is determined based on current observation x, which is the combination of current server observation x_{server} and current task observation x_{task}. Current server observation x_{server} describes available CPU C^v_{CPU}, memory C^v_{MEM} of requested VMs on server, whereas x_{task} is comprised of requested CPU R^v_{CPU}, memory R^v_{MEM} of several types of VM and task deadline. Therefore, state is a sequence of actions and observations, $s_t = x_1, a_1, x_2, a_2, ..., a_{t-1}, x_t$, and is input to the proposed deep Q-learning-based two-stage RP-TS processor. The processor then learns optimal allocation strategies depending on these sequences. All sequences are assumed to terminate in finite steps, which leads to a large but finite semi-Markov decision process (SMDP) and each sequence is a distinct state. The proposed processor chooses action based on current state and get a reward from environment, i.e., energy cost, and changes system state in the meantime, then uses Q value of action-reward pair to train DQN to "maximize" long-term reward, i.e., minimize long-term energy cost in our case.

3) Reward Function: The goal of the two-stage RP-TS processor is to minimize long-term energy cost by taking a sequence of actions. After taking action a_t at current state s_t, system will evolve into a new state s_{t+1} and receive a reward r_t from the environment, which is energy cost increase of action a_t, i.e., current energy cost minus previous energy cost (at time T^{pre}_{start}). For $Stage_1$, reward function can be calculated as price increase:

$$r_{Stage_1} = Price\left({}^u_nT_{start}, Pwr^{F_f}_{ttl}({}^u_nT_{start}) - Pwr^{F_f}_{ttl}(T^{pre}_{start})\right). \tag{5}$$

Similarly, reward function in $Stage_2$ can be calculated as:

$$r_{Stage_2} = Price\left({}^u_nT_{start}, Pwr^{\psi_{m}f}_{ttl}({}^u_nT_{start}) - Pwr^{\psi_{m}f}_{ttl}(T^{pre}_{start})\right). \tag{6}$$

C. Semi-Markov Decision Process (SMDP) Formulation

As the aforementioned state transition, we formulate the energy cost minimization problem of cloud resource allocation into SMDP problem. In Q-learning procedure, DRL agent's ultima goal is to maximize value function $Q(s, a)$, which specifies what is optimal in long-run. In other words, the value of a state is the total amount of reward that an agent can expect to accumulate for the future, starting from that state [27]. Optimum action-value function $Q^*(s, a)$ after seeing sequence s and taking action a is the maximum expected achievable reward by following any policy, which obeys Bellman equation is defined as:

$$Q^*(s, a) = E_{s'}[r + \gamma \max_{a'} Q^*(s', a')|s, a] \tag{7}$$

Q-learning is proven to have the ability to achieve optimum policy under SMDP environment and can get stunning results even in non-stationary environment [28]. In this work, we update value estimates as follows:

$$\begin{aligned} Q_{t+1}(s_t, a_t) &= Q_t(s_t, a_t) + \\ &\eta\left(r_{t+1} + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\right) \end{aligned} \tag{8}$$

where learning rate $\eta \in (0, 1]$ and discount factor $\gamma \in [0, 1]$.

IV. Deep Q-Learning Algorithm for DRL-Cloud With Experience Replay

The algorithm for training DQN is modified from standard Q-learning by using experience replay and target network to make it suitable for training large neural network with high speed of converging.

A. Training Details for Deep Q-Networks

Experience Replay: At each time-step of inner loop in Algorithm 2, store transition (s_{t+1}, a_t, r_t, s_t) into a replay memory Δ, and apply Q-learning updates to minibatch of experience which is selected randomly from the pool of stored samples. This approach is superior to standard Q-learning in several ways. First, data efficiency is higher because each step of experience is potentially replayed many times in many weight updates. Second, learning from randomly selected experiences instead of sequential experience breaks the correlation of learning data and reduces the variance of update, which provides learning procedure with higher efficiency. Third, behavior distribution of randomly selected minibatch of experience is averaged over previous states, which makes learning procedure stable and filters oscillations or divergence in the parameters.

Target Network: A separate neural network (target network) is used in DQN to generate target Q value in Q-learning update. Target network has same structure with evaluation network, but has different parameters. More specifically, parameters of target network are cloned from evaluation network in every ζ steps. This modification overmatches standard Q-learning by adding delay between the time an update effects Q and the time an update effects target, which eliminates divergence and oscillations even more.

Exploration & Exploitation: Exploration and exploitation is a puzzle about focusing on exploring new, indeterminate actions or spending time to exploit existing outstanding policy. The strategy we use in this work is $\epsilon-greedy$ with decreasing ϵ, that is with probability ϵ (start with large ϵ) choose a random action, otherwise choose the action with highest Q-value, and decrease ϵ over time to a minimum value. This strategy explores actions randomly at the beginning and settles down to a smaller exploration rate, which makes the DRL agent to exploit existing known-good policy to return high reward more often.

Algorithm 1: Control Algorithm for DRL-Cloud

```
1  Initialize realistic price model
2  Initialize environment and deep Q-network $DQN_{Stage_1}$
3  Run $DQN_{Stage_1}$ and store user request allocation
4  Initialize environment and deep Q-network $DQN_{Stage_2}$
5  for $f = 1, F$ do
6      for $\tau = 1, T$ do
7          Run $DQN_{Stage_2}$ and store user request allocation
8      end
9  end
10 Calculate final user request allocation matrix, i.e., $Ur$ for every server
11 Calculate final energy consumption $Pwr_{ttl}$ and electric bill $TotalCost$
12 return $TotalCost$
```

Algorithm 2: Deep Q-Learning for DRL-Based RP-TS With Experience Replay

```
Initialize replay memory $\Delta$ to capacity $\Omega$
Initialize action-value function $Q$ with random weights $\delta$
Initialize target action-value function $\hat{Q}$ with weights $\delta' = \delta$
for $episode = 1, E$ do
    Reset cloud server environment to initial state
    Initialize sequence $s_1 = \{x_1\}$
    for $t = 1, T$ do
        With probability $\epsilon$ choose a random action $a_t$
        otherwise choose $a_t = argmax_a Q(s_t, a; \delta)$
        Execute action $a_t$ and observe next observation $x_{t+1}$, reward
          $r_t$, and reject signal
        if $reject = 1$ then
            Run DQN again to get a new action $a'_t$
            if $a_t \neq a'_t$ then
                Replace $a_t$ with $a'_t$
            end
        end
        Set $s_{t+1} = s_t, a_t, x_{t+1}$
        Store transition $(s_{t+1}, a_t, r_t, s_t)$ in $\Delta$
        Sample random minibatch of transitions $(s_{j+1}, a_j, r_j, s_j)$
          from $\Delta$
        $target_j = \begin{cases} r_j, \text{if episode terminates at step j+1} \\ r_j + \gamma max_{a'} \hat{Q}(s_{j+1}, a'; \delta'), \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(target_j - Q(s_j, a_j; \delta))^2$
        Every $\Gamma$ steps, train evaluation network, decrease $\epsilon$
        Every $\zeta$ steps, copy $Q$ to $\hat{Q}$
    end
end
return All actions $a_t$
```

B. System Control Algorithm and the Two-Stage RP-TS Processor Algorithm with Experience Replay

Control algorithm in this work is shown in Algorithm 1, which implements the hierarchical structure described in Section III-B. And the two-stage RP-TS processor algorithm is shown in Algorithm 2.

V. Experimental Results

A. Experiment Setup

Three baselines are used to compare with DRL-Cloud:

- **The Greedy Method:** The CSP tries every option to find the assignment that yields the minimum energy cost increase. The CSP rejects tasks according to SLA.
- **The Round-Robin (RR) Method:** The CSP assigns each task in circular order. If the current assignment violates SLA, the scheduler will try the following options until non-violation. A task will be rejected by the CSP if it is rejected by all possible assignments.
- **FERPTS [9]:** one contemporary algorithm, that is aware of historical decisions and current scheduling of other tasks with the introduction of congestion concept.

Comparison is based on three indicators: energy cost, runtime and reject task number during the whole RP-TS process. We conduct two sets of experiments: one set on small-scale problems with $3,000$ to $5,000$ user requests and 100 to 300 servers that are clustered into 10 server farms; the other on a large-scale problem with $50,000$ to $200,000$ user requests and 500 to $5,000$ servers in 10 to 100 clusters. Note that, in this paper we use a very large-scale configuration for user workload model and cloud platform model[1]. We adopt the price data from [29], [30], where we consider that utility announces the price 24 hours ahead. Note that price is an input to the DRL-Cloud framework, which can accommodate on-the-fly price changes, e.g., price policy that updates every 10min. We use real user workload trace from Google cluster-usage traces [31], which represents 29 days' worth of cell information from May 2011, on a cluster of about 12.5k machines. Task dependencies in user workload model and the amount of resources in VMs and servers in cloud platform model are randomly generated, whereas information about the number of tasks in each job, task requirements of CPU and memory is retrieved from real data.

[1]Up to 180 servers and $10,000$ tasks in [3], up to 150 servers and 280 tasks in [9], up to 200 servers and $95,000$ tasks in [18].

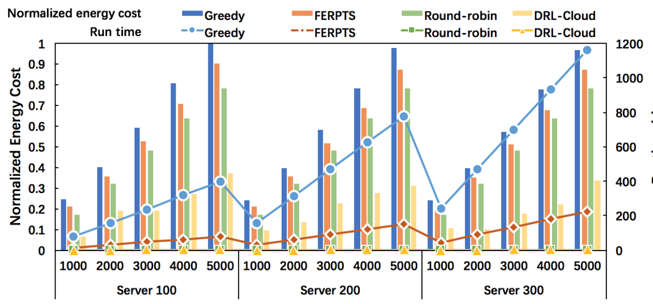

Fig. 3. Runtime and energy cost comparisons with baselines for small scale workload and platform configuration. Energy cost is normalized with regard to the energy cost of Greedy method for 100 servers and 5,000 requests.

For deep Q-learning, learning rate $\eta = 0.1$, discount factor $\gamma = 0.9$, and ϵ is decreased from 0.9 by 0.05 in each learning iteration. All parameters are chosen from commonly used range [11], [32] by giving relatively optimal performance. All simulation experiments are conducted in Python environment with TensorFlow [33] on a MacBook Pro (2012) with 2.6 GHz Intel Core i7 processor, 8GB 1600MHz DDR3 memory.

B. Experiments on Small-Scale Workloads and Platforms

Runtime and energy cost comparisons with three baselines are shown in Figure 3. One can observe that DRL-Cloud consistently outperforms the three baselines in different scenarios. Compared to the Greedy baseline, DRL-Cloud achieves up to 4X energy cost efficiency improvement and 480X runtime reduction. Compared to RR, DRL-Cloud outperforms with up to 3X less energy cost while rejects 3X less tasks on average, but uses 91.42% more runtime. This is because RR takes incoming task and allocates it immediately without decision making process. This gives RR remarkably low runtime when server number is small and decisions can be made in few tries. Compared to FERPTS, DRL-Cloud achieves up to 3X energy cost efficiency improvement and 92X runtime reduction. To be noticed, RR outperforms FERPTS in terms of energy cost because RR allocates tasks evenly and rejects more tasks due to hard deadline violation, and rejected tasks are not calculated in energy cost.

C. Experiments on Large-Scale Workloads and Platforms

Both FERPTS and Greedy use much more than 30 minutes for the large-scale experiment, so only RR is scalable to be used for large-scale comparison with DRL-Cloud. As shown in Table I, compared to RR, DRL-Cloud achieves up to 225% energy cost improvement, and rejects 253% less tasks in the meantime. Compared with RR, DRL-Cloud results in higher runtime when server number is 500 to $2,000$, but results in lower runtime when server number is larger (when server number and task number are $5,000$ and $200,000$, DRL-Cloud achieves 144%, 218% and 249% runtime reduction, energy cost efficiency improvement and reject rate reduction, respectively). This is because in RR, tasks are dispatched evenly in servers; servers are tried in turn when current server is overloaded. This process does not need much time when server/task number are small and existing servers can accept

TABLE I
COMPARISON OF ENERGY COST, RUNTIME AND REJECT TASK NUMBER BETWEEN DRL-CLOUD AND ROUND-ROBIN

Problem Scale		Runtime (s)		Reject Rate	Energy Cost
Server	Task	RR	DRL-Cloud	Improvement	Improvement
500	50,000	5	29	248.48%	222.97%
	100,000	11	60	250.76%	200.79%
	200,000	24	24	252.64%	199.71%
2,000	50,000	18	30	250.32%	225.20%
	100,000	37	61	247.93%	225.04%
	200,000	78	128	250.76%	213.26%
5,000	50,000	48	30	247.41%	220.60%
	100,000	97	64	248.64%	218.71%
	200,000	196	135	249.17%	218.16%

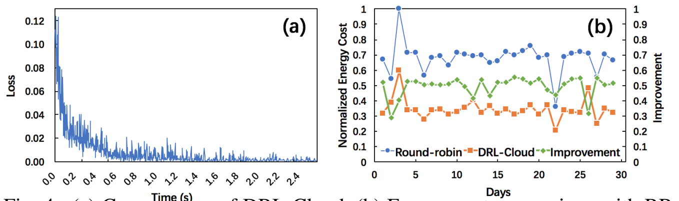

Fig. 4. (a) Convergence of DRL-Cloud. (b) Energy cost comparison with RR in long-run (29 days) on large scale workloads and platform configuration.

or reject incoming tasks within few tries. However, it would take longer time when server/task number are large.

D. Long-Term Experiments and Convergence

In this section, long-term comparison (within one month) of RR and DRL-Cloud with $5,000$ servers and $50,000$ tasks per day is experimented to test adaptability of changing user pattern in long term. The result is shown in Figure 4(b). Compare to RR, DRL-Cloud achieves 2X energy cost efficiency improvement, 2X runtime reduction and 3X reject rate reduction on average. Average runtime of RR and DRL-Cloud are 58.40s and 30.43s respectively. To be noticed, runtime improvement and reject rate improvement of DRL-Cloud during 29 days is up to 12X and 45X. Convergence situation is shown in Figure 4(a), we can see that DRL-Cloud converges fast, and this is because of the training techniques we used in Section IV.

VI. CONCLUSION

We presented DRL-Cloud, a novel DRL-based system with two-stage resource provisioning and task scheduling to reduce energy cost for cloud service providers with large-scale data centers and large amounts of user requests with dependencies. DRL-Cloud is highly scalable and highly adaptable compared to the state-of-the-art methods, and the training algorithm converges fast, thanks to the training techniques in our two-stage RP-TS processor such as experience replay, target networks as well as exploration and exploitation. Compared with one of the contemporary algorithms that has stunning energy cost efficiency and low task reject rate: FERPTS, DRL-Cloud achieves up to 320% energy cost efficiency improvement while maintaining lower task reject rate on average. For a large-scale problem with $5,000$ servers and $200,000$ task, and compared to the round-robin baseline which offers extraordinary runtime, DRL-Cloud achieves DRL-Cloud achieves up to 144% runtime reduction, 218% energy cost efficiency improvement and 249% lower reject rate.

REFERENCES

[1] J. Li *et al.*, "Negotiation-based resource provisioning and task scheduling algorithm for cloud systems," in *Quality Electronic Design (ISQED), 2016 17th International Symposium on*. IEEE, 2016, pp. 338–343.

[2] J. W. Rittinghouse and J. F. Ransome, *Cloud computing: implementation, management, and security*. CRC press, 2016.

[3] Y. Gao *et al.*, "An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems," in *Hardware/Software Codesign and System Synthesis*. IEEE, 2013, pp. 1–10.

[4] P. Delforge, "Americas data centers consuming and wasting growing amounts of energy," *Natural Resource Defence Councle*, 2014.

[5] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[6] P. Wieder *et al.*, *Service level agreements for cloud computing*. Springer Science & Business Media, 2011.

[7] Q. Zhang *et al.*, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud," in *Distributed Computing Systems (ICDCS)*.

[8] Y. Xiaoguang, C. Tingbin, and Z. Qisong, "Research on cloud computing schedule based on improved hybrid pso," in *Computer Science and Network Technology (ICCSNT), 2013 3rd International Conference on*. IEEE, 2013, pp. 388–391.

[9] H. Li *et al.*, "Fast and energy-aware resource provisioning and task scheduling for cloud systems," in *Quality Electronic Design (ISQED), 2017 18th International Symposium on*. IEEE, 2017, pp. 174–179.

[10] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[11] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[12] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[13] Z. Li *et al.*, "Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *International Conference on Computer Design (ICCD)*. IEEE, 2016, pp. 678–681.

[14] A. Ren *et al.*, "Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017, pp. 405–418.

[15] Z. Li *et al.*, "Structural design optimization for deep convolutional neural networks using stochastic computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 250–253.

[16] J. Li *et al.*, "Towards acceleration of deep convolutional neural networks using stochastic computing," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 115–120.

[17] Z. Yuan *et al.*, "Softmax regression design for stochastic computing based deep convolutional neural networks," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 467–470.

[18] N. Liu *et al.*, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 372–382.

[19] M. Isard *et al.*, "Dryad: distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS operating systems review*, vol. 41, no. 3. ACM, 2007, pp. 59–72.

[20] A.-H. Mohsenian-Rad and A. Leon-Garcia, "Optimal residential load control with price prediction in real-time electricity pricing environments," *IEEE transactions on Smart Grid*, vol. 1, no. 2, pp. 120–133, 2010.

[21] J. Li *et al.*, "Negotiation-based task scheduling and storage control algorithm to minimize user's electric bills under dynamic prices," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE, 2015, pp. 261–266.

[22] D. Warneke and O. Kao, "Nephele: efficient parallel data processing in the cloud," in *Proceedings of the 2nd workshop on many-task computing on grids and supercomputers*. ACM, 2009, p. 8.

[23] Y. Xue *et al.*, "Fundamental challenges toward making the iot a reachable reality: A model-centric investigation," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 3, p. 53, 2017.

[24] J. Li *et al.*, "Negotiation-based task scheduling to minimize users electricity bills under dynamic energy prices," in *OnlineGreencomm*. IEEE, 2014, pp. 1–6.

[25] J. Abushnaf, A. Rassau, and W. Górnisiewicz, "Impact of dynamic energy pricing schemes on a novel multi-user home energy management system," *Electric power systems research*, vol. 125, pp. 124–132, 2015.

[26] J. Li *et al.*, "Cts2m: concurrent task scheduling and storage management for residential energy consumers under dynamic energy pricing," *IET Cyber-Physical Systems: Theory & Applications*, 2017.

[27] G. Zaccone, *Getting Started with TensorFlow*. Packt Publishing Ltd, 2016.

[28] S. J. Bradtke and M. O. Duff, "Reinforcement learning methods for continuous-time markov decision problems," in *Advances in neural information processing systems*, 1995, pp. 393–400.

[29] "Real-time hourly market electrical price." [Online]. Available: https://hourlypricing.comed.com/live-prices/

[30] A. Faruqui, "The ethics of dynamic pricing," *The Electricity Journal*, vol. 23, no. 6, pp. 13–27, 2010.

[31] Google cluster data. [Online]. Available: https://github.com/google/cluster-data

[32] A. L. Strehl *et al.*, "Pac model-free reinforcement learning," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 881–888.

[33] J. Allaire *et al.*, *tensorflow: R Interface to TensorFlow*, 2016. [Online]. Available: https://github.com/rstudio/tensorflow