Efficient Compute-Intensive Job Allocation in Data Centers via Deep Reinforcement Learning

Deliang Yi, Xin Zhou, Yonggang Wen, Rui Tan

Abstract—Reducing the energy consumption of the servers in a data center via proper job allocation is desirable. Existing advanced job allocation algorithms, based on constrained optimization formulations capturing servers' complex power consumption and thermal dynamics, often scale poorly with the data center size and optimization horizon. This paper applies deep reinforcement learning to build an allocation algorithm for long-lasting and compute-intensive jobs that are increasingly seen among today's computation demands. Specifically, a deep Q-network is trained to allocate jobs, aiming to maximize a cumulative reward over long horizons. The training is performed offline using a computational model based on long short-term memory networks that capture the servers' power and thermal dynamics. This offline training approach avoids slow online convergence, low energy efficiency, and potential server overheating during the agent's extensive state-action space exploration if it directly interacts with the physical data center in the usually adopted online learning scheme. At run time, the trained Q-network is forward-propagated with little computation to allocate jobs. Evaluation based on eight months' physical state and job arrival records from a national supercomputing data center hosting 1,152 processors shows that our solution reduces computing power consumption by more than 10% and processor temperature by more than 4℃ without sacrificing job processing throughput.

Index Terms—Job allocation, data center, energy efficiency, deep reinforcement learning.

1 Introduction

Energy efficiency is a key metric of a data center serving the endlessly growing computation demands. In the U.S., data centers consumed 70 billion kWh of electricity in 2014, which accounted for 1.8% of the country's total electricity consumption [1]. In Singapore, this ratio is up to 7% due to the tropical condition [2]. Despite the adoption of greener designs and operational improvements in recent years, the energy consumption of data centers will still increase due to the rise of hyper-scale data centers [3]. Continual energy efficiency improvement for data centers remains an urgent and strategic task.

High PUE (power usage effectiveness) with an average of 1.9 [4] was a major issue in the past decade. However, the latest facility designs have significantly reduced the PUE to 1.3 [5] and even 1.06 [6] for commissioned data centers. Therefore, in this paper, we focus on improving the energy efficiency of the computing infrastructure, i.e., to reduce the energy consumed by the servers while maintaining the job processing throughput. The approach to this goal will be perpendicular to the efforts of reducing PUE (e.g., by improving cooling system operation [7] or raising server room temperature setpoint [8]). The computing energy efficiency improvement will engender various benefits. First, the less heat generation by the servers reduces the possibility of processor frequency throttling that degrades the quality of service. Second, according to the Arrhenius equation [9], the failure rates of server components increase exponentially with the servers' internal temperatures. Thus, less heat generation implies servers' longer lifetimes and lower maintenance cost. Third, the reduced heat generation also lowers the workload and energy consumption of the cooling systems in maintaining the server room temperatures.

An important approach to improving computing energy efficiency is to properly allocate the computing jobs to the servers. With the emergence of big data-driven business, scientific research, and social services, the long-lasting and compute-intensive jobs (e.g., genomic data analysis, domain-specific optimization and simulations, deep learning, etc.) form a major portion of today's computation demands. Thus, efficient allocation approaches for compute-intensive jobs are important to the data centers providing such services. Various job scheduling algorithms have been proposed [10]–[13]. In cloud computing data centers, the computation can be allocated by migrating and consolidating the virtual machines that process massive and short jobs [10], [11]. However, these approaches are ill-suited for long-lasting and compute-intensive jobs because the migration will interrupt the job execution and also cause the overhead of moving data. Thermal-aware job allocation can be formulated as constrained optimization problems based on certain models capturing the power consumption and heat processes to minimize energy use [12], [13]. However, due to the complexity of the models, the algorithms solving these problems generally need extensive search and scale poorly with data center size and optimization horizon.

Recently, deep reinforcement learning (DRL), which is an important extension of the traditional reinforcement learning (RL) method, has been applied to various sophisticated online optimization problems with large solution spaces. In a traditional RL system, an *agent* interacts with an *environment* by iteratively sending *action* to the environment, monitoring the *state* of the environment, and assessing a scalar

• *D. Yi, X. Zhou, Y. Wen, R. Tan are with School of Computer Science and Engineering, Nanyang Technological University, Singapore.*
E-mail: {yideliang, zhouxin, ygwen, tanrui}@ntu.edu.sg

reward to guide the next action. The agent's objective is to maximize the cumulative reward. During the interactions, the agent builds a look-up table that can be used afterward to choose an action based on the state of the environment. RL approaches can be model-based [14] and model-free [15]. Model-free RL does not require an *a priori* model on the state transition of the environment; instead, it learns the reward dynamics at run time in the iterative interactions with the environment and eventually approaches to the optimal action strategy. However, the look-up table will have an extremely large size for the optimization problems with large state and action spaces, which may negatively affect the learning process and model performance. DRL addresses this issue by replacing the table with a deep neural network called deep Q-network (DQN). The DQN is trained during the agent-environment interactions to capture the optimal action strategy. The approach in this paper belongs to model-free DRL.

Allocating continually arriving jobs to a large number of servers in a data center is an optimization problem with a large solution space, since allocating a pending job to each server is a candidate solution. Applying DRL to the job allocation problem gives two key advantages. First, during the learning phase, DRL learns the end-to-end mapping from the data center state and the job allocation action to the resulted reward function that can be defined to capture one or more optimization objectives such as reducing energy consumption and server temperatures. Moreover, after an adequate exploration of the state-action space, the trained DQN will inherently encompass the job allocation strategy that maximizes the overall reward for a long-term period similar to the training period. Second, after the learning completes, the DRL-based job allocator that forward-propagates the DQN to allocate jobs has light computation overhead. Thus, different from the model-based job allocation that adopts explicit constrained optimization formulations and often scales poorly with data center size and optimization horizon, the DRL-based solution will be computationally lightweight after the training process is fully completed.

However, we face two major challenges in applying DRL to the job allocation problem. First, in the training phase, before DRL converges, to adequately explore the state-action space, it may attempt random and radical actions that may lead to significant deviations from the optimum and server overheating. Second, the training convergence time of DRL is in general proportional to the size of the state-action space [16]. For the considered job allocation problem, the size of the state-action space is in quadratic of the number of servers (cf. §3). Thus, the DRL may have unacceptably long training convergence time when the number of servers is large. To tackle these two issues, we adopt an offline training approach. Specifically, we construct a computational model based on neural networks to provide system state prediction capability. The computational model can be trained based on extensive system state and job arrival traces collected from the target data center. Then, the training of the DRL-based job allocator is performed offline driven by the computational model and real job arrival history. As a result, the real-world time for completing the training is no longer a concern. For instance, we can complete the training for Singapore's National Supercomputing Center (NSCC) hosting 1,152 processors within one day. In contrast, our simulations show that the online training needs about 40 days to converge. After the completion of the offline training, the DRL-based job allocator is commissioned to actuate in the physical data center.

We implement and evaluate our approach for NSCC. Specifically, based on system state and job arrival traces over a period of six months, we implement the computational model for predicting system state (including processors' temperatures and servers' power consumption) using long short-term memory (LSTM) networks and then train the DRL-based job allocator using the model. The training aims at maximizing a weighted sum of server power savings and processor temperature reductions. After that, we reuse the model to conduct simulations driven by real job arrival records over a period of 52 days. The evaluation shows that, with our DRL-based job allocator, the simulated data center can save more than 10% computing power and reduce the average processor temperature by more than 4°C without compromising job processing throughput, compared with several baseline approaches including a basic round-robin allocator and a short-horizon online optimizer.

The rest of the paper is organized as follows. §2 reviews related work. §3 describes the problem and overviews our approach. §4 and §5 present the designs of the computational model for system state prediction and the DRL-based job allocator, respectively. §6 presents evaluation results. §7 discusses several issues. §8 concludes this paper.

2 Related Work

Workload management in cloud computing data centers has been widely studied. It is often achieved by managing the virtual machines (VMs) that serve massive, frequent, and short computing jobs (e.g., web requests). In [11], [17], [18], the VMs are consolidated and/or migrated to reduce the number of active physical servers. Different from VM with migratability as its key advantage, the compute-intensive jobs considered in this paper can be dependent on massive data that may have high migration overhead. Thus, the VM management approaches are ill-suited for our problem.

A variety of computing job allocation and scheduling algorithms have been proposed. Early studies [19]–[21] schedule a collection of jobs known *a priori*. Differently, we consider jobs that arrive dynamically. Accumulating sufficient arrived jobs to run the scheduling algorithms will lead to undesirable job waiting times. Recent studies schedule computing jobs to achieve various objectives. In [22], the jobs are scheduled to minimize the average job waiting time under various resource constraints. In [23], the jobs are allocated among geographically distributed data centers to reduce the communication overhead among them. The above two studies [22], [23] fall short of considering the thermal effects and requirements. The studies [24]–[26] develop various constrained optimization formulations that explicitly address the thermal effects and aim at improving energy efficiency. The approach in [24] allocates the jobs to minimize the electricity bills while satisfying several constraints such as processing time, total electricity budget, and the number of servers, etc. Based on certain models describing the servers' power consumption behaviors and thermal processes, the approaches in [25], [26] use heuristic algorithms to reduce the power consumption of the data center. Due to the complexity of the thermal processes and server/facility power consumption behaviors, the constrained optimization formulations often lead to high-complexity solvers that cannot scale well with the size of the data center and the optimization horizon.

A recent study [27] applies RL techniques to allocate arriving jobs. Specifically, in the proposed two-tier framework, the *global tier* applies DRL to allocate jobs to the servers in a cluster to reduce server power consumption, whereas the *local tier* applies the traditional RL to adjust a single parameter of a server (i.e., a timeout before the server in the idle state goes to sleep) such that the overhead of server state transition and the resulted job latency can be reduced. The local tier builds an LSTM network to predict the arrival time of the subsequent job. The global tier of [27] and our proposed approach address a similar problem of allocating jobs to servers. However, they differ in the following aspects. First, the global tier of [27] follows the online learning scheme that may have undesirably long convergence time when the number of servers is large. Thus, the evaluation performed in [27] is limited to 30 servers. Differently, our approach adopts offline training that is not concerned with the convergence time in scaling with the number of servers. For instance, we can build a DRL-based job allocator for NSCC with 1,152 processors. In §6.4.2, we also conduct simulations to compare the online learning approach, which is the essence of [27], and our offline training approach for NSCC. Second, the study [27] does not consider the thermal effect of job allocation. The online training may cause server overheating due to DRL's random attempts. In contrast, our approach performs temperature prediction and integrates processor temperatures into the reward function. As shown in §6, our offline training scheme can effectively prevent overheating. Third, the server power consumption model in [27] is a simplistic binary model (i.e., peak and idle powers). In contrast, we build LSTM networks to capture the complex thermal and power consumption dynamics.

Another recent study [28] applies online DRL to allocate computing resources to arriving jobs such that the job latency is reduced. However, for compute-intensive jobs considered in this paper, the job latency is not a key concern. Different from [28], our approach aims to reduce the power consumption and heat generation of servers, which are important factors of the operating cost of a data center.

Our prior work [29] presented our approach of applying DRL for job allocation. Based on [29], we make two new contributions in this paper. First, we add job classification to the computational model and build a separate LSTM network for each class. Owing to the improved prediction accuracy, new evaluation shows that our approach achieves more computing power saving and processor temperature reduction. Second, we add experiments to measure the computation overhead of our approach and others approaches.

3 Problem Statement & Approach Overview

This paper aims at developing DRL-based job allocation mechanisms to achieve power saving and temperature reduction. This section states the problem and present the overview of the proposed approach.

3.1 Problem Statement

In this paper, we abstract the computing infrastructure in a data center as a collection of N processors denoted by $\{p_1, p_2, \ldots, p_N\}$. We assume that p_i has n_i ($n_i \geq 1$) processing cores. The utilization of p_i, denoted by u_i, is the average utilization of p_i's cores. Denote by t_i the temperature of p_i, by w_i the power consumption of p_i and the associated supporting devices (e.g., main memory, hard disc, etc.). Note that u_i greatly affects both t_i and w_i.

We consider compute-intensive and deadline-free jobs with the following characteristics. First, each job is a process executed on a single processor only and can use multiple cores on the processor to optimize multithreading performance. To meet this requirement, a large-scale computing task can be decomposed into multiple jobs that will be allocated to multiple processors. Second, each job is *compute-intensive* in that the cores used by the job will be fully utilized. Thus, each core should be exclusively used by a single job only, because otherwise the contention among multiple jobs for a shared core will increase overhead and cause inefficiency. Third, the jobs do not impose deadlines. The above characteristics well model many long-lasting computing tasks, such as those submitted to NSCC.

We define the data center's *system state* as follows.

Definition 1 (System state). The data center's system state is a vector consisting of all processors' utilizations, temperatures, power consumption, and the numbers of spare cores. Formally, the state $\mathbf{x} = [\mathbf{u}, \mathbf{t}, \mathbf{w}, \mathbf{c}] \in \mathbb{R}^{4N}$, where $\mathbf{u} = [u_1, \ldots, u_N]$, $\mathbf{t} = [t_1, \ldots, t_N]$, $\mathbf{w} = [w_1, \ldots, w_N]$, $\mathbf{c} = [c_1, \ldots, c_N]$, c_i is the number of spare cores on p_i.

To simplify the discussion, we consider a problem of allocating a single job at the front of a job queue to one of the N processors for execution. In §5.2, we will present the extension to allocating multiple jobs at the front of the job queue to one or more processors for execution. We define the *allocation action* as follows.

Definition 2 (Allocation action). The $\mathbf{a} = [a_1, \ldots, a_N]$ is the allocation action, where $a_i = 1$ or $a_i = 0$ means that the job is allocated to processor p_i or not. Thus, $\sum_{i=1}^{N} a_i = 1$.

Each job in the queue is described by the number of cores requested by the job. As the job is compute-intensive, based on the number of requested cores only, we can predict the job's impact on the utilization, temperature, and power consumption of any processor that the job is allocated to. This will be shown with more details in §4. Following the standard formulation of RL, the allocation action $\mathbf{a}$ is chosen to maximize the expected return $\mathbb{E}[R(k)]$ defined from the current time step k. Note that we discretize time into steps in assessing the return. Specifically, the return $R(k)$ is defined as the exponential average of future rewards, i.e., $R(k) = \sum_{\tau=0}^{\infty} \gamma^{\tau} r(k + \tau + 1)$, where $\gamma \in (0, 1)$ is a constant discount factor and $r(k)$ is a scalar reward in the system state $\mathbf{x}(k)$. The reward function $r(k)$ can be defined by the data center operator to drive the DRL toward the desired goal. §5 will present the detailed form of $r(k)$ used in our

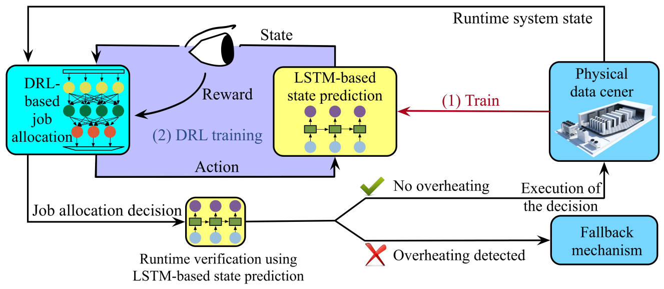


Fig. 1. Workflow of our offline training approach. The first step is to construct LSTM-based state prediction model; the second step is to train the DRL-based job allocator. The trained job allocator makes job allocation decisions based on the runtime system state.

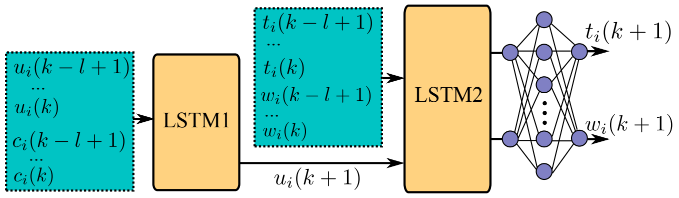


Fig. 2. A tandem of LSTM networks for system state prediction. The first LSTM network is to predict the utilization; the second LSTM network is to predict the temperature and power consumption.

evaluation. The jobs in the queue are allocated sequentially and continuously to the processors until the queue is empty.

3.2 Approach Overview

The allocation action $\mathbf{a}$ for the job at the queue front should be chosen according to the current state $\mathbf{x}(k)$. The size of the state-action space is $4N \times N$, where $4N$ and N are the dimensions of the state and action spaces, respectively. As N is often large (up to thousands) in typical data centers, it is infeasible to pre-compute the job allocation for every possible state to maximize the expected return. In this paper, we apply DRL to address the issue. During the learning phase, a DQN is trained to capture the allocation policy that maximizes the expected return. The job allocator trained with a sufficiently large number of system states is expected to give the optimal allocation at run time by forward-propagating the DQN based on the system state. Thus, the DRL-based job allocator will have low run-time overhead.

During the training phase, the DRL will extensively explore the state-action space by applying a large number of tentative actions and then learning the responses of the system as well as the impact on the reward. These tentative actions can lead to large deviations of the reward from the optimum. In particular, they may result in undesirable service quality degradation and server overheating. Fig. 1 illustrates the workflow of our offline training approach to address this issue. In the step (1), we collect a large amount of system state and job allocation records from the physical data center to train a computational model that can predict the future system states given the current state and a job allocation. In the step (2), we use the computational model and real job traces to drive the offline training of the DRL-based job allocator. The trained job allocator is commissioned to actuate in the physical data center. The following sections of this paper will present the detailed designs of the computational model and the job allocator.

The computational model can also be used to predict the consequence of executing a job allocation decision to detect potential server overheating. The decision that will not cause undesirable server overheating will be executed. Otherwise, a fallback mechanism that does not try to maximize the reward will be used to allocate the job. For instance, the job can be allocated to the coolest processor. Other advanced thermal-aware job allocators [30] can also be adopted. Note that the computational model may generate false negatives in predicting server overheating. In such cases, reactive measures such as throttling CPU frequency and increasing cooling capacity of the air cooling unit (ACU) will take effect to avoid thermal unsafety.

4 System State Prediction

This section presents the design of our system state prediction based on long short-term memory (LSTM) networks and the evaluation using real data traces.

4.1 Prediction Approach

Data center thermal state prediction is often performed using computational fluid dynamics (CFD) models. However, the CFD models generally require extensive calibration by domain experts for accuracy [31]. Moreover, CFD introduces notably high computation overhead, making it unsuitable for driving DRL. In this work, we build neural networks for the prediction.

Recurrent neural networks (RNNs) can well capture temporal correlations. LSTM [32] is an RNN architecture that addresses the vanishing and exploding gradient problems of conventional RNNs. Its outperforming performance has been demonstrated in a number of sequence prediction tasks such as speech recognition. Gated RNN [33] is an alternative with lower compute overhead but inferior prediction accuracy for sequences with fast dynamics, compared with LSTM. As the main purpose of the prediction system is for offline training of DRL, we choose LSTM. We design a bank of LSTM-based predictors to predict the next system state $\mathbf{x}(k+1)$ based on the current job allocation $\mathbf{a}$ and the latest $l \in \mathbb{Z}_{>0}$ measured system states, i.e., $\mathbf{x}(k-l+1), \ldots, \mathbf{x}(k)$. Specifically, the ith predictor in the bank predicts the next system sub-state corresponding to the ith processor p_i, which is defined by $\mathbf{x}_i(k+1) = [u_i(k+1), t_i(k+1), w_i(k+1), c_i(k+1)] \in \mathbb{R}^4$, based on $\mathbf{x}_i(k-l+1), \ldots, \mathbf{x}_i(k)$. Thus, the LSTM-based predictors in the bank predict the processors' states separately.

By analyzing the internal causality among the system state components, we design the predictor as a tandem of two LSTM networks for *utilization prediction* and *temperature and power prediction*, respectively. The tandem is illustrated in Fig. 2 and explained as follows.

Utilization prediction: For the ith processor p_i, the first LSTM network predicts the processor's next utilization $u_i(k+1)$ based on the past utilizations $u_i(k-l+1), \ldots, u_i(k)$, the past spare cores $c_i(k-l+1), \ldots, c_i(k)$, and the number of p_i's cores to be used by the allocated job, which is denoted by $m_i(k)$. Specifically, if $a_i = 0$ (i.e., the job is not allocated to p_i), $m_i(k) = 0$; otherwise, $m_i(k)$ is the number of cores requested by the job.

Temperature and power prediction: The second LSTM network predicts the processor's next temperature $t_i(k+1)$ and power consumption $w_i(k+1)$ based on the past temperatures $t_i(k-l+1), \ldots, t_i(k)$, past power consumption values $w_i(k-l+1), \ldots, w_i(k)$, and predicted utilization $u_i(k+1)$. Note that the output layer of the LSTM has two units for temperature and power consumption. After the LSTM, we add a fully connected layer with 100 units and then another output layer with two units to yield the final prediction.

The rationale of the above tandem design is that the two LSTM networks capture two causal processes with different time constants. The first LSTM network captures a *computation domain* process with a short time constant. Ideally, the processor utilization should respond immediately to the allocation of the job. Moreover, under the assumption of compute-intensive jobs made in §3.1, the next utilization can be predicted analytically as $u_i(k+1) = u_i(k) + \frac{m_i(k)}{n_i}$. The LSTM captures other realistic factors that are beyond the compute-intensive assumption such as non-full utilization of cores due to core swapping during job execution and I/O-induced waiting, etc. Differently, the second LSTM network captures a *physical domain* process with a larger time constant. That is, the responses of the processor temperature and power consumption to the utilization changes can be dynamic processes over time. Intuitively, our tandem design integrating the above knowledge about the system state evolution will outperform those based on a pure black box approach. For instance, another design option is to use a single LSTM network with the past system states as the input and the next system state as the output. Our evaluation in §6.2 shows that this single-LSTM design yields inferior prediction accuracy.

Real system state data traces collected from the target data center running simple/heuristic job allocation algorithms can be used as the training data to build the LSTM-based predictors. The two LSTM networks for each processor are trained separately. We use the mean squared error (MSE) between the prediction and the ground truth over a training batch as the loss function. Backpropagation is used to derive the network gradients and update the LSTM network parameters.

4.2 Evaluation for a Real Data Center

We apply the prediction approach described in §4.1 to NSCC hosting 1,152 processors in 16 racks. Each processor has 24 cores. The data center uses both air cooling and liquid cooling. It offers computing services to multiple scientific research organizations. Each job submitted by the user consists of an executable, the number of requested cores, and the expected execution time. The data center currently applies a round-robin job allocator. We collect the job allocation history from the data center for six months. The system state is sampled every ten minutes. Accordingly, we set the time step length for state prediction to be ten minutes. From our results of profiling the job arrivals in this data center, 97% of the intervals between two consecutive job arrivals are longer than ten minutes. Thus, with the 10-min time step length, each of most time steps has one or zero job arrival only. More profiles about the compute jobs in NSCC, including the compliance of the jobs to the compute-intensive assumption and the processor temperature/power dynamics, can be found in Appendix A.[1] We divide the dataset collected from NSCC into training, validation, and testing datasets that have 20000, 3126, and 2159 system states, respectively. We set $l = 3$.

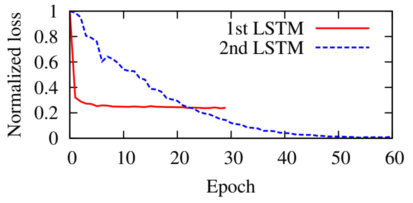


Fig. 3. Running loss of LSTM networks during training. The x-axis represents the index of the training epoch; the y-axis represents the loss function value that is normalized using the initial loss value.

First, we present the training and evaluation of the utilization prediction. The hidden size of the first LSTM is set to be 32. Training settings include: the learning rate is 0.01; mini-batch size is 32; training time is 30 epochs. The solid curve in Fig. 3 shows the running loss curve during the training of the first LSTM for a processor. It shows that the training converges after about ten epochs. The top part of Fig. 4 shows the ground truth and the prediction of the utilization of a processor. The root mean square error (RMSE) of the prediction is 3% only.

Second, we present the training and evaluation of the temperature and power prediction. The hidden size of the second LSTM is set to be 128. Training settings include: learning rate is 0.001; mini-batch size is 15; training time is 85 epochs. Fig. 3 shows the running loss curve during the training of the second LSTM for a processor. It shows that the training converges after about 50 epochs. Compared with the first LSTM, the second LSTM takes a longer time to converge. This is because the dynamics of the physical domain process addressed by the second LSTM is more complex than the job-utilization process addressed by the first LSTM. The middle and bottom parts of Fig. 4 show the temperature and power consumption predictions made by the tandem of the two LSTM networks for a processor and the corresponding ground truths. We can see that the predictions track the ground truths. The RMSEs of the temperature and power predictions are $1.24°\text{C}$ and $8.29\,\text{W}$.

In §6.2, we will present more evaluation results under various settings of the LSTM networks.

4.3 Improved Prediction Approach

Previous sections consider compute-intensive jobs that will fully utilize the cores assigned and lead to deterministic processor utilization. The first LSTM network of the tandem structure illustrated in Fig. 2 can capture the deviations from the assumption of compute-intensive jobs. In this section, we present an improved processor utilization prediction approach that can better deal with the deviations. Based on the extensive data traces collected from NSCC, we observe that the processor utilization traces have certain patterns. Thus,

1. Due to space limitations, all appendixes are omitted and can be found in the supplementary file of this paper.

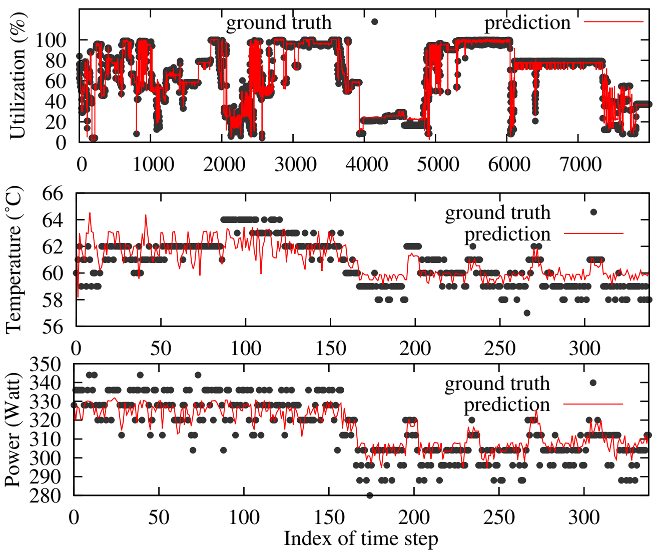


Fig. 4. Prediction results for a processor. Top: utilization; middle: temperature; bottom: power usage. The x-axis represents the index of time step; the y-axes of the three sub-figures represent the processor utilization, the processor temperature, and the power consumption.

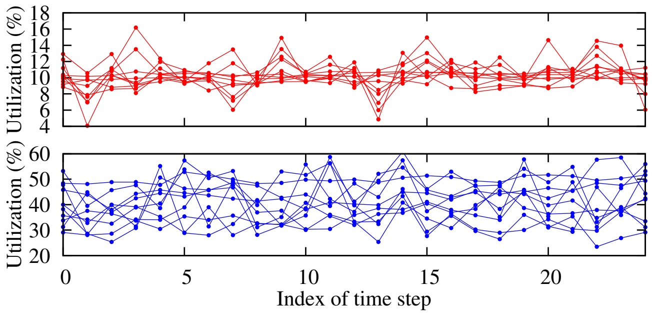


Fig. 5. Two example clusters of jobs. The x-axis represents the index of time step; the y-axes represent processor utilization.

we apply the expectation maximization (EM) algorithm to construct a Gaussian mixture model (GMM) based on the processor utilization traces with identical lengths of l time steps. The number of Gaussian distributions in the GMM, denoted by K, is predefined. Note that the EM algorithm is an supervised learning algorithm. Fig. 5 shows the processor utilization traces in two different clusters generated by the EM algorithm. We can see that they exhibit different patterns. In the first cluster, the utilization varies less over time. Differently, in the second cluster, the utilization varies more over time. Intuitively, if we construct separate LSTM networks for different clusters, the prediction accuracy can be improved.

The training procedure of these separate LSTM networks is as follows. First, the utilization traces contained in the training data are classified by the constructed GMM. Then, the utilization traces in the kth class are used to train an LSTM network (i.e., LSTM1-k) to predict the processor utilization. Note that the LSTM1-k also uses the spare core trace as an input, same as the setting for LSTM1 in Fig. 2. Fig. 6 shows the running loss trace of LSTM1-k for all three job clusters (i.e., $K = 3$) during the training. We can see that all three LSTM networks converge.

Fig. 7 illustrates the improved processor utilization prediction when being used for the offline training of DRL agent, in which the GMM and the LSTM1 networks have been well trained. To predict the utilization of a processor, the processor's utilization trace in the immediately past l time steps is classified by the GMM. Based on the classification result, the LSTM1 network corresponding to the class is selected to perform the utilization prediction. The inputs to the selected LSTM1 network include the utilization trace and the spare core trace. The output of the LSTM1 network is used as an input to the LSTM2 network shown in Fig. 2 to predict the processor's temperature and power consumption. In §6.3 and §6.5, we will evaluate the accuracy of the improved prediction approach and the overall performance of the DRL agent trained with the improved prediction.

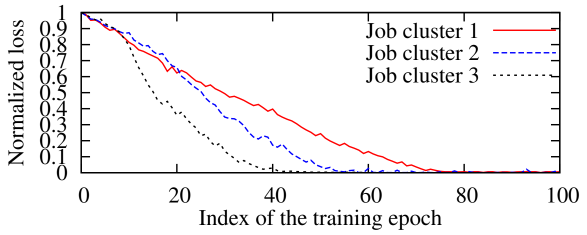


Fig. 6. Running loss of LSTM1-k for three job clusters during the training process. The x-axis represents the index of training epoch; the y-axis represents the loss function value that is normalized using the initial loss value. The total number of clusters K is three.

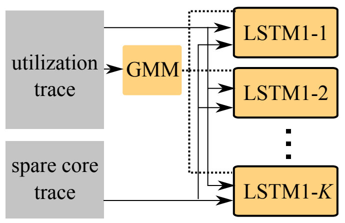


Fig. 7. Improved utilization prediction. Solid arrows represent data flows; dotted line represents selection signal. Based on the GMM's job classification result, an LSTM1 network is selected for utilization prediction.

5 DRL-Based Job Allocation

To formulate the job allocation as an RL problem, the RL's system and action are the data center's system state $\mathbf{x}$ and job allocation action $\mathbf{a}$ defined in Definitions 1 and 2. As discussed in §1, the traditional RL builds a look-up table $Q(\mathbf{x}, \mathbf{a})$ to assess the value of taking an action $\mathbf{a}$ given the current state $\mathbf{x}$. DRL builds a DQN to approximate $Q(\mathbf{x}, \mathbf{a})$ for problems with large state-action spaces. The DQN needs to be "deep" to well approximate $Q(\mathbf{x}, \mathbf{a})$ that can be highly complicated due to the complex coupling between the state, action, and value. This section presents the design of the DRL-based job allocators that allocate a single job (§5.1) or multiple jobs (§5.2) each time.

5.1 Design of DRL-Based Job Allocator

As discussed in §3.2, the training phase of DRL follows a trial-and-error paradigm. Thus, the training may attempt random and radical actions that will lead to significant deviations from the optimal values and server overheating. Thus, we use the system state prediction model presented in §4 as the environment to train the job allocator in an offline manner. The training can be driven by real historical job arrival records.

The data center operator can define the reward function to direct DRL to desired goals. For instance, we may focus on minimizing the total computing power consumption by defining the reward to be $r(k) = -\sum_{i=1}^{N} w_i(k)$. We can also direct DRL to outperform any baseline job allocator by defining the reward to be

$$r(k) = -\sum_{i=1}^{N}\Big(w_i^{\mathrm{DRL}}(k) - w_i^{\mathrm{BL}}(k)\Big) + \alpha\Big(t_i^{\mathrm{DRL}}(k) - t_i^{\mathrm{BL}}(k)\Big), \quad (1)$$

where the α is a non-negative weight; the superscripts DRL and BL denote the sub-states under the DRL and the baseline control schemes, respectively. With a smaller α, the DRL is driven more toward reducing the total power consumption of processors; with a larger α, the DRL is driven more toward reducing the average processor temperature. In §6, we will quantitatively evaluate the impact of the α setting on the performance of the DRL-based job allocator.

We now discuss the techniques that we use to speed up the convergence of the DRL training. Many deep learning algorithms assume that the training samples are independent. However, in the DRL paradigm, the online learning of the DQN will receive temporally correlated training samples as the system state transits in response to the actions. Thus, the online DRL may not converge. Fortunately, with the system state prediction model as the environment, we can generate the training samples computationally and perform offline training. With the offline training data, we use a technique called *experience replay* [34], [35] to deal with the training sample correlation issue mentioned earlier. Specifically, a number of temporally consecutive training samples compose an *episode*. In the inner loop of the training algorithm, a certain number of episodes are randomly selected from a pool of episodes to form a mini-batch. This technique effectively breaks the training sample temporal correlation and improves the efficiency of training data use as each episode may be used multiple times.

5.2 Allocating Multiple Jobs Each Time

§5.1 presented the DRL-based job allocator when a single job at the front of the job queue is allocated each time. This section discusses the extension to allocating J jobs each time, where J is a fixed integer. Now, the allocation action $\mathbf{a}$ becomes a $J \times N$ (0,1)-matrix, in which the (j,i)th element $a_{j,i} = 1$ or $a_{j,i} = 0$ represents that the jth job is allocated to processor p_i or not. Thus, $\sum_{i=1}^{N} a_{j,i} = 1$. When the number of pending jobs in the queue (denoted by J_Q) is less than J, we supplement the queue with $J - J_Q$ *dummy jobs* requesting zero cores. As we build a separate LSTM tandem for each processor, the system state prediction approach in §4 can also be applied for the case of allocating multiple jobs. The training of the DRL is same as that presented in §5.1. The main challenge caused by allocating multiple jobs is that the dimension of the action space significantly increases to N^J. In general, with a larger action space, the DRL will need more training data and longer training time for convergence. For instance, the offline training of the DRL-based job allocator with $J = 1$ and $J = 2$ for NSCC takes about 1 and 1.5 days, respectively. In §6, we will evaluate the performance improvement brought by allocating multiple jobs each time.

TABLE 1
Impact of hyperparameter settings on prediction accuracy.

Hyperparameters				Avg error		Accuracy	
learn. rate	input size	hidden size	batch size	t_i	w_i	t_i	w_i
1e-3	3	32	20	1.65	5.45	97.57%	97.21%
1e-3	**3**	**128**	**20**	**1.09**	**4.89**	**98.42%**	**97.54%**
5e-3	3	128	30	2.05	6.12	96.98%	96.87%
1e-3	5	128	30	1.87	5.34	97.25%	97.26%
1e-3	3	128	40	1.92	5.63	97.17%	97.12%
5e-4	3	64	30	1.43	5.24	97.89%	97.32%

[a]The units for temperature and power consumption are °C and W.
[b]The Adam optimizer and MSE loss function are used in training.

6 Performance Evaluation

In this section, we extensively evaluate our DRL-based job allocator against several baseline approaches.

6.1 Implementation and Evaluation Methodology

This section presents the implementation details of different approaches and the evaluation methodology.

6.1.1 Implementation of DRL-based job allocator

We build the DQN with three dense layers: the input layer admits the system state; the hidden layer has 24 nodes; the output layer consists of N units. The pending J jobs are allocated to the processors corresponding to the output-layer units giving the J highest values. The input and hidden layers use rectified linear units (ReLUs); the output layer uses a linear activation function. We choose MSE as the loss function and Adam as the optimizer. The Adam optimizer is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The epsilon of DRL, i.e., the probability that DRL randomly chooses an action, is set to be 0.2. Additionally, we set two hyperparameters called *epsilon decay* and *epsilon minimum* to be 0.999 and 0.01, respectively. During the training, the epsilon is decayed by multiplying it with the epsilon decay every job allocation until the epsilon minimum. Other settings include: learning rate is 0.001; mini-batch size is 5; discount factor λ is 0.99. By *DRL1* and *DRL2*, we refer to the DRL-based job allocators with J is 1 or 2, i.e., to allocate one or two jobs in each time step.

6.1.2 Implementations of baseline approaches

We implement several baseline job allocators as follows.

Round-robin (RR) job allocator: RR allocates the job at the front of the job queue each time to the processors in a round robin fashion. If the processor in turn does not have enough spare cores to admit the job, RR skips this processor and checks the next processor.

Job consolidator: Different from RR that tries to spread the jobs to the processors, the job consolidator tries to reduce the number of processors to serve the jobs. Specifically, it first identifies a set of eligible processors with enough cores that can admit the job at the front of the job queue. Then, it assigns the job to the processor whose post-allocation utilization would be higher than that of any other eligible processor. Note that the post-allocation processor utilization can be predicted by the system state prediction model.

Online optimizer: For an optimization horizon of h time steps, the online optimizer exhaustively searches for the allocation actions $\{\mathbf{a}(k), \mathbf{a}(k+1), \ldots, \mathbf{a}(k+h-1)\}$ such that the return $R(k) = \sum_{\tau=0}^{h} \gamma^{\tau} r(k+\tau+1)$ is maximized, where the setting for the discount factor γ is same as the DRL-based job allocator. We consider two variants:

Online-opt1 adopts a reward function of $r(k) = -\sum_{i=1}^{N} w_i^{\mathrm{OPT}}(k) + \alpha \cdot t_i^{\mathrm{OPT}}(k)$, where $w_i^{\mathrm{OPT}}(k)$ and $t_i^{\mathrm{OPT}}(k)$ are the computing power consumption and processor temperature under the online optimizer.

Online-opt2 adopts a reward function similar to the DRL-based job allocator: $r(k) = -\sum_{i=1}^{N} \left(w_i^{\mathrm{OPT}}(k) - w_i^{\mathrm{BL}}(k)\right) + \alpha \left(t_i^{\mathrm{OPT}}(k) - t_i^{\mathrm{BL}}(k)\right)$.

Both variants have an exponential complexity with respect to h, i.e., $\mathcal{O}(N^h)$. Thus, they scale poorly with h.

6.1.3 Evaluation methodology

We conduct trace-driven simulations to compare the approaches described in §6.1.1 and §6.1.2. In §4, we have constructed the system state prediction model using real data traces collected from NSCC. As discussed in §5.1, we use the model as the environment to train the DRL-based job allocator. In our evaluation, we also use the model to simulate the state evolution of the data center with a job allocator being evaluated. We drive the simulations using real job arrival records of the data center. Note that these records are not used in the training phases of the system state prediction model and the DRL-based job allocator. To show the DRL-based job allocator's robustness against the inherent randomness of its training process, we run 10 training-testing processes with the same hyperparameters driven by the same training and testing data. We report the average and standard deviation (s.d.) of the 10 runs' results.

6.2 Evaluation Results of System State Prediction

This section evaluates the accuracy of the system state prediction approach presented in §4.1 under various settings. Denoting by P_m and T_m the prediction and the true value, respectively, the prediction accuracy is computed as $1 - \frac{1}{M}\sum_{m=1}^{M} \frac{P_m - T_m}{T_m}$, where M is the number of data points. Table 1 shows the average prediction error and the prediction accuracy of processor temperature (t_i) and power consumption (w_i) under various combinations of hyperparameter settings. With the Adam optimizer and the hyperparameter settings highlighted by bold text in Table 1, the minimum average temperature and power prediction errors are 1.09°C and 4.89 W only. The corresponding prediction accuracies are 98.42% and 97.54%. These results show that our LSTM tandem can predict the processor temperature and power consumption accurately.

We also compare our LSTM tandem design with several alternative design options of (1) single LSTM network, (2) using stochastic gradient descent (SGD) optimizer instead of Adam optimizer for the LSTM tandem, and (3) using fully connected linear network instead of LSTM. Fig. 8 shows the accuracy of different designs in predicting processor temperature and power consumption. We can see that our design achieves the highest accuracy. Consistent with our discussion in §4.1, our LSTM tandem capturing knowledge about the system state evolution outperforms the single LSTM network that follows the black box design approach. The other design options of using SGD optimizer and linear network give worse prediction accuracy.

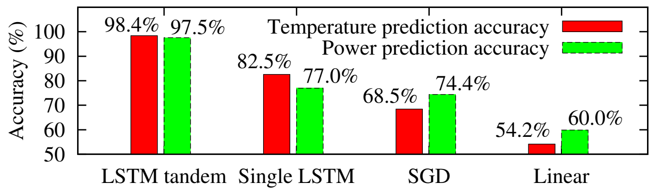


Fig. 8. Temperature and power prediction accuracy of different approaches. The x-axis represents the various approaches; the y-axis represents the accuracy in predicting processor temperature and power consumption that are differentiated by the bar groups.

TABLE 2
Hyperparameter settings and prediction accuracy for three clusters.

Hyperparameters				Avg error		Accuracy	
learn. rate	input size	hidden size	batch size	t_i	w_i	t_i	w_i
1e-3	3	128	30	0.46	2.59	98.43%	97.84%
5e-3	3	64	40	0.82	2.72	97.96%	97.98%
1e-3	3	128	30	0.69	2.79	98.63%	98.14%

[a]The units for temperature and power consumption are °C and W.
[b]The Adam optimizer and MSE loss function are used in training.

6.3 Evaluation Results of Improved State Prediction

This section evaluates the accuracy of the improved system state prediction approach presented in §4.3. The processor utilization traces in the training data are classified into three clusters by the EM algorithm. For each cluster, we train a separate LSTM network for utilization prediction. We follow the approach in §6.2 to optimize the settings of the hyperparameters. Table 2 shows the best test accuracies for the three clusters and the corresponding hyperparameters. Compared with the results shown in Table 1, the test accuracies for the first and the third clusters are higher than the best test accuracy in Table 1. For the second cluster, the test accuracy for temperature prediction is slightly lower than the best accuracy of 98.42% in Table 1; but the test accuracy for power consumption prediction is higher than that in Table 1.

6.4 Evaluation Results of DRL-based Job Allocator

6.4.1 Training convergence of DRL1

We adopt the reward function in Eq. (1) with RR and job consolidator as the baseline approaches, respectively.

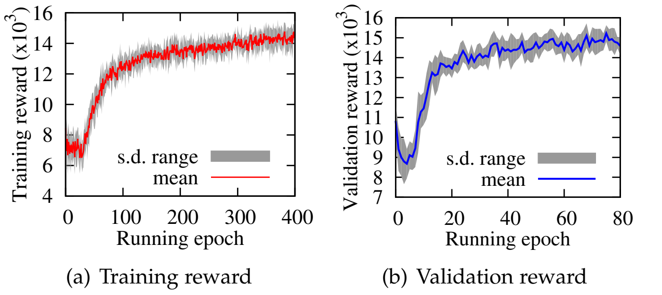


Fig. 9. Training and validation rewards of *DRL1*. Eq. (1) uses RR as baseline ($\alpha = 0.5$). Mean and s.d. are obtained from 10 runs. The x-axis represents the running epoch; the y-axis represents the reward.

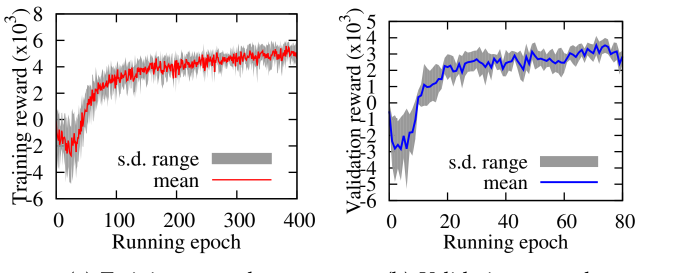


Fig. 10. Training and validation rewards of *DRL1*. Eq. (1) uses job consolidator as baseline ($\alpha = 0.5$). Mean and s.d. are obtained from 10 runs. The x-axis represents the running epoch; the y-axis represents the reward.

Fig. 9 shows the training and validation reward traces when Eq. (1) is instantiated with RR as the baseline. Note that we perform a validation epoch every five training epochs. The mean traces and the corresponding s.d. ranges are obtained from 10 runs of the training-validation process. We can see that both the training and validation rewards become flattened after about 200 training epochs. The positive rewards suggest that *DRL1* can be trained to outperform the RR approach. Fig. 10 shows the results when Eq. (1) is instantiated with the job consolidator as the baseline. The positive rewards suggest *DRL1*'s superior performance. By comparing Fig. 9 and Fig. 10, *DRL1* achieves larger rewards with respect to RR than the job consolidator. This is because that the job consolidator outperforms RR in terms of the reward concerning power and temperature reduction. In §6.5, we will show that the DRL-based job allocator trained with the improved system state prediction approach achieves more power saving and processor temperature reduction.

6.4.2 Convergence & temperature spikes of online learning

We conduct simulations to investigate the convergence time and processor temperature profile of DRL when it directly interacts with the physical data center to perform online learning. This online learning design is consistent with the approach described in [27]. The simulations are driven by real job arrival records collected from NSCC over a period of 52 days. During the 52 days, a total of 1,500 jobs arrived. On each simulated day, we train the DRL-based allocator for one epoch or 400 epochs based on the jobs that have arrived on that day and before. After the daily update of the DRL, we perform simulations driven by all the jobs and the system state predictor constructed in §6.2 to test the performance of the daily updated DRL-based job allocator. Figs. 11(a) and 11(b) show the processor temperature and server power consumption in the daily testing when one or 400 training epochs are performed for each update. As a comparison, we test *DRL1* that has been adequately trained during the offline learning phase. The offline training of *DRL1* takes about one day. Fig. 11(c) shows the testing result for *DRL1*.

The processor temperatures and power consumption in Fig. 11(a) are consistently higher than those in Fig. 11(c). This suggests that the online learning with one training epoch each day has not converged after 50 days. The slow online convergence is caused by the large state-action space and the inadequate training each day. Moreover, from Fig. 11(a), the servers experience spikes of processor temperature and power consumption during the testing. They are caused by the random attempts of DRL during the online learning. Differently, in Fig. 11(c), *DRL1* does not cause spikes of processor temperature and power consumption. From Fig. 11(b), the online learning with 400 training epochs each day needs about 40 days to converge. It also causes processor temperature spikes in the first 30 days.

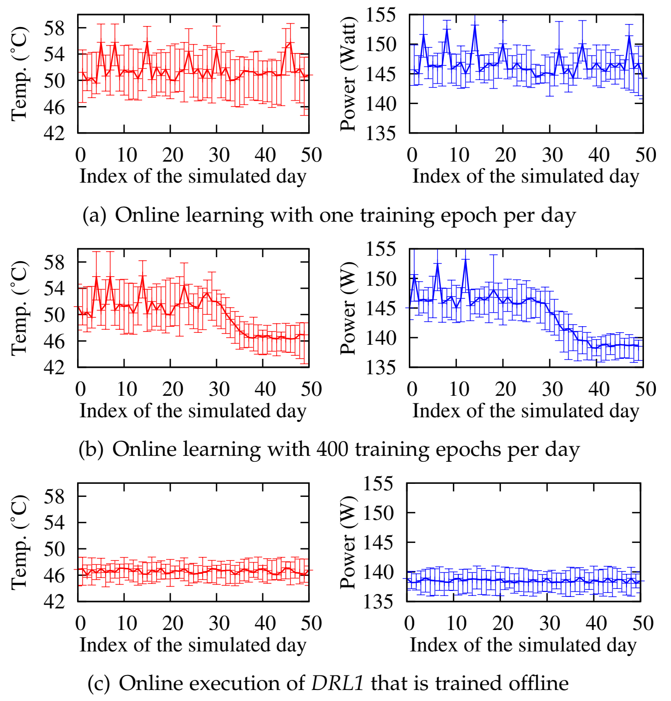


Fig. 11. Processor temperature & power consumption. Error bar represents min & max among 1,152 processors over testing data. The x-axis of (a), (b), and (c) represents the index of the simulated day; the y-axis of the left parts of (a), (b), and (c) represents the processor temperature; the y-axis of the right parts of (a), (b), and (c) represents processor power consumption.

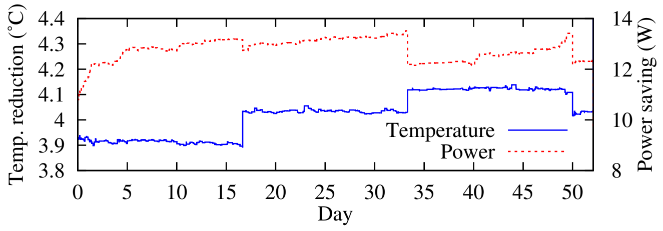
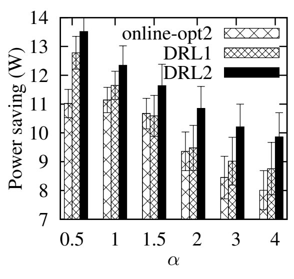


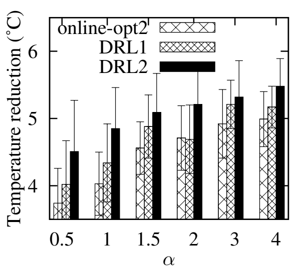
Fig. 12. Temperature reduction (left y-axis) and power saving (right y-axis) for a processor by *DRL1* with respect to RR. The x-axis represents the index of the simulated day. The α in Eq. (1) is 0.5.

6.4.3 Run-time temperature reduction and power saving

We conduct trace-driven simulations to evaluate the temperature reduction and power saving achieved by the DRL-based job allocators that are trained under the reward in Eq. (1) with RR as the baseline and different α and J settings. The simulations are driven by the real job arrival records used in §6.4.2, which span 52 days. Fig. 12 shows the temperature reduction and power saving for a certain processor when the simulated data center adopts *DRL1* ($\alpha = 0.5$) instead of RR. We can see that for this processor, *DRL1* reduces the processor temperature by about 4°C and power consumption by about 12 W.

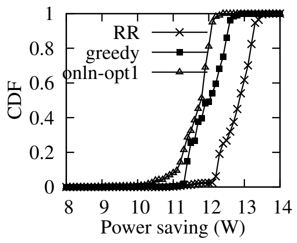


(a) Power saving

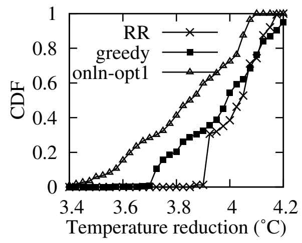


(b) Temperature reduction

Fig. 13. Impact of α on (a) temperature reduction and (b) power saving of a processor with respect to RR. The error bar represents s.d.; for *online-opt2*, the optimization horizon $h = 1$.



(a) CDF of power saving



(b) CDF of temperature reduction

Fig. 14. CDFs of (a) power saving and (b) temperature reduction by *DRL1* with respect to various baseline approaches. The y-axis represents the value of cumulative distributed function. The α in Eq. (1) is 0.5; for *online-opt1*, the optimization horizon $h = 1$.

We also investigate the impact of the weight α in Eq. (1) on the temperature reduction and power saving with respect to RR. The results for a certain processor are shown in Fig. 13. In Fig. 13(a), with smaller α, more power can be saved. The s.d. of the power saving increases with α. This is because, with a larger α, the power saving becomes a less important component in the reward function and more uncertain at run time. In Fig. 13(b), with a larger α, *DRL1* and *DRL2* achieve more temperature reduction. For the same reason aforementioned, the s.d. of the temperature reduction generally decreases with α. These results are consistent with our discussion in §5.1 regarding the impact of α on the training goal. Compared with *DRL1*, *DRL2* achieves more power savings and temperature reductions. This is because, compared with sequentially scheduling individual jobs, jointly scheduling two jobs each time will have a better chance to further increase the reward. However, *DRL2* requires 1.5x training time compared with *DRL1*.

We also apply the online optimizer *online-opt2* with the horizon $h = 1$ and the reward function defined with RR as the baseline. The power saving and temperature reduction achieved by *online-opt2* are also shown in Fig. 13. The impact of α on the power saving and temperature reduction is similar to that under the DRL solutions. Compared with the online optimizer, our DRL solutions (i.e., *DRL1* and *DRL2*) save more power and achieve larger temperature reduction. Note that, as evaluated in §6.4.4, *online-opt* with $h = 2$ has unacceptably long computation time (40 minutes) to allocate a job, rendering the approach unrealistic. Thus, we skip evaluating *online-opt* with $h \geq 2$.

We investigate the per-processor power savings for all the 1,152 processors in the data center that are achieved by

TABLE 3
Average relative power saving & temperature reduction achieved by *DRL1* with respect to baseline approaches.

α	Relative power saving (%)			Avg temp. reduction (°C)		
	RR	greedy	online-opt1	RR	greedy	online-opt1
0	9.89	9.45	9.28	3.75	3.58	3.34
0.5	9.31	9.16	9.09	4.08	3.94	3.71
1.0	8.51	8.22	8.31	4.46	4.01	3.89
1.5	7.72	7.45	7.42	4.95	4.30	4.02
2.0	6.87	6.81	6.72	4.75	4.38	4.17
3.0	6.57	6.43	6.35	5.31	4.84	4.52
4.0	6.33	6.28	6.29	5.25	5.01	4.87

$h = 1$ for online-opt1.

TABLE 4
Per-job computation latency.

Approach	Mean(s)	S.d.(s)
online-opt1 ($h = 1$, GPU)	2.32	0.13
online-opt1 ($h = 2$, GPU)	2654	125
DRL (GPU)	0.05	0.003
DRL (CPU)	0.47	0.006

DRL1 with respect to different baseline approaches over the simulated period of 52 days. Fig. 14(a) shows the cumulative distribution function (CDF) of the per-processor power savings when $\alpha = 0.5$. We can see that *DRL1* saves the largest power over RR. It also saves more than 10 W over the *online-opt1*. Fig. 14(b) shows the CDF of the temperature reduction. We can see that, compared with the baseline approaches, *DRL1* reduces the processor temperature by 3.4°C to 4.2°C. Table 3 shows the average relative power saving and temperature reduction with respect to various baselines under different α settings. The average relative power saving is computed as $\frac{1}{M}\sum_{m=1}^{M}\frac{w_m^{\mathrm{BL}}-w_m^{\mathrm{DRL}}}{w_m^{\mathrm{BL}}}$, where the superscripts DRL and BL denote the per-processor power consumption (w) under the *DRL1* and baseline job allocators, respectively; the M is the total number of data points for all processors in the 52 days. We can see that, for any baseline, the relative power saving achieved by *DRL1* decreases with α; differently, the average temperature reduction achieved increases with α. These results are consistent with intuition and the result in Fig. 13 for a single processor. From Table 3, compared with the baseline approaches, *DRL1* saves computing power by more than 9% and reduces processor temperature by more than 3°C.

6.4.4 Computation overhead

We assess the computation overhead of different job allocation approaches. The job allocators are executed on a workstation computer equipped with an Xeon E3-1220 CPU, 32GB memory, and a Tesla K40c 12GB GPU. The forward propagation computations of the LSTM and Q-network use the GPU. Table 4 shows the computation time of different approaches in allocating a single job. We can see that the *online-opt1* with $h = 1$ takes 46x computation time compared with *DRL1*. This is because *online-opt1* needs to forward-propagate 1,152 LSTM tandems for each candidate allocation action among all the 1,152 candidates. When $h = 2$, the *online-opt1* takes more than 40 minutes to allocate a job, which is unacceptable.

In addition, we have also evaluated the job processing throughput of our approach and various baseline approaches. The results can be found in Appendix B.

TABLE 5
Relative power saving & temperature reduction achieved by *DRL1* with improved state prediction with respect to baseline approaches.

α	Relative power saving (%)			Avg temp. reduction (°C)		
	RR	greedy	online-opt1	RR	greedy	online-opt1
0	11.52	11.05	10.89	4.88	4.56	4.32
0.5	10.85	10.63	10.26	5.54	5.23	5.08
1.0	10.42	10.18	9.97	5.75	5.23	5.08
1.5	10.25	10.02	9.76	6.02	5.84	5.61

$h = 1$ for online-opt1.

6.5 Job Allocator with Improved State Prediction

This section presents evaluation results of a DRL-based job allocator trained with the improved state prediction approach presented in §4.3. Table 5 shows the average relative power saving and temperature reduction achieved by *DRL1* with respect to various baselines under different α settings. The method of computing the average relative power saving is same as in §6.4.3. We can see that, for any baseline, the relative power saving achieved by *DRL1* decreases with α; differently, the average temperature reduction achieved increases with α. By comparing the results in Table 5 and Table 3, we can see that with the improved system state prediction, the trained DRL-based job allocator can achieve more power saving and processor temperature reduction.

7 Discussions

Adaptability to changes: Our approach requires a training phase, which is a one-time overhead but brings continued benefits of reduced computing power and processor temperatures. The training data can be readily obtained in today's data centers: the core utilization and temperatures can be recorded by various monitoring tools; the server power consumption can be recorded by smart racks and servers' built-in power meters. To adapt to the change of job patterns and servers' power/heat models due to aging, the DRL-based job allocator can be re-trained periodically. When there are new servers deployed, the LSTM network tandems trained for the existing servers of the same models as the new servers can be used to retrain the DRL agent. Thus, the DRL agent can quickly adapt to new server deployment. Other changes without adding or removing servers (e.g., servers re-layout on the racks) do not require retraining the DRL agent, because the system state prediction is performed at the granularity of individual servers.

Scalability to hyperscale data center (HDC): An HDC can be divided into multiple zones, where each zone hosts a similar number of processors as in this paper. Then, a DRL agent is trained for each zone. With this divide-and-conquer approach, the job allocation computation overhead for the HDC is proportional to the number of zones.

Policy gradient: This paper adopts DQN to build the DRL agent. Policy gradient approach is a major alternative for building DRL agent. Compared with DQN that deals with discrete action space, policy gradient can additionally handle continuous action space. Policy gradient often gives faster convergence rate than DQN, but has a tendency to converge to local optimums. As the job allocation problem has a discrete action space and convergence rate is not a key concern of the offline training of DQN, this paper chooses DQN. However, it is interesting for future work to extensively compare DQN and policy gradient for large-scale data center job allocation problem and beyond.

8 Conclusion

This paper applies DRL to allocate compute-intensive jobs to the servers in a data center. We first build a system state prediction model based on LSTM networks and then use the model to train the DRL-based job allocator. Our training approach avoids potential computing service quality degradation and server overheating. For a supercomputing data center with 1,152 processors, we build the LSTM networks and use real job arrival records over 8 months to train and test the job allocator. Results show that with our allocator, the simulated data center saves computing power by more than 10% and reduces processor temperatures by more than 4°C while maintaining job processing throughput.

Acknowledgment

This work is funded by National Research Foundation via Green Data Centre Research (GDCR) and Green Buildings Innovation Cluster (GBIC), administered by Infocommunications Media Development Authority and Building and Construction Authority, respectively.

References

[1] E. O'Shaughnessy, C. Liu, and J. Heeter, "Status and trends in the u.s. voluntary green power market (2015 data)," National Renewable Energy Laboratory, Tech. Rep. NREL/TP-6A20-67147.
[2] (2018) Singapore is top data center hub in SE Asia: report. [Online]. Available: https://bit.ly/2LEeV2B
[3] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, "United States data center energy usage report," Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-1005775, 2016.
[4] A. Sullivan. (2018) ENERGY STAR ™for data centers. [Online]. Available: https://bit.ly/2LdVUoC
[5] Datacenter Dynamics. (2018) Aliyun cools new china data center using lake water. [Online]. Available: https://bit.ly/2Oceifo
[6] Google Data Centers. (2018) Efficiency: How we do it. [Online]. Available: https://bit.ly/2O84KSz
[7] S. Greenberg, E. Mills, B. Tschudi, P. Rumsey, and B. Myatt, "Best practices for data centers: Lessons learned from benchmarking 22 data centers," *The ACEEE Summer Study on Energy Efficiency in Buildings*, vol. 3, pp. 76–87, 2006.
[8] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder, "Temperature management in data centers: Why some (might) like it hot," in *ACM SIGMETRICS*, 2012.
[9] JEDEC. (2018) Arrhenius equation (for reliability). [Online]. Available: https://bit.ly/2VjetJi
[10] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Efficient datacenter resource utilization through cloud resource overcommitment," in *INFOCOM*, 2015.
[11] F. Farahnakian, P. Liljeberg, and J. Plosila, "Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning," in *22nd Intl. Conf. Parallel Distrib. Netw. Process.*, 2014.
[12] Q. Tang, S. K. Gupta, and G. Varsamopoulos, "Thermal-aware task scheduling for data centers through minimizing heat recirculation," in *IEEE Intl. Conf. Cluster Comput.*, 2007.
[13] M. Polverini, A. Cianfrani, S. Ren, and A. V. Vasilakos, "Thermal-aware scheduling of batch jobs in geographically distributed data centers." *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 71–84, 2014.
[14] T. Wang, X. Bao, I. Clavera, and et al., "Benchmarking model-based reinforcement learning," *arXiv preprint arXiv:1907.02057*, 2019.
[15] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[17] Z. Peng, D. Cui, J. Zuo, Q. Li, B. Xu, and W. Lin, "Random task scheduling scheme based on reinforcement learning in cloud computing," *Cluster Computing*, vol. 18, no. 4, pp. 1595–1607, 2015.
[18] J. J. Jheng, F. H. Tseng, H. C. Chao, and L. D. Chou, "A novel vm workload prediction using grey forecasting model in cloud data center," in *Intl. Conf. Inf. Netw.*, 2014.
[19] Q. Tang, S. K. S. Gupta, and G. Varsamopoulos, "Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, 2008.
[20] T. Mukherjee, A. Banerjee, G. Varsamopoulos, S. K. Gupta, and S. Rungta, "Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers," *Computer Networks*, vol. 53, no. 17, pp. 2888–2904, 2009.
[21] L. Wang, S. U. Khan, and J. Dayal, "Thermal aware workload placement with task-temperature profiles in a data center," *The Journal of Supercomputing*, vol. 61, no. 3, pp. 780–803, 2012.
[22] T. T. Tran, M. Padmanabhan, P. Y. Zhang, H. Li, D. G. Down, and J. C. Beck, "Multi-stage resource-aware scheduling for data centers with heterogeneous servers," *Journal of Scheduling*, no. 12, 2017.
[23] M. W. Convolbo, J. Chou, C. H. Hsu, and Y. C. Chung, "Geodis: towards the optimization of data locality-aware job scheduling in geo-distributed data centers," *Computing*, no. 12, pp. 1–26, 2017.
[24] C. Gu, C. Liu, J. Zhang, H. Huang, and X. Jia, "Green scheduling for cloud data centers using renewable resources," in *INFOCOM*, 2015.
[25] Q. Tang, S. K. S. Gupta, D. Stanzione, and P. Cayton, "Thermal-aware task scheduling to minimize energy usage of blade server based datacenters," in *IEEE Intl. Sym. Dependable Autom. Secure Comput.*, 2015.
[26] L. Cupertino, G. Da Costa, A. Oleksiak, W. Pia, J.-M. Pierson, J. Salom, L. Siso, P. Stolf, H. Sun, and T. Zilio, "Energy-efficient, thermal-aware modeling and simulation of data centers: the coolemall approach and evaluation results," *Ad Hoc Networks*, vol. 25, pp. 535–553, 2015.
[27] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *ICDCS*, 2017.
[28] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *HotNets*, 2016.
[29] D. Yi, X. Zhou, Y. Wen, and R. Tan, "Toward efficient compute-intensive job allocation for green data centers: A deep reinforcement learning approach," in *ICDCS*, 2019.
[30] M. T. Chaudhry, T. C. Ling, A. Manzoor, S. A. Hussain, and J. Kim, "Thermal-aware scheduling in green data centers," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, p. 39, 2015.
[31] J. Chen, R. Tan, Y. Wang, G. Xing, X. Wang, X. Wang, B. Punch, and D. Colbry, "A high-fidelity temperature distribution forecasting system for data centers," in *RTSS*, 2012.
[32] F. A. Gers, J. Schmidhuber, and F. A. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, pp. 2451–2471, 2000.
[33] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.
[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
[35] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *arXiv preprint arXiv:1712.01275*, 2017.

Deliang Yi received the Bachelor degree from the School of Computer Science, Beihang University, China, in 2017. He is currently a master student and research engineer with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His major research interests include deep reinforcement learning, time-series data prediction and energy-efficient data center optimization.

Xin Zhou is now a Research Fellow at the Cloud Computing and Application Platform (CAP) group in Nanyang Technological University. He received the M.E. and Ph.D. from the Department of Information Engineering, Hiroshima University, Japan in 2013 and 2016 respectively. He is expert in thermal-aware ICT subsystem optimization via learning-based approaches, such as DRL and DNN to perform job/task scheduling, network diagnostics and impact analytics. His research interests include reconfigurable architectures, parallel computing, parallel architecture, FPGA computing, deep reinforcement learning and green data center.

Yonggang Wen (S'99-M'08-SM'14) is a Professor with School of Computer Science and Engineering (SCSE) at Nanyang Technological University (NTU), Singapore. He also serves as the Associate Dean (Research) at College of Engineering, and the Director of Nanyang Technopreneurship Centre at NTU. He received his PhD degree in Electrical Engineering and Computer Science (minor in Western Literature) from Massachusetts Institute of Technology (MIT), Cambridge, USA, in 2008. He has worked extensively in learning-based system prototyping and performance optimization for large-scale networked computer systems. Previously he led product development in content delivery network at Cisco, which had a revenue impact of 3 Billion US dollars globally. His work in Multi-Screen Cloud Social TV has been featured by global media (more than 1600 news articles from over 29 countries) and received 2013 ASEAN ICT Awards (Gold Medal). His recent work on Cloud3DView, as the only academia entry, has won 2016 ASEAN ICT Awards (Gold Medal) and 2015 Datacentre Dynamics Awards 2015 – APAC ('Oscar' award of data centre industry). He is the sole winner of 2016 Nanyang Awards in Innovation and Entrepreneurship at NTU. He is a co-recipient of multiple best papers awards, including 2015 IEEE Multimedia Best Paper Award, 2016 IEEE Globecom, 2016 IEEE Infocom MuSIC Workshop, 2015 EAI/ICST Chinacom, 2014 IEEE WCSP, 2013 IEEE Globecom and 2012 IEEE EUC. He received 2016 IEEE ComSoc MMTC Distinguished Leadership Award. He serves on editorial boards for multiple transactions and journals, including IEEE Transactions on Circuits and Systems for Video Technology, IEEE Wireless Communication Magazine, IEEE Communications Survey & Tutorials, IEEE Transactions on Multimedia, IEEE Transactions on Signal and Information Processing over Networks, IEEE Access Journal and Elsevier Ad Hoc Networks, and was elected as the Chair for IEEE ComSoc Multimedia Communication Technical Committee (2014-2016). His research interests include cloud computing, green data center, distributed machine learning, blockchain, big data analytics, multimedia network and mobile computing.

Rui Tan (M'08-SM'18) is an Assistant Professor at School of Computer Science and Engineering, Nanyang Technological University, Singapore. Previously, he was a Research Scientist (2012-2015) and a Senior Research Scientist (2015) at Advanced Digital Sciences Center, a Singapore-based research center of University of Illinois at Urbana-Champaign (UIUC), a Principle Research Affiliate (2012-2015) at Coordinated Science Lab of UIUC, and a postdoctoral Research Associate (2010-2012) at Michigan State University. He received the Ph.D. (2010) degree in computer science from City University of Hong Kong, the B.S. (2004) and M.S. (2007) degrees from Shanghai Jiao Tong University. His research interests include cyberphysical systems, sensor networks, and ubiquitous computing systems. He received the Best Paper Awards from IPSN'17, CPSR-SG'17, Best Paper Runner-Ups from IEEE PerCom'13 and IPSN'14.