GreenDRL: Managing Green Datacenters Using Deep Reinforcement Learning

Kuo Zhang
Rutgers University
New Brunswick, NJ, USA
kz181@rutgers.edu

Peijian Wang
Rutgers University
New Brunswick, NJ, USA
pw277@rutgers.edu

Ning Gu
Rutgers University
New Brunswick, NJ, USA
ng522@rutgers.edu

Thu D. Nguyen
Rutgers University
New Brunswick, NJ, USA
tdnguyen@rutgers.edu

ABSTRACT

Managing datacenters to maximize efficiency and sustainability is a complex and challenging problem. In this work, we explore the use of deep reinforcement learning (RL) to manage "green" datacenters, bringing a robust approach for designing efficient management systems that account for specific workload, datacenter, and environmental characteristics. We design and evaluate GreenDRL, a system that combines a deep RL agent with simple heuristics to manage workload, energy consumption, and cooling in the presence of onsite generation of renewable energy to minimize brown energy consumption and cost. Our design addresses several important challenges, including adaptability, robustness, and effective learning in an environment comprising an enormous state/action space and multiple stochastic processes. Evaluation results (using simulation) show that GreenDRL is able to learn important principles such as delaying deferrable jobs to leverage variable generation of renewable (solar) energy, and avoiding the use of power-intensive cooling settings even at the expense of leaving some renewable energy unused. In an environment where a fraction of the workload is deferrable by up to 12 hours, GreenDRL can reduce grid electricity consumption for days with different solar energy generation and temperature characteristics by 32–54% compared to a FIFO baseline approach. GreenDRL also matches or outperforms a management approach that uses linear programming together with oracular future knowledge to manage workload and server energy consumption, but leaves the management of the cooling system to a separate (and independent) controller. Overall, our work shows that deep RL is a promising technique for building efficient management systems for green datacenters.

CCS CONCEPTS

• **Software and its engineering** → **Scheduling**; **Power management**.

KEYWORDS

Datacenter, green datacenter, deep reinforcement learning, scheduling, power management

ACM Reference Format:
Kuo Zhang, Peijian Wang, Ning Gu, and Thu D. Nguyen. 2022. GreenDRL: Managing Green Datacenters Using Deep Reinforcement Learning. In *Symposium on Cloud Computing (SoCC '22), November 7–11, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3542929.3563501

1 INTRODUCTION

Datacenters (DCs) account for roughly 1% of total worldwide electricity use in 2018 [30]. This large electricity consumption leads to both high financial costs and carbon emission since a significant fraction of electricity is produced by burning fossil fuels; e.g., in 2021, ~60% of electricity in the US was produced using natural gas and coal [3]. Fortunately, many recent advances have increased the sustainability of DCs, slowing the growth of grid electricity consumption despite continued rapid growth in computing demand [30].

One set of advances has been encapsulated in the design and construction of "green" DCs with onsite generation of renewable energy and efficient cooling systems, e.g., [6, 14, 23]. However, while studies have shown that green DCs are effective at reducing "brown" electricity usage, runtime management to maximize benefits is complex. For example, aggressive scheduling of jobs to run at peak generation of solar

SoCC '22, November 7–11, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9414-7/22/11.
https://doi.org/10.1145/3542929.3563501

energy may be beneficial on cooler days, but may trigger expensive cooling states on hot days. A number of works have explored management of green DCs, e.g., [14–16, 25], but most of them depend on predictions of stochastic processes (with different robustness to prediction inaccuracies), make significant simplifying assumptions, and/or only consider independent management of different aspects of a green DC, which can lead to inefficiencies. Further, tuning the management system to optimize performance for specific characteristics of the workload, DC, and environmental conditions can often be an arduous task.

In this paper, we propose, design and evaluate a management system called GreenDRL that uses deep reinforcement learning (RL) to jointly manage several controllable aspects of a green DC. GreenDRL does not rely on predictions of the future, and can be systematically tuned to optimize performance via training. We design and evaluate GreenDRL in the specific context of Parasol [14], a DC with onsite generation of solar energy, a hybrid cooling system that combines power-efficient "free-cooling" and compressive cooling, and a workload where a fraction of the jobs may be deferred to provide some scheduling flexibility. We focus on this particular setup because we have detailed measurements collected from Parasol over an extended time period for training and evaluation. When presenting evaluation results, we discuss evidence showing that GreenDRL was able to learn *principles* that would allow the approach to generalize to wider classes of green and more conventional DCs.

We argue that the use of deep RL brings a robust systematic approach to building efficient management systems optimized for specific workload and datacenter characteristics, avoiding the need for human effort-intensive heuristic design and performance tuning. At the same time, four significant challenges must be addressed in order to apply deep RL to the domain of datacenter management.

First, the deep RL agent must learn an efficient parameterized policy in a large parameter space (assuming the use of policy-based RL). Second, the learning challenge is exacerbated by the need to learn in a highly stochastic environment (e.g., job arrival, outside temperature, and renewable energy generation) with an unbounded horizon. These stochastic processes significantly increase the difficulty of training due to the variance added to the reward signal and the large state/action space to explore. Third, the deep RL agent should be scalable to manage large DCs, and easily adaptable as a DC builds out or changes (e.g., adapting via re-training the deep RL agent without requiring significant changes to its structure). Fourth, the management system must be robust to short-term dynamic changes in the DC, including server outages because of failures or maintenance.

We address the first and fourth challenges by partitioning GreenDRL into two components, a control agent (CA) that uses a neural network and inputs describing the state of the DC to make overall decisions for cooling, workload scheduling, and server power consumption, and a control module (CM) that implements software control of DC systems and a simple heuristic-based algorithm for short-term job dispatching and placement. More specifically, time is divided into a sequence of time slots, and the CA decides what operational state the cooling system should be set to, the maximum number of deferrable jobs that may be dispatched, and the number of servers that should be active at the beginning of each time slot. The CM then actuates the control of the cooling system, set servers' power states (inactive servers are put into a low power state to reduce energy consumption), and dispatch jobs on active servers.

The CA learns an effective management policy by training its neural network using simulation of the DC. Our hybrid design avoids the need for the CA to learn to control all aspects of the DC, e.g., job dispatch and placement, thereby reducing the complexity of the policy that must be learned and the corresponding parameters space defining the learning problem. The CM implements some simple heuristics that are known to work well, and the CA accounts for the workings of those heuristics when seeking to optimize its learned policy during training. Note that the CM can also implement more sophisticated policies such as gang scheduling that are important in some environments but are difficult to encode in a deep RL agent. We leave the exploration of the interaction between the CA and more sophisticated CMs for future work.

The partitioning of GreenDRL also allows the CA to ignore temporary dynamics of the system such as server failures. The CM is responsible for choosing the specific servers that should be active, and can implement appropriate policies for handling server crashes, failures, etc. If system throughput temporarily degrades because of such dynamics, e.g., it takes some time for the CM to detect a server failure and compensate by activating another server, the queues of waiting jobs will build up, signaling to the CA the need to increase the number of active servers until the backlog has been cleared.

Retraining of the CA allows GreenDRL to adapt to long-term changes in the workload and DC (the third challenge). In addition, we use a (continuous) multivariate normal distribution (MND) to describe the CA's probabilistic action policy rather than a discrete probability distribution used in previous related works [4, 27, 28, 35, 46]. Specifically, the CA uses a three-variable MND, with one corresponding to the number of active servers, the second the dispatch limit for deferrable jobs, and the third the cooling setting. This MND is completely describable with 12 parameters corresponding to means and variances of each variable, and the covariances. The CA's neural network structure is thus independent of many characteristics of the DC such as the number of servers

and the exact control of the cooling system, so that it can be retrained without structural changes as the DC evolves.

Finally, we address the second challenge by modifying the state-of-art training algorithm Proximal Policy Optimization (PPO) [39]. Specifically, we adopt variance reduction techniques [28, 29] to handle the high stochasticity in the CA's environment, thus making the training stable.

We evaluate GreenDRL using a simulator that models Parasol and workloads derived from industry traces. Simulation allows us to easily study GreenDRL's performance under a variety of different environmental conditions. We plan to complement this simulation-based study by implementing and experimentally evaluating a prototype of GreenDRL using Parasol in the future. We also study GreenDRL's sensitivity to parameter settings and preliminarily explore GreenDRL's scalability by scaling the simulated DC using simple assumptions.

Our evaluation results show that GreenDRL can learn important principles, including the need to jointly manage the multiple control decisions it is responsible for. For example, GreenDRL successfully learns to delay deferrable jobs to increase green energy and decrease brown energy consumption. GreenDRL also successfully learns to forgo the use of some solar energy if expensive cooling is required on a hot day, and that total server and cooling energy consumption should not exceed available green energy when possible.

Under a variety of environmental conditions, GreenDRL reduces consumption of brown grid electricity by 32–54% compared to a baseline FIFO workload scheduling algorithm coupled with independent reactive cooling control when running a workload deriving from a Google trace [36]. GreenDRL also matches or outperforms a policy (up to 24% reduction) that uses linear programming and oracular future knowledge of the workload and environmental conditions for workload scheduling while relying on an independent optimized reactive cooling control. In these cases, GreenDRL's improved performance arises from its joint management of server and cooling energy consumption. Over an entire year, GreenDRL reduces brown grid electricity consumption by 18% compared to FIFO.

Overall, evaluation results show that deep RL is a promising technique for building efficient management systems for green DCs, and likely for other types of DCs as well.

2 RELATED WORK

Green DCs. Many previous works have explored efficient management of green DCs [14–16, 18, 21, 23, 34, 42]. Managing renewable energy and cooling are two important aspects. Most existing works address them separately. GreenSwitch [14] and GreenHadoop [16] shift the workload and turn off unnecessary machines to match the renewable energy supply. However, they do not manage cooling.

CoolAir [15] manages server power consumption, workload, and cooling to control temperature, temperature variations, and humidity in a DC. CoolAir can reduce cooling cost while tightly controlling temperature and reduce temperature varation. El-Sayed et al. [12] investigate how temperature impacts hardware reliability and server performance and energy consumption. Their results suggest possibilities for saving energy while limiting the negative effects of increasing temperature. Desu et al. [11] propose to dynamically concentrate workload on a minimal set of servers and provide adequate cooling to save energy. They use long-term prediction to decide the active server set, and short-term prediction to manage a pool of standby servers for rapid workload fluctuations. Moore et al.'s work [32] controls the heat generation by temperature-aware workload placement, which is orthogonal to our work. None of them considers onsite generation of renewable energy.

Perhaps the most related work is [25], which also studies the problem of integrating green energy supply and multiple cooling techniques. The authors propose a convex optimization based workload management scheme. Our approach differs in many ways. First, like most model-based methods, their solution relies on accurate prediction of future information. Accurate predictions may be possible in some environments but more challenging in others. Second, it requires access to the system dynamics/models and depends on some special structure or assumptions (e.g. convex) of the models. Further, the models usually need to be simple enough to get the solution. As an early-step work, our solution also relies on some accurate models in the simulator for training our agent. However, we don't need any assumptions about the models.

RL for workload scheduling and DC management. Several related works use a monolithic RL agent to make all control or scheduling decisions [4, 27, 46]. As already mentioned, the partitioning of GreenDRL into two components allows us to adjust the complexity of the CA's learning problem and avoid embedding certain attributes of the DC into the structure of the neural network.

Decima [28] combines a graph neural network to process job and cluster information, and an RL agent to learn scheduling algorithms for jobs with directed acyclic graph (DAG) sub-task structures. It focuses on workload scheduling, especially challenges introduced by continuous arrival of DAG jobs. In contrast, we study simpler jobs but a more complex problem involving many aspects of green DCs. We do use a variance-reducing technique similar to that in Decima.

Lazic et al.'s work uses a model-based RL approach to regulate temperatures and airflow inside large DCs [22]. They model a DC's thermal dynamics as a linear auto-regressive model and use model-predictive control (MPC) to solve the

problem. It is difficult to extend their method to problems that have long scheduling horizons like our. The work [46] trains a deep Q-network [31] to learn a discrete policy for allocating CPU-intensive jobs to servers in a DC. Liu et al. [24] propose a two-level hierarchical framework to handle resource allocation and server power management. DeepEE [35] jointly manages cooling and workload scheduling in a conventional DC, where cooling has a continuous action space and workload scheduling has a discrete action space. To tackle the challenge of hybrid action space, the authors propose a parameterized action space based Deep Q-Network algorithm, which combines the ideas of DQN and actor-critic policy. However, they do not consider renewable energy and free cooling. Further, some aspects of their design may introduce concern for a real DC, e.g., dispatching one task at one time and a neural network with an output for each server.

3 GREENDRL DESIGN

3.1 Problem Statement

Workload. There are two categories of jobs: nondeferrable jobs that should be dispatched as soon as possible and deferrable jobs that may be delayed but should be dispatched within a threshold time period after arrival (e.g., 12 hours). In this work, we only consider CPU-intensive jobs that do not include latency sensitive online request processing. Each job specifies the number of CPU cores that it needs.

Power/Energy. The DC has onsite sources of renewable energy, and can draw power from the electricity grid as needed. In this work, we only consider solar energy. The DC operator pays a constant price for grid electricity, while renewable energy is "free."

Servers. The DC hosts a set of servers, each of which can be turned on (active) or suspended (typically set to a low-power state such as ACPI S3) for power management.

Cooling. The DC's cooling system is used to keep the internal temperature below a threshold (e.g., 30℃). In this work, we consider a hybrid cooling system that includes a low-power "free-cooling" mode and a more power-intensive compressive cooling mode.

Objectives. Objectives for the DC manager include:

- keep the internal temperature below the threshold;
- minimize waiting times for nondeferrable jobs;
- minimize delaying deferrable jobs for longer than the threshold delay time period (called the "dispatch deadline"); and,
- minimize energy cost.

The management system activates/suspends servers, controls the cooling system, and schedules jobs on active servers. While trade-offs between the objectives are required, they are listed above in order of most to least important. For example, it is highly undesirable to reduce energy cost by allowing the inside temperature to rise above the threshold.

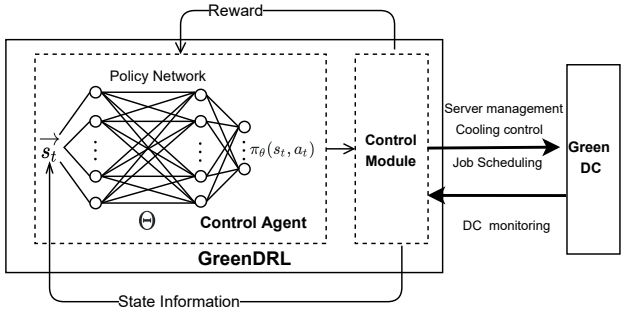


Figure 1: GreenDRL overview

3.2 Design Overview

As already mentioned and shown in Figure 1, GreenDRL comprises two components: a Control Agent (CA) and a Control Module (CM). The CA implements a learned policy that is defined by a parameterized neural network and interacts with its environment in a sequential manner. Time is divided into discrete slots, and at the beginning of each time slot, the CA observes the current state of the DC, and encodes it into an input vector for the neural network. Then the CA chooses a set of actions based on the outputs of the neural network. Once the actions have been carried out, the DC will transition into another state, and the CA receives a scalar reward at the end of the time slot based on the outcomes of the actions. The objective of the CA is to maximize the expected cumulative award over a time window.

We use a multivariate normal distribution (MND) to model the CA's multidimensional probabilistic actions. In particular, when given an input vector, the CA's neural network outputs the parameters of an MND. The CA then chooses a specific set of actions by sampling the MND. GreenDRL uses a three-dimensional MND for controlling the cooling system, determining the number of active servers, and limiting the number of deferrable jobs that may be dispatched.

The CA communicates its decisions to the CM, which then actuates the control actions. Throughout the time slot, the CM heuristically dispatches jobs to active servers as new jobs arrive and resources are freed by completed jobs.

The reward given for each time slot represents a metric that reflects how well the CA is achieving the objectives specified in Section 3.1. We use the reward function, historical data, and a simulator of the DC to train the CA offline.

The CA's neural network is not specific to the experimental setting of our work. However, the number of hidden layers and the number of neurons in each layer may need to be adjusted depending on the complexity of the systems

being managed/controlled; e.g., a DC with many inside temperature inputs, inside and outside humidity readings, and a more complex cooling system may require changes in the neural network to capture an efficient management policy.

The CM's heuristics for job dispatching and placement are general. The sub-component that actuates the control actions are necessarily specific to a DC.

3.3 Control Agent (CA)

The CA contains a parameterized feedforward neural network [37] that approximates a parameterized conditional probability distribution for selecting actions given an input system state.

State. The state of the DC at the beginning of a time slot t is captured in the features of an input vector $\mathbf{s}_t$. GreenDRL currently uses:

- current hour of the day (0 - 23),
- amount of solar energy generated in the previous slot,
- exponential moving average of solar energy generated,
- lengths of the deferrable and nondeferrable job queues,
- wait time of the "oldest" waiting nondeferrable job,
- total number of CPU cores required by waiting nondeferrable jobs,
- earliest deadline of queued deferrable jobs,
- number of deferrable jobs delayed past their deadlines and their total required number of CPU cores.
- number of deferrable jobs that will be delayed past their deadlines if not dispatched in this slot and their total required number of CPU cores,
- inside temperature and outside temperature at the end of each of the previous two slots,
- free cooling fan speed and AC operation status in the previous slot,
- exponential moving average of inside and outside temperature change from slot to slot, and
- number of active CPUs (no. active server × no. CPU cores per server) and how many are currently idle.

As shall be seen below, GreenDRL performs well with these input features. We plan to further explore feature selection and the corresponding impact on GreenDRL's training and performance in the future.

Neural Network. The CA's design is based on policy-based RL methods [39, 41]. In policy-based RL, an agent's action is determined by a parameterized policy $\pi_\theta(\mathbf{s}_t, \mathbf{a}_t) = P_\theta(\mathbf{a}_t|\mathbf{s}_t)$, i.e., the probability of taking action $\mathbf{a}_t$ at state $\mathbf{s}_t$ given the policy parameter θ. The agent chooses a specific action by sampling the policy distribution. The policy $\pi_\theta(\mathbf{s}_t, \mathbf{a}_t)$ can be approximated in many ways, as long as $\pi_\theta(\mathbf{s}_t, \mathbf{a}_t)$ is differentiable and represents a probability distribution. We define the CA's policy as a parameterized MND and use a neural network (often called a policy network) to approximate it.

Our policy network is a fully-connected feedforward neural network with three hidden layers. Each hidden layer contains 24 neuron units. We choose three hidden layers and 24 neurons in each layer by using grid search hyper-parameter optimization. As suggested by [5], we use the same size for all hidden layers. The policy network uses exponential linear units (Elu) as its activation function, which can alleviate the dead neurons problem [8]. The output of the neural network defines an MND for a given input vector $\mathbf{s}_t$, where θ are the weights inside the policy network. The MND has three dimensions that correspond to a cooling sub-action, a server allocation sub-action, and a job dispatching sub-action:

$$\begin{pmatrix} X_{cooling} \\ X_{alloc} \\ X_{cap} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_1^\theta \\ \mu_2^\theta \\ \mu_3^\theta \end{pmatrix}, \begin{pmatrix} \sigma_{11} & 0 & 0 \\ 0 & \sigma_{22} & 0 \\ 0 & 0 & \sigma_{33} \end{pmatrix}\right) \quad (1)$$

We use the neural network to approximate the mean, and treat the variances as hyper-parameters which are independent of the input state. Following [38], we set the covariances of the MND to be zero instead of learnable parameters to reduce the learning parameter space size.

Action Mapping. The CA selects an action $\mathbf{a}_t$ by drawing a sample, $(x_{cooling}, x_{alloc}, x_{cap})$, from the learned MND corresponding to system state $\mathbf{s}_t$. The sample value of each sub-action ($x_{cooling}$, x_{alloc}, x_{cap}) is in the range $(-\infty, \infty)$. It is necessary to map the three values to specific DC control operations. The mapping rules can be problem-specific, with the only requirement being that the rules should be consistent across training and deployment time. In our problem, $x_{cooling}$ will be mapped to a specific cooling scheme, x_{alloc} will be mapped to the number of servers that should be active during a slot, and x_{cap} will be mapped to the maximum number of deferrable jobs to dispatch during a slot. We illustrate the mapping rules for our problem in Section 4.1.

Benefits of Using MNDs. Modeling the CA's action using an MND helps to make the CA adaptable and scalable. It enables the decoupling of the policy network structure and the problem size (e.g., the number of cooling control options and number of servers). The CA can output the policy distribution with at most 12 parameters. If we model the actions with a discrete distribution such as a categorical distribution with k events, the output layer of the CA's policy network would contain at least k units to specify the distribution. Therefore, unlike previous work [4, 27], our CA's policy network structure is independent of the problem size.

Reward Function. The reward function must be carefully designed to reflect the management objectives. In this work, we define the reward function as a negative weighted sum of

a temperature violation penalty, penalties for delaying jobs, and electricity cost:

$$r_t = -\left(\lambda_1 p_t^{temp} + \lambda_2 p_t^{defer} + \lambda_3 p_t^{nondefer} + \lambda_4 c_t\right) \quad (2)$$

The temperature violation penalty p_t^{temp} is a piece-wise linear function that returns 0 for inlet temperatures below the desired threshold, and linearly increasing values as the inlet temperature rises above the threshold. p_t^{defer} is a linear function of the total delayed time of deferrable jobs deferred past their deadlines. $p_t^{nondefer}$ is a linear function of the total waiting time of nondeferrable jobs still queued or were dispatched in time slot t. c_t is the electricity cost during time slot t. In this work, we assume a constant electricity price and so c_t is directly proportional to the energy consumed in the slot.

The weights in the reward function should reflect the relative importance of each management objective. For example, in our evaluation, we set a higher penalty weight for temperature violations because it is undesirable save energy cost while allowing the DC to overheat. We use common hyper-parameter tuning approaches to search for "reasonable" weights in the reward function.

3.4 Training the CA's Neural Network

As previously mentioned, we modify an off-the-shelf policy gradient based RL training algorithm to train the CA. In this subsection, we introduce the basics of policy gradient based training methods and then present the adaptations that we have made to meet the needs of our application. We summarize our training algorithm in Algorithm 1.

Policy Gradient Methods (PMGs). Along with the CA's decision making process under a policy π_θ, a *trajectory*, $\{\mathbf{s}_0, \mathbf{a}_0, r_0, ..., \mathbf{s}_t, \mathbf{a}_t, r_t, ..., \mathbf{s}_T, \mathbf{a}_T, r_T\}$, is generated under the policy. The objective of learning is to maximize the *start value* $V_{s_0}^{\pi_\theta} = E_{\pi_\theta}\left[\sum_{t=0}^{T} r_t\right]$, i.e., the expected cumulative reward from time step 0 to T. Intuitively, the CA seeks to learn a policy (during training), following which it can earn the maximum expected cumulative reward.

PGMs learn the policy parameters θ by using *gradient-ascent* [17] to solve an optimization problem [39, 41] that maximizes the learning objective function $V_{s_0}^{\pi_\theta}$. PGMs estimate the gradients based on generated training data, i.e., m T-length trajectories in each training iteration. Once the agent collects m trajectories, it can estimate the gradient and update policy parameter θ via gradient ascent: $\theta \leftarrow \theta + \alpha \cdot \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T} G_t^{(i)} \nabla_\theta \log \pi_\theta(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)})$ where $G_t^{(i)} = \sum_{t'=t}^{T} r_{t'}$ is the *return* that represents the accumulated reward from time step t for trajectory i, α is the learning rate. The gradient estimate is unbiased, but can experience large variances [40]. One way to reduce the variance is subtracting a *baseline*, b_t, from each $G_t^{(i)}$ [19, 40, 45]. With a baseline, the policy update formula becomes:

$$\theta \leftarrow \theta + \alpha \cdot \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T}(G_t^{(i)} - b_t)\nabla_\theta \log \pi_\theta(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}) \quad (3)$$

The intuition of the policy update is that $G_t^{(i)} - b_t$ estimates how much better (or worse) the return $G_t^{(i)}$ is than some average case b_t under the current policy. Each training update will modify the policy to increase (or decrease) the probability of a trajectory if its return G_t is better (or worse) than the baseline b_t.

Proximal Policy Optimization (PPO). A bad gradient update is damaging in RL training. To make each update "safe," PPO has been developed based on PGMs [39]. Intuitively, PPO updates the policy while satisfying a constraint on how far apart the new and old policies are allowed to be. We adopt PPO in our training algorithm (line 15 in Algorithm 1).

Baseline selection. It is worth noting that $G_t^{(i)}$ is *time dependent*. So we also use a *time dependent* baseline, which is the cumulative reward from slot t to the end, then average over the m trajectories generated for a particular training iteration (line 10 in Algorithm 1). This baseline has been shown effective for RL training in an environment with external stochastic processes [28, 29].

Additional details. We set the variances of the MND policy to relatively large values at the beginning of the training and gradually decay them (line 16 in Algorithm 1). The training algorithm samples the start time of the trajectories from the

Algorithm 1 GreenDRL training algorithm

1: Initialize policy parameter θ, variance $\vec{\sigma} = [\sigma_{11}, \sigma_{22}, \sigma_{33}]$, and parameters ϵ_1, ϵ_2, τ_{mean}
2: **for** iteration = 1, 2, ... **do**
3: Sample training trajectory length $T \sim Exponential(\tau_{mean})$
4: Clip length T with a lower-bound and upper-bound
5: Sample T_0 and get a trace H starting from T_0 to $T_0 + T$
6: **for** $i = 1, .., m$ **do**
7: Execute the current policy on trace H to collect a trajectory of length T starting from T_0: $\mathcal{T}_i = \{\mathbf{s}_0^{(i)}, \mathbf{a}_0^{(i)}, r_0^{(i)}, ..., \mathbf{s}_T^{(i)}, \mathbf{a}_T^{(i)}, r_T^{(i)}\}$
8: **end for**
9: **for** $t = 0, ..., T$ **do**
10: Compute the baseline: $b_t = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=t}^{T} r_k^{(i)}$
11: **end for**
12: **for all** time slot $t = 0, ..., T$ in each trajectory $\mathcal{T}_i$ **do**
13: Compute return: $G_t^{(i)} = \sum_{t'=t}^{T} r_{t'}^{(i)}$
14: **end for**
15: Update policy parameters: $\theta_{k+1} \leftarrow PPOupdate(G_t^{(i)}, b_t, \theta_k; T, m)$
16: Decay the variance of the MND policy: $\vec{\sigma} \leftarrow \vec{\sigma} \cdot \epsilon_1$
17: Increase the mean of trajectory length: $\tau_{mean} \leftarrow \tau_{mean} \cdot \epsilon_2$
18: **end for**

training trace randomly to help the CA explore different DC environment patterns. Similar to the approach in work [28], our training algorithm samples the trajectory length from an exponential distribution whose mean grows gradually. By increasing the trajectory length over time, we encourage the agent to learn a basic policy quickly at the beginning of the training and then improve the policy gradually. We clip the trajectory length by two thresholds (line 4 in Algorithm 1) because we want the training trajectory to be reasonably long for the agent to observe the long-term effect of its actions but not so long that training becomes overly expensive.

3.5 Control Module (CM)

The CM actuates control actions for server power management and cooling control according to the CA's instructions. It is also responsible for scheduling jobs on servers, and handling server failures.

Server power management. At the beginning of each time slot, the CM receives the number of CPU cores that should be active from the CA. The CM will then: (1) wake up servers if the total number of CPU cores in the currently active servers is less than the CA's decision, or (2) put some servers into a low-power sleep state (e.g., ACPI S3) if the number of active CPU cores exceeds the CA's decision.

An active server selected for suspension will be marked as *unschedulable*, and the CM will stop dispatching jobs to it. The CM puts such an unschedulable server into a sleep state immediately if all leftover running jobs on this server finish before the next slot, otherwise the CM marks the server as active again at the beginning of the next slot. When multiple candidates are available, the CM prefers to select an active server with fewer running jobs to suspend. A sleeping server takes some time to wake up and re-join the active cluster.

Cooling control. The CM controls the cooling system according to the CA's cooling sub-action. Control actions include setting the free cooling fan speed, activating the AC (compressive cooling), or turning off both cooling units.

Job dispatch and placement. The CM dispatches jobs to active servers using simple, well-known heuristics. It monitors all scheduling events such as job arrival, job completion, and servers entering the active state. It maintains a queue of nondeferrable jobs in order of arrival, and a queue of deferrable jobs in order of dispatch deadlines. It attempts to dispatch jobs to available resources at the occurrence of any scheduling event.

The CM respects the job dispatch sub-action and uses the following heuristics: (1) always try to dispatch nondeferrable jobs before dispatching any deferrable jobs, (2) dispatch jobs according to the ordering of the queues, (3) place each job on the server with the fewest unallocated CPU cores yet can meet the job's requested number of cores, and (4) skip a job if it cannot fit on any active server (because of fragmentation) to avoid head-of-queue blocking. However, if a job has been skipped over more than a threshold number of times, then the CM will stop dispatching jobs until the blocked job can be dispatched. This strategy has been shown effective to alleviate the head-of-queue blocking problem [10].

Server failures. The CM monitors server failures. When an active server crashes, the CM tags the jobs running on that server as failed and attempts to restart the server. If the server cannot be restarted, or has failed more than a threshold number of times within a time period, then the CM removes the server from service. If there are suspended servers, the CM wakes one up as a replacement.

Temperature emergencies. The CM also monitors the DC inlet air temperature and will set the cooling system to the maximum setting if the temperature rises above a redline threshold. If the temperature emergency persists, then it will turn all servers off.

4 EVALUATION METHODOLOGY

As mentioned above, we evaluate GreenDRL using simulation. In this section, we first describe Parasol [14] and the models that we have constructed for our simulator using data collected from the DC. We then describe two alternative management approaches that we will compare to GreenDRL, and our evaluation methodology.

4.1 Parasol and Simulation Models

Parasol. Parasol comprises a 7'×12' container, a set of 16 polycrystalline solar panels, a grid-tie, a Dantherm Flexibox 450 air-side economizer ("free-cooling"), and a Dantherm iA/C 19000 direct-expansion air conditioner (AC). Parasol can host two 42U racks of IT equipment. It draws additional power from the electricity grid when there is insufficient solar energy.

Parasol's cooling is controlled by a TKS 3000 controller. Free cooling is used whenever the outside temperature is lower than a programmable threshold, with the TKS controller modulating the fan speed according to a temperature sensor inside the container and the threshold. The minimum fan speed is 15%. The free cooling unit consumes between 8W and 425W depending on fan speed. When the outside temperature exceeds the threshold, the controller switches from free cooling (turning the unit off) to the AC. The AC controls its own cooling cycles according to an internal temperature sensor and threshold settings, consuming about 2.3kW when the compressor is on. Parasol is partitioned into a hot and cold aisle for cooling efficiency.

Parasol hosts several server configurations. We are only simulating one: Supermicro SYS-5019S-M2 with Intel Xeon E-1275v6 (4C/8T), 64G DDR4 RAM, and a 960 GB SSD.

We use 7 months of data collected from Parasol to build and parameterize the models described below. We validated our simulator by comparing detailed simulation results, e.g., inside temperatures vs. time, against recorded historical data for days with different environmental and operational characteristics.

Server power model. We use a linear power model to estimate server power consumption [13, 20]:

$$P_{server}(u) = P_{idle} + (P_{peak} - P_{idle})u \tag{4}$$

where u is the CPU utilization of an active server, and P_{idle} and P_{peak} represent the idle ($u = 0\%$) and peak power ($u = 100\%$) draws, respectively. It has been shown that the above model is reasonably accurate, especially when estimating the total power consumption of a server cluster [13].

Cooling thermal and power models. Simulation time is divided into discrete time slots. We developed models that calculate: (1) the inlet air temperature at the beginning of the next time slot, and (2) the average power consumption during a time slot. For free cooling, the models are:

$$T^{in}_{t+1} = (1 - e^{c_1 F_t}) * (T^{out}_t - T^{in}_t) + c_2 P^{it}_t + T^{in}_t \tag{5}$$

$$P^{fc}_t = \alpha F^2_t + \beta F_t + \gamma, \tag{6}$$

where t is the time slot, T^{in}_t is the inlet air temperature (at time t), T^{out}_t is the outside temperature, F_t is the fan speed, P^{it}_t is the average IT power, P^{fc}_t is the average power draw of the free cooling unit, and c_1, c_2, α, β, and γ are constants. $P^{fc}_t = 0$ when free-cooling is turned off during a time slot.

The thermal model for the AC unit is:

$$T^{in}_{t+1} = d_1 T^{in}_t + d_2 T^{out}_t + d_3 P^{it}_t + d_4, \tag{7}$$

The AC unit has its own internal cycle, with varying power consumption. However, its average power consumption is stable. Thus, we model the power consumption of the AC as a constant when the unit is on, and 0 when it is off [15].

4.2 Baseline Management Policies

We compare GreenDRL's performance to two baseline policies. As is typical of state-of-the-art management systems, each of the baseline policies manages the number of active servers and schedules the workload while leaving control of the cooling system to a different controller. Neither policies are derived from existing RL-based work (e.g., [28, 35, 46]) because they do not handle important aspects of our problem such as leveraging onsite renewable energy, controlling the free cooling system, and workload constraints. Adapting the approaches would have required deep structural changes.

FIFO. FIFO does not work based on time slots. At any time, FIFO keeps as many active servers as needed to dispatch all jobs, and suspends servers that have been idle for a threshold period (15 seconds in our experiments, which is equal to the server wake up duration). FIFO queues jobs only when there is not enough available resource even if all servers are turned on. It places jobs on servers using the same heuristic as described in GreenDRL's CM (Section 3.5). FIFO relies on a simulated TKS controller for its cooling control.

LP. LP adapts GreenSwitch [14], a state-of-the-art green DC management framework, to our environment. GreenSwitch formulates a Mixed Integer Linear Programming (MILP) problem to minimize the overall electricity cost, subject to some workload and battery operational constraints, by managing workloads and energy sources in green DCs.

LP adapts GreenSwitch's MILP formulation to account for different assumptions, the most important of which is our assumption that each deferrable job has its own deadline based on its arrival time. LP is similar to GreenSwitch in that it solves for the the number of active servers required to meet the power demand of jobs while ignoring possible fragmentation due to placement.

The actual dispatching and placement of jobs on active servers are done by a heuristic-based component. In our experiments, we use GreenDRL's CM. GreenSwitch uses simple predictions of the future, and resolves the optimization formulation at the beginning of every time slot to account for inaccuracies in its predictions. LP's formulation is more expensive to solve, so we make the simplifying assumption that LP has oracular knowledge of future job arrivals, solar energy generation, and outside temperatures. LP's MILP formulation is presented in Appendix A.

Overall, LP works as follows. At the beginning of an experiment, it solves the MILP formulation for the number of servers that should be active at the beginning of each time slot for the entire experiment time horizon. Then, the CM handles server activation and suspension according to the solution, and dispatches and places jobs on active servers.

Because we solve LP's formulation only once at the beginning of each experiment, the queue of deferrable jobs may build up due to fragmentation: servers cannot be fully utilized as assumed in LP's formulation. We use a simple heuristic to address this issue. Specifically, at the beginning of a time slot, the CM will allocate additional servers (up to the maximum number of servers) in order to dispatch all jobs that have reached their dispatch deadline.

LP (like GreenSwitch) does not control cooling. LP is an MILP optimization problem that cannot be solved in a reasonable amount of time if control of the cooling system is

added. Rather, it depends on a cooling optimizer used in CoolAir [15] to keep the inlet temperature within the appropriate operational range. Specifically, the cooling optimizer enumerates all possible cooling control decisions that will not incur temperature violation in the next time slot and selects the one with the least cooling energy cost. Note that the cooling optimizer needs the information of average IT power consumption over the coming slot. This information can be derived from the MILP solution.

LP is not a realistic online policy. Rather, we have designed LP to be an optimistic (although not optimal) policy that is generally representative of a class of policies such as [14] and [11] to help gauge GreenDRL's performance.

4.3 GreenDRL Prototype

We implement the CA's policy network using Tensorflow 2.0 [1]. The implementation is used as a module in our current simulator; we expect that it will be reused in entirety in a real implementation (in Parasol).

As previously mentioned, at the beginning of each time slot, the CA outputs a sample $(x_{cooling}, x_{alloc}, x_{cap})$ drawn from the MND. S_{active}, the number of active server, is then set to:

$$S_{active} = \left\lceil \frac{\max(1, \min(x_{alloc}, MS_{max}))}{M} \right\rceil \quad (8)$$

where S_{max} is the number of servers in the DC, and M is the number of CPU cores in each server.

For cooling, the CM selects one of the following three schemes depending on the value of $x_{cooling}$: (1) both cooling units turned off (recirculation) if $x_{cooling} < 15$, (2) free-cooling on at fan speed $round(x_{cooling})$ and AC off if $15 \leq x_{cooling} \leq 100$, (3) free-cooling off and AC on if $x_{cooling} > 100$.

The maximum number of deferrable jobs to dispatch in a slot is $C_{defer} = max(0, round(x_{cap}))$

We assume that GreenDRL can control the cooling system directly (rather than depending on the TKS) and is able to turn the two units on/off and set the fan speed for the free-cooling unit.

When training the CA's neural network, the initial mean of the trajectory length (exponential distribution) is 24 hours, and the maximum and minimum length of a trajectory is 96 hours and 48 hours, respectively. The CA collects 16 trajectories for computing each gradient update. The initial variances of the MND are 10^2 (cooling dimension) , 15^2 (resource allocation dimension), and 20^2 (dispatch cap dimension). The decay rate for the variances is 0.9997^2. The final variances are set to 1 for inference. The rate for increasing the mean of trajectory lengths is 1.0001. We use the RMSProp optimizer and its default hyper-parameters in TensorFlow for training. The hyper-parameter ϵ used in PPO-Clip is 0.2. The reward discount factor is 1.

4.4 Workloads and Environmental Traces

Workloads. We assume compute-intensive jobs, so that each CPU core assigned to a job will be fully (100%) utilized. Examples of such jobs include CPU-bound batch processing jobs [9] and high-performance computing (HPC) jobs [26]. Our experiment uses two workload traces sampled from production clusters.

Google trace [36] includes information about job submission and execution on a cluster of about 12.5k machines, collected over 29 days. Each job is submitted with a few parameters such as scheduling class, resource request, and constraints. The types of jobs in this trace is diverse, including service jobs, batch processing jobs and HPC jobs.

Similar to what has been done in [7], we eliminated all long-lasting production service jobs because request scheduling is more appropriate for such applications rather than job scheduling. We mark a job as deferrable with a certain probability so that the ratio of deferrable workload is around 75% in the sampled trace. We scale jobs' resource requirements so that each job can be run on a single simulated server. Each deferrable job's dispatching deadline is set to 12 hours after arrival. We choose these values partly by using information from [2, 43], with the 75% deferrable jobs being somewhat higher than reported because we are only focusing on non-service jobs. We explore the impact of different settings in a sensitivity study discussed below.

Alibaba trace [44] was collected from an Alibaba production cluster with 6472 GPUs (on about 1800 machines) over a two-month period. The workloads are a mix of training and inference jobs, which are computing-intensive on GPUs. Each job contains a certain number of instances and gang-scheduling requirement is prevalent. We use the job's total GPU requests as its resource requirement in our simulation. In the sampled trace, we scale the job resource request so that any job can be executed in our simulated server. Before sampling, we excluded jobs whose resource requests are extremely large to avoid an overly skewed distribution of resource request (i.e., the majority of the jobs have the minimum resource request, 1 CPU). The number of jobs discarded is less than 1%. The resulting trace contains about 74% deferrable jobs.

Figure 2 shows the characteristics of the sampled workloads. For both traces, the average load is about 50% of the total simulated cluster capacity. We use the Google and Alibaba workloads for overall comparison with FIFO and LP, and use the Google workload with varying individual workload characteristics to study GreenDRL's sensitivity.

Environmental traces. We use historical data collected from Parasol's environment for the calendar year 2013. The trace contains solar energy generation and outside temperature for each time slot (five minutes). Training and evaluation mostly

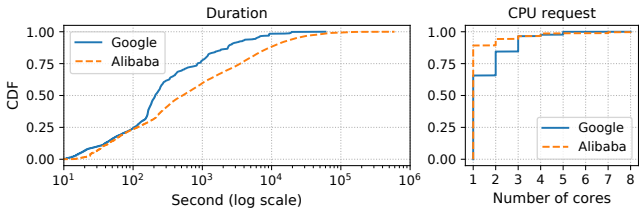


Figure 2: CDFs of job duration and CPU request from Google and Alibaba traces.

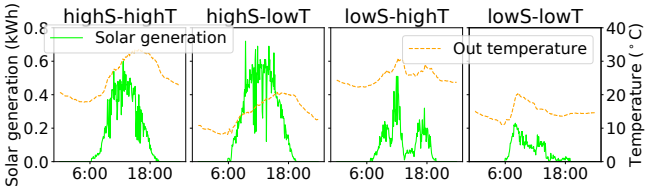


Figure 3: Outside temperatures and solar energy generation for four days with different characteristics.

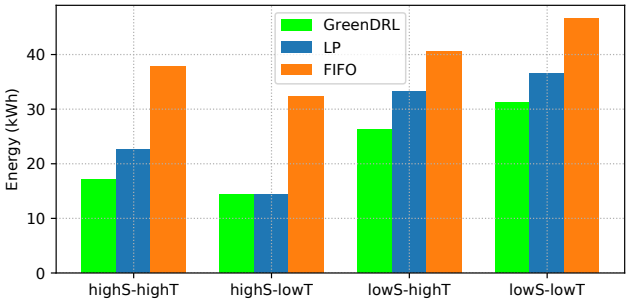


Figure 4: Brown energy consumption when running the Google workload.

use data from May and June, with outside temperatures in the range of 5°C to 35°C and free cooling and AC being used at different times to maintain the DC inlet temperature.

DC. Our simulated DC hosts 32 servers, each with 8 CPU cores. Time slot duration is five minutes. The server wake-up time is 15 seconds, and the average wake-up power is the same as the active idle power, 20W. The peak power is 130W. The AC draws 2.3kW on average when it is on and the peak power demand of the free-cooling unit is 425W. The treshold inlet temperature is 30°C.

Evaluation Experiments. We select four days with different environmental characteristics to explore and evaluate GreenDRL's performance and exclude them from the training data (Figure 3). We use each of the day as the third day in a 4-day long trace. The first two days comprise a "warm-up" period for deferrable jobs to be accumulated. Starting from the third day, the results are typically stable and similar to subsequent days. Thus, we compare performance for the third day for GreenDRL and the two baseline policies. Finally, we add a fourth day so that LP cannot arbitrarily delay deferrable jobs in the last day out of the scheduling window to save cost.

The four selected days lead to four different 4-day long traces, with one including a third day having relatively high solar energy generation and high outside temperature (denoted as highS-highT), and three more with the third day being highS-lowT, lowS-highT, lowS-lowT, respectively.

For evaluation workloads, we randomly selected a 4-day long trace from each of the Google and Alibaba traces. The Google and Alibaba evaluation traces start from day 2 and and day 5, respectively.

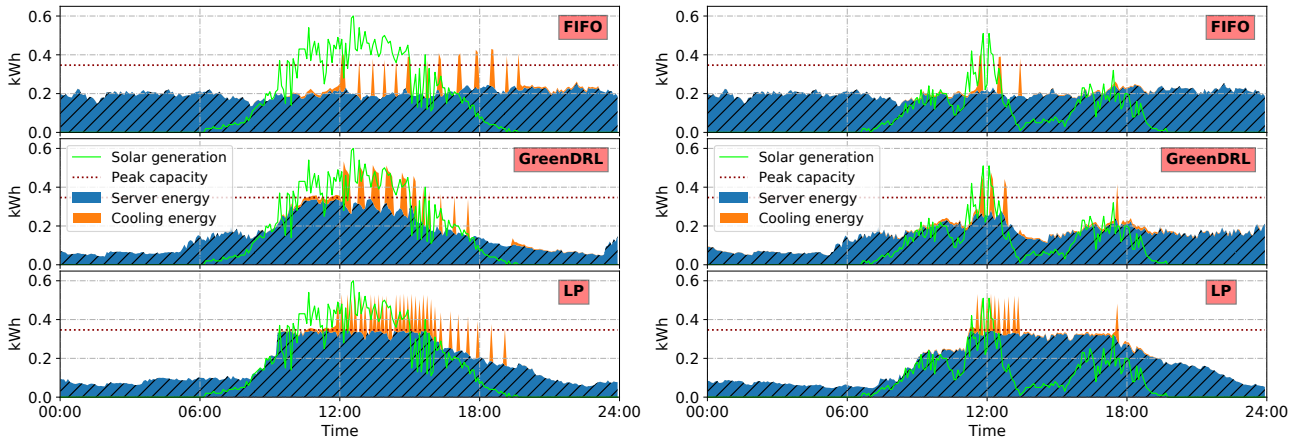
We also evaluate GreenDRL's performance over an entire year. For this evaluation, we stitch together the Google trace to get 365 days. We use environmental data collected from Parasol for the calendar year 2013.

Training the CA. We use Parasol's environmental data from May and June 2013, excluding the four days selected for evaluation, and the entire Google and Alibaba traces for training. When training, we always train two GreenDRL instances with the same hyper-parameters but different random seeds for each experiment setting, and deem the training results valid if the final rewards of the two differ by ≤2% when evaluating on a validation dataset. Based on our overall experience of conducting trainings, approximately 95% of the training results are valid. We checkpoint learned model parameters periodically while the training converges (when reward stops increasing or fluctuates within a small range for a threshold number of iterations). We select the model with the maximum validation reward from all checkpoints as the final training result.

5 EVALUATION RESULTS

CA training and inference. All results reported before the our sensitivity studies and for the year-long evaluation are from experiments using a CA that was trained just once for the Google workload and once for the Alibaba workload. We do retrain the CA when changing important characteristics of the workload or components of the reward function in our sensitivity studies.

Training the CA takes ~5-8k iterations to converge, corresponding to ~48-72 execution hours on a server with a Xeon Silver 4110 CPU (8 cores / 16 threads) and 64GB ECC DDR4. We do not use GPUs for training because most of the time is spent on DC simulation instead of the training itself. On the same machine, the CA takes ~100ms to make a decision. The low inference overhead makes it possible to use the CA for shorter time slots than our current setting of 5 minutes.



(a) highS-highT **(b) lowS-highT**

Figure 5: Solar energy generation, server energy, cooling energy and cluster-wise server peak energy per slot with Google workload (cooling energy is added on top of server energy).

Brown energy consumption. Figure 4 plots brown energy consumption when the Google workload runs on days with different amounts of solar energy and outside temperatures. As expected, these results show that both GreenDRL and LP significantly reduce brown energy consumption compared to FIFO, with GreenDRL reducing consumption by 32-54%. GreenDRL also matches LP's performance on the highS-lowT day and reduces brown energy consumption by 14-24% on the three other days.

Figure 5 plots energy consumed by the servers and cooling system together with solar energy generation against time for the highS-highT and lowS-highT days. We observe that GreenDRL and LP outperform FIFO by delaying deferrable jobs to run in time slots with ample solar energy. Less obviously, the total energy consumed by the activated servers are lower under GreenDRL and LP compared to FIFO because (a) servers are turned on/off less often, and (b) jobs are deferred to ensure high utilization for active servers even in the absence of solar energy, reducing base energy consumption (the energy consumed by a server even when it is idle).

More interestingly, on both days, GreenDRL activates fewer servers compared to LP during the hottest part of the day. GreenDRL's decisions leave some solar energy to power the AC and reduces heat generation leading to reduced need for cooling. This leads GreenDRL to consume only 1.97kWh of brown energy for cooling compared to 3.96kWh under LP for the highS-highT day. When there is low production of solar energy, GreenDRL also defers job execution to later in the day, when lower temperatures require less cooling. These results demonstrate that GreenDRL is able to learn the importance of jointly managing server activation and cooling, and effectively leverages this joint management to outperform LP. The gap between brown energy consumption under the two policies would be larger except for the fact that there is more solar energy than can be consumed by all the servers in the datacenter during periods of high production, thus automatically leaving some solar energy for cooling under LP.

While we do not show the detailed energy consumption for the highS-lowT and lowS-lowT days, it is intuitive that GreenDRL and LP perform similarly for these two days. Both policies are able to defer jobs to consume solar energy without needing to run the AC during the highS-lowT day. GreenDRL's joint management of cooling and server activation does lead to minimal benefits on the lowS-lowT day.

Results for the Alibaba workload are similar, although the differences in performance are smaller: GreenDRL saves 22-37% compared to FIFO and 5-15% compared to LP on the highS-highT, lowS-highT, and lowS-lowT days. LP does slightly outperform GreenDRL on the highS-lowT day; as discussed above, we expect the policies to have comparable performance for the highS-lowT day, and in this case, the difference is very small. For the remainder of the discussion, we focus on the Google workload while noting that results for the Alibaba workload are similar.

For the Google workload, GreenDRL reduces brown electricity consumption by 18% compared to FIFO when the policies were run over an entire year. (It was not computationally feasible to run LP for a full year.) Note that GreenDRL achieves this performance improvement even though the CA is trained only with summer environmental data.

Cooling control and inside temperature. FIFO (with TKS management of cooling) always maintain the inside temperature at or below 30°C while both GreenDRL and LP (with

greedy optimized cooling control) allowed the inside temperature to exceed 30°C by small amounts during a small number of time slots. For example, in highS-highT day under GreenDRL, the inside temperature rises above 30°C at the end of 6 slots (288 slots in 24 hours), reaching a maximum temperature of 30.6°C. Under LP, the inside temperature rises above 30°C at the end of 14 slots, reaching a maximum temperature of 30.1°C.

It is not surprising that GreenDRL allows some violation of the 30°C threshold. The CA is penalized when the inside temperature rises to above the threshold, but it may be worthwhile to trade off the penalty for reduced brown energy consumption. Further, the probabilistic nature of the CA's decision making may also lead to some violations. Nevertheless, the steep rise in penalty with increasing temperatures discourages GreenDRL from allowing serious violations. We can decrease the number of violations and the maximum temperature reached by increasing the penalty. However, reaching less than 31°C 2% of the time seems acceptable.

Figure 6 plots the operation of the cooling system, outside temperature, and inside temperature against time for 24 hours. These results confirm that GreenDRL learns to operate the cooling system to maintain safe inside temperatures while minimizing brown energy consumption by using lower power configurations when possible. Compared to LP, GreenDRL often cools the DC more than needed (to several degrees lower than 30°C) both while using free-cooling and AC cooling. Overall, GreenDRL's aggressiveness leads to only a small amount of added energy consumption. Nevertheless, it does point to an area of potential improvement for GreenDRL in future work.

Figure 7 plots the CA's decisions (cooling, cap on dispatching of deferrable jobs, and the number of active servers) in the time period with solar energy in the highS-highT day. We observe that the CA tends to reduce the number of active server whenever it turns on the AC so that the combined energy consumption does not exceed the available solar energy (by too much). This reemphasizes GreenDRL's ability to learn to jointly manage cooling and server energy consumption to avoid/reduce the use of brown energy.

Workload scheduling. Using the highS-highT day as a representative example, the sampled Google workload never exceeds the capacity of the DC. Thus, 98% of the nondeferrable jobs are dispatched immediately on arrival by FIFO, while the remaining jobs may be delayed up to 15 seconds for a server to be activated. Under GreenDRL, 1% of the nondeferrable jobs are delayed by 18 seconds or more. LP performs much worse than both GreenDRL and FIFO, delaying 21% by more than 15 seconds, and 1% by 350 seconds or more. These delays under LP result from LP's ignorance of fragmentation in job placement when deciding on the number of active

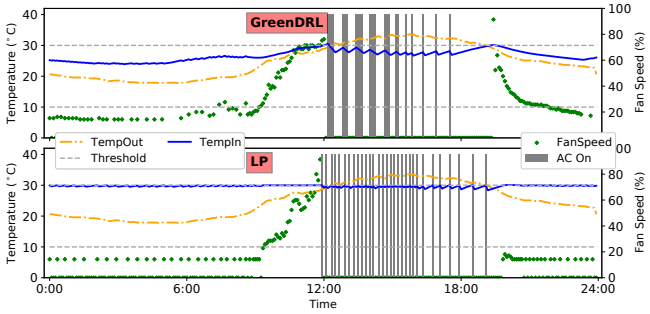


Figure 6: Cooling control comparison in the highS-highT day.

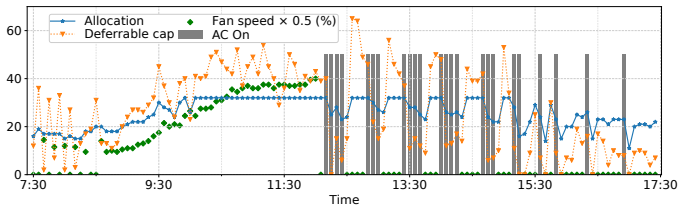


Figure 7: GreenDRL's decision making in the highS-highT day from 7:30 to 17:30

servers. Further, under LP, the CM will greedily dispatch as many deferrable jobs as it can at the beginning of each time slot. Nondeferrable jobs arriving after the start of a time slot will thus often have to wait for dispatched deferrable jobs to complete or more servers to be activated the beginning of the next time slot. GreenDRL reduces this wait time by capping the number of deferrable jobs that can be dispatched in a time slot, leaving spare capacity to handle arriving nondeferrable jobs. Note that this trades off additional energy consumption to reduce wait times for nondeferrable jobs.

GreenDRL leads to some violation of the dispatch deadline for deferrable jobs. Specifically, under GreenDRL, 1% of the deferrable jobs are dispatched up to 288 seconds after their deadlines. Under LP, all deferrable jobs are dispatched before or at the deadlines. Deadline violations are possible under GreenDRL for several reasons, including the fact that violations lead to penalties but are not absolute constraints and GreenDRL's probabilistic nature. We can reduce the number and lengths of violations by increasing the corresponding penalty, as shown in the sensitivity evaluation. However, the frequency and length of violations are both already quite small for our current weighting of penalties.

Sensitivity. We have studied GreenDRL's sensitivity to a number of important parameters, including the length of the dispatch deadline for deferrable jobs (6, 12, and 24 hours), the ratio between deferrable and nondeferrable jobs (50% and 75% deferrable), intensity of workload (50% and 70% average

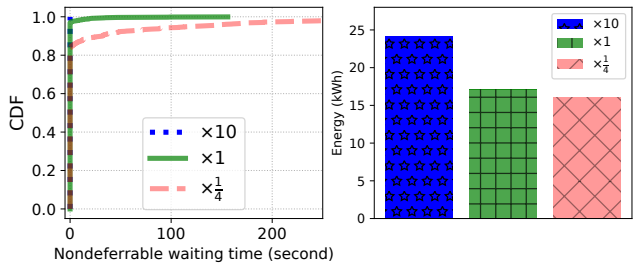


Figure 8: GreenDRL with different weights on the penalty for delaying nondeferrable jobs when the Google workload is run in a highS-highT day. The left graph plots the CDF of wait times for nondeferrable jobs. The right graph plots brown energy consumption.

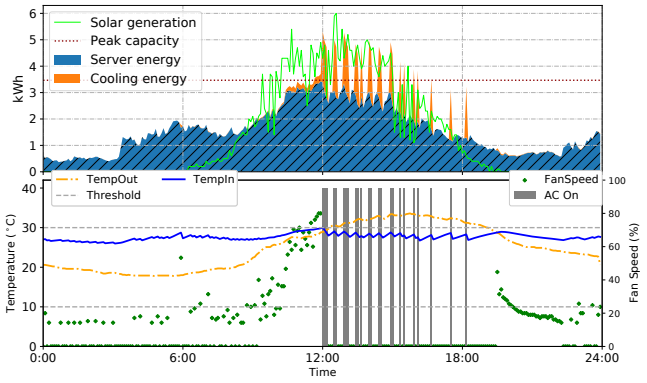


Figure 9: GreenDRL energy consumption (top) and cooling control (bottom) in a scaled DC when running the Google workload on the highS-highT day.

utilization), and different weightings for the components of the reward function. Results match intuition; for example, when only 50% of the jobs are deferrable, GreenDRL's savings for brown energy consumption compared to FIFO and LP are smaller, 27-44% and 1.2-12.5%, respectively. Similarly, higher average utilization reduces the flexibility that GreenDRL has to delay deferrable jobs to consume green energy and avoid expensive cooling. Thus, GreenDRL's savings in brown energy consumption compared to FIFO are smaller.

Reducing the dispatch deadline for deferrable jobs to 6 hours also reduces GreenDRL's ability to delay deferrable jobs to increase green energy consumption, leading to 30% saving compared to FIFO on the highS-highT day. Stretching the deadline to 24 hours does not bring much benefits given that 12 hours already allowed GreenDRL to fully utilize all servers when green energy is available. Finally, GreenDRL is not sensitive to a mix of different dispatch deadlines. When run on a workload where 25%, 50%, and 25% of deferrable jobs have 6, 12, and 24 hours dispatch deadlines, respectively, GreenDRL did not incur additional deadline violations while achieving energy savings of 37-53% compared to FIFO.

Perhaps the most interesting results are for sensitivity to penalty weightings. Ideally, GreenDRL should achieve different trade-offs among its objectives when weights in the reward function are changed. Figure 8 shows the results when we change the weight of the penalty for delaying nondeferrable jobs. Increasing the weight by 10× can completely avoid nondeferrable waiting but incurs increased energy consumption. Interestingly, GreenDRL learns to reserve more free resources (by allocating more servers and reducing the cap for dispatching deferrable jobs). In contrast, decreasing the weight by (1/4)× increases the energy saving but also increases the wait time for many nondeferrable jobs.

Scalability. We simulate a larger DC to provide evidence for GreenDRL's scalability. We increase the server number by 10 (320 servers) and keep the single server power model unchanged. We revise our cooling model which has similar cooling effects for the cluster but with 10× power consumption. We realize that these scalings are rough approximations at best. But it allows us to study whether GreenDRL can still learn the fundamental principles and control policies with the *same* policy neural network structure and training algorithm; i.e., GreenDRL's neural network structure and size are not dependent on DC size, at least within the order of magnitude change that we are studying.

The training takes 3 times longer because of the larger simulation. The inference time remains the same because it only depends on GreenDRL's policy neural network structure. As show in Figure 9, the GreenDRL can learn the similar behavior as before, e.g., shifting workload execution to solar-rich time, jointly managing server and cooling power. The brown energy consumption increases proportionally (by a factor of 10.4 in the highS-highT day). These results show the robustness of GreenDRL's design to DC size.

Summary. LP is not an optimal policy given various simplifying assumptions. However, it is quite powerful since it is given oracular knowledge of the future. Thus, it is encouraging that GreenDRL can match LP's performance for minimizing brown energy consumption on days when AC cooling is not needed, while also keeping wait times for nondeferrable jobs and violations of deferrable dispatch deadlines low. It is further encouraging that GreenDRL can learn to jointly manage cooling and server energy consumption to outperform LP on days when AC cooling is needed. While it is certainly possible to tweak GreenDRL in various ways to further improve performance, we believe that the results above demonstrate that deep RL can be effectively applied to the complex multi-dimensional problem of managing energy consumption in a green datacenter.

6 DISCUSSIONS

Offline simulation and training. GreenDRL relies on simulation to train its agent. Building accurate models for simulation to train an RL agent can be especially challenging. Exploration of the learning space often leads to simulated execution under unusual operating conditions. For example, in our cooling problem, a good policy is usually just keeping the inside temperature right below the maximum threshold. So traditional management systems, like optimization or heuristic, just need the cooling model to work well in a narrow inside temperature range.

In contrast, training an RL agent may require exploring states with very low or very high inside temperatures for the agent to learn the extreme penalties associated with allowing the state. Unfortunately, it is very difficult to gather data at these operating points to build the models, and it is difficult to extrapolate from data obtained under "normal" conditions; for cooling, thermal dynamics might be quite different in different temperature ranges.

Thus, it would be desirable to eliminate the need for simulation and models, and directly train the agent using the real system. But the challenges of system safety and long training time need to be addressed before real deployment. It is also possible to explore a combination of offline and online training [33], where offline training can learn a relative good/safe policy from historical data without the need for simulation and models and online training will improve the agent's performance on the real system.

Stability. Although the overall performance is good, GreenDRL will sometimes vary its decisions rapidly within a time period. For example, it may rapidly alternate between turning servers and the cooling system on/off from one time slot to the next. This behavior is acceptable with the current reward function because the rewards/costs for these state changes are low. However, it may be disadvantageous from the perspective of system stability. Thus, it would be worthwhile to investigate a systematic approach to designing reward functions that would encourage "smooth" operation.

7 CONCLUSION

In this work, we study the use of deep RL to jointly manage all controllable aspects (e.g., workload, power, cooling) of a green DC. Specifically, we propose and evaluate GreenDRL, a management system that combines deep RL and simple heuristics. Our design addresses several important challenges of the problem domain, brining a robust approach for designing efficient management systems that can account for specific workload, datacenter, and environmental characteristics. Simulation results using historical data collected from Parasol, an experimental green DC, show that GreenDRL successfully learns many important management principles and outperforms two baselines policies. We thus conclude that deep RL is a promising technique for building efficient DC management systems.

ACKNOWLEDGEMENTS

This work was partially supported by NSF grant #1730043. We thank the anonymous reviewers and our shepherd, Noman Bashir, for helping to improve the paper.

A LP FORMULATION

Decision Variables:

$w_t^d \in [0, N_{max} \times C]$: amount (CPU-seconds) of deferrable workload, scheduled in slot t. N_{max} is the maximum number of servers, C is the capacity (CPU-seconds) of each server.

$n_t \in \{1, \ldots, N_{max}\}$: number of active servers in slot t.

$n_t^{wkup} \in \{0, \ldots, N_{max} - 1\}$: number of servers activated in slot t.

$e_t \in [0, +\infty]$: grid energy consumed in slot t.

Parameters:

W_t^n: amount of non-deferrable workload that should be executed in slot t, assuming all non-deferrable jobs are dispatched immediately on arrival.

W_t^d: amount of deferrable workload that should be executed in slot t, assuming all deferrable jobs are dispatched immediately on arrival.

L_t: minimum amount of deferrable workload that must be executed in slot t to avoid deadline violations.

$Green_t$: green energy available in slot t.

$f_e(W_t^n, w_t^d, n_t)$: total energy consumed by active servers in slot t.

C^{wkup}: capacity loss during a server's start-up.

P: the constant electricity price.

Optimization problem:

$$\min \left\{ \sum_{t=0}^{T} e_t P \right\}, s.t.$$

Capacity constraints:

$$W_t^n + w_t^d \leq n_t C - n_t^{wkup} C^{wkup}, \forall t \leq T$$

$$n_t^{wkup} = \max(0, n_t - n_{t-1}), \forall t \geq 1$$

Cumulative dispatched load cannot exceed cumulative arrived load, cumulative dispatched load must be at least amount needed to avoid deadline violation:

$$L_t \leq \sum_{\tau=0}^{t} w_\tau^d \leq \sum_{\tau=0}^{t} W_\tau^d, \forall t \leq T,$$

$$e_t = max(f_e(W_t^n, w_t^d, n_t) - Green_t, 0), \forall t \leq T$$

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[2] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Manoj Chakkaravarthy, Udit Gupta, David Brooks, and Carole-Jean Wu. 2022. Carbon Explorer: A Holistic Approach for Designing Carbon Aware Datacenters. arXiv:2201.10036

[3] U.S. Energy Information Administration. 2022. August 2022 Monthly Energy Review.

[4] Yixin Bao, Yanghua Peng, and Chuan Wu. 2019. Deep Learning-based Job Placement in Distributed Machine Learning Clusters. In *Proceedings of the IEEE Internaltional Conference on Computer Communications (INFOCOM)*.

[5] Yoshua Bengio. 2012. Practical Recommendations for Gradient-based Training of Deep Rrchitectures. arXiv:1206.5533

[6] Josep L Berral, Íñigo Goiri, Thu D Nguyen, Ricard Gavalda, Jordi Torres, and Ricardo Bianchini. 2014. Building Green Cloud Services at Low Cost. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*.

[7] Andrew Chung, Jun Woo Park, and Gregory R. Ganger. 2018. Stratus: Cost-Aware Container Scheduling in the Public Cloud. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*.

[8] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and Accurate Deep Network Learning by Exponential Linear Units (elus). arxiv preprint arxiv:1511.07289

[9] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the Symposium on Operating Systems Design & Implementation (OSDI)*.

[10] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. 2015. Hawk: Hybrid Datacenter Scheduling. In *Proceedings of the USENIX Annual Technical Conference (ATC)*.

[11] Anuroop Desu, Udaya Puvvadi, Tyler Stachecki, Sagar Vishwakarma, Sadegh Khalili, Kanad Ghose, and Bahgat G. Sammakia. 2021. Latency-Aware Dynamic Server and Cooling Capacity Provisioner for Data Centers. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*.

[12] Nosayba El-Sayed, Ioan A Stefanovici, George Amvrosiadis, Andy A Hwang, and Bianca Schroeder. 2012. Temperature management in data centers: why some (might) like it hot. In *Proceedings of the ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*.

[13] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power provisioning for a warehouse-sized computer. *ACM SIGARCH Computer Architecture News* 35, 2 (2007).

[14] Inigo Goiri, William Katsak, Kien Le, Thu D. Nguyen, and Ricardo Bianchini. 2013. Parasol and GreenSwitch: Managing Datacenters Powered by Renewable Energy. In *Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[15] Inigo Goiri, Thu D. Nguyen, and Ricardo Bianchini. 2015. CoolAir: Temperature- and Variation-Aware Management for Free-Cooled Datacenters. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[16] Íñigo Goiri, Kien Le, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. 2012. GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[18] Sriram Govindan, Di Wang, Anand Sivasubramaniam, and Bhuvan Urgaonkar. 2012. Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[19] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. 2004. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research* 5, Nov (2004).

[20] Fanxin Kong and Xue Liu. 2014. A survey on green-energy-aware power management for datacenters. *ACM Computing Surveys (CSUR)* 47, 2 (2014).

[21] Vasileios Kontorinis, Liuyi Eric Zhang, Baris Aksanli, Jack Sampson, Houman Homayoun, Eddie Pettis, Dean M Tullsen, and Tajana Simunic Rosing. 2012. Managing distributed ups energy for effective power capping in data centers. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*.

[22] Nevena Lazic, Tyler Lu, Craig Boutilier, Moonkyung Ryu, Eehern Wong, Binz Roy, and Greg Imwalle. 2018. Data Center Cooling Using Model-Predictive Control. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

[23] Chao Li, Amer Qouneh, and Tao Li. 2012. iSwitch: Coordinating and optimizing renewable energy powered server clusters. *ACM SIGARCH Computer Architecture News* 40, 3 (2012).

[24] Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yanzhi Wang. 2017. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*.

[25] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. 2012. Renewable and Cooling Aware Workload Management for Sustainable Data Centers. In *Proceedings of the ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*.

[26] Uri Lublin and Dror G. Feitelson. 2003. The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs. *J. Parallel Distrib. Comput.* 63, 11 (2003).

[27] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*.

[28] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning Scheduling Algorithms for Data Processing Clusters. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[29] Hongzi Mao, Shaileshh Bojja Venkatakrishnan, Malte Schwarzkopf, and Mohammad Alizadeh. 2018. Variance Reduction for Reinforcement Learning in Input-driven Environments. arxiv preprint arxiv:1807.02264

[30] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. 2020. Recalibrating Global Data Center Energy-use Estimates. *Science* 367, 6481 (2020).

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-Level Control Through Deep Reinforcement Learning. *Nature* 518, 7540 (2015).

[32] Justin Moore, Jeff Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. 2005. Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers. In *Proceedings of the USENIX Annual Technical Conference (ATC)*.

[33] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. 2020. AWAC: Accelerating Online Reinforcement Learning with Offline Datasets. arXiv:2006.09359

[34] Ehsan Pakbaznia and Massoud Pedram. 2009. Minimizing Data Center Cooling and Server Power Costs. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*.

[35] Yongyi Ran, Han Hu, Xin Zhou, and Yonggang Wen. 2019. DeepEE: Joint Optimization of Job Scheduling and Cooling Control for Data Center Energy Efficiency Using Deep Reinforcement Learning. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*.

[36] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*.

[37] Jürgen Schmidhuber. 2015. Deep Learning in Neural Networks: An Overview. *Neural Networks* 61 (2015).

[38] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust Region Policy Optimization. In *Proceedings of the International Conference on Machine Learning*.

[39] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347

[40] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction*. MIT press.

[41] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*.

[42] Rahul Urgaonkar, Bhuvan Urgaonkar, Michael J Neely, and Anand Sivasubramaniam. 2011. Optimal Power Cost Management Using Stored Energy in Data Centers. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*.

[43] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale Cluster Management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (Eurosys)*.

[44] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[45] Ronald J Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement learning. *Machine learning* 8, 3-4 (1992).

[46] Deliang Yi, Xin Zhou, Yonggang Wen, and Rui Tan. 2019. Toward Efficient Compute-Intensive Job Allocation for Green Data Centers: a Deep Reinforcement Learning Approach. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*.