DeepPM: Efficient Power Management in Edge Data Centers using Energy Storage

Zhihui Shao
University of California, Riverside

Mohammad A. Islam
University of Texas at Arlington

Shaolei Ren
University of California, Riverside

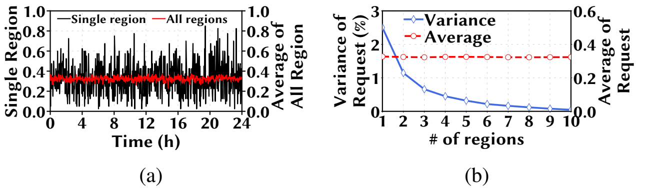
***Abstract*—With the rapid development of the Internet of Things (IoT), computational workloads are gradually moving toward the internet edge for low latency. Due to significant workload fluctuations, edge data centers built in distributed locations suffer from resource underutilization and requires capacity underprovisioning to avoid wasting capital investment. The workload fluctuations, however, also make edge data centers more suitable for battery-assisted power management to counter the performance impact due to underprovisioning. In particular, the workload fluctuations allow the battery to be frequently recharged and made available for temporary capacity boosts. But, using batteries can overload the data center cooling system which is designed with a matching capacity of the power system. In this paper, we design a novel power management solution, DeepPM, that exploits the UPS battery and cold air inside the edge data center as energy storage to boost the performance. DeepPM uses deep reinforcement learning (DRL) to learn the data center thermal behavior online in a model-free manner and uses it on-the-fly to determine power allocation for optimum latency performance without overheating the data center. Our evaluation shows that DeepPM can improve latency performance by more than 50% compared to a power capping baseline while the server inlet temperature remains within safe operating limits (e.g., 32°C).**

***Keywords*-Edge data center, power management, deep reinforcement learning**

I. Introduction

With the emerging Internet of Things (IoT), 5G network and embedded artificial intelligence, computation workloads are gradually moving from the cloud toward Internet edge [3]. Edge data centers can provide computing services with ultra-low latencies, offering great opportunities for latency-critical applications, such as smart cities, augmented reality and intelligent video acceleration [14], [27]. According to Cisco's report [4], approximately 30% of internet workloads will be processed in edge data centers by 2022.

To achieve user proximity and provide low latency computing services, edge data centers are geographically spread to many locations. However, due to the loss of multiplexing at the aggregate (i.e., random fluctuations canceling out each other), it also results in more workload fluctuations in the edge data center as compared to that of a large centralized one. As an example of typical edge data center workload, we look at Uber's rideshare requests in ten different regions of the Boston area [32]. The rideshare requests in each region in the Uber data set can be seen as the typical workload pattern of an edge data center dedicatedly serving the regional users. Fig. 1(a) shows the rideshare request for a single region (Haymarket Square) as well as the average for all the regions normalized to the single region peak. Here we see a staggering variation in regional user requests (i.e., workload) compared to the aggregated of the entire area. Fig. 1(b) further shows that the variation in the workload decreases as we combine more regions together.



(a)

(b)

Figure 1. (a) Normalized Uber rideshare requests in a single region vs average of all regions in the Boston area [32]. (b) Change in variance and average number of requests with region aggregation.

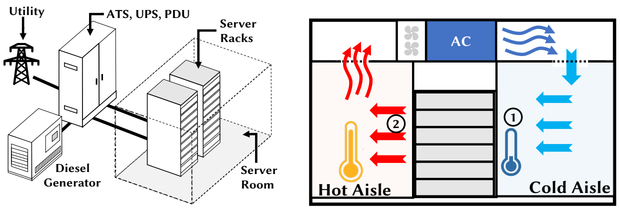
These rapid workload fluctuations can have a detrimental effect on efficient data center management. It leads to resource underutilization and wasted power and cooling capacities when the edge data center is sized to meet the peak demand [16]. Consequently, resource underprovisioning (i.e., allocating less capacity than the peak demand) has been widely adopted in modern data centers to improve efficiency [1], [9]. However, underprovisioning requires power capping to avoid infrastructure overloads when the demand exceeds the capacity. But, since power capping may also adversely affect the latency performance, we need a graceful dynamic power management to facilitate underprovisioning without severely affecting the response latency [1], [9].

In this work, we identify that the rapid fluctuations in edge data center workloads are particularly suitable for employing capacity-constrained energy storage devices such as batteries to temporarily increase the infrastructure capacity during overloads and counter the performance impact due to power capping. More specifically, in edge data centers, the workload spikes are short-lived (Fig. 1(a)) which, if done judiciously, allows the battery to be frequently recharged between overload events and be made available for supplementing the data center capacity. The same, however, is not applicable for larger data centers where slowly changing workload results in extended capacity overloads (e.g., tens of minutes) that the battery cannot sustain (i.e., running out of energy due to the lack of recharging opportunity).

In particular, we aim at using the batteries in data center's uninterruptible power supply (UPS) unit which provides backup power during utility power outages [12], [13]. The main idea here is to store energy (i.e., recharge) in the UPS battery when the power demand is low and use (i.e., discharge) it later at a suitable time, for example when the demand exceeds the capacity. While prior works have also studied using UPS battery for data center power management, a key unwanted pitfall of cooling system overload has been mostly overlooked [12], [13]. Data center cooling system is typically designed with capacity matching that of the power system. Hence, using the extra power from the UPS battery also drives the corresponding server heat generation beyond the cooling system's capacity. When overloaded, the cooling system cannot remove all the heat generated, leading to rapid temperature build-up inside the data center due to heat accumulation. According to [20], the server inlet temperature may increase by more than 10°C if the cooling system is overloaded for 10 minutes. Such cooling system overloads and the ensuing temperature increase can lead to automatic server shutdown to avoid permanent damage and fire hazard. For instance, Dell EMC server will perform a protective shutdown once the server inlet temperature exceeds the threshold of 32°C [6].

Our contributions. In this paper, we develop a novel power management solution for edge data centers to use UPS batteries to boost capacity while also keeping the data center cool. While we exploit the rapid fluctuations and short-lived spikes of edge data center workloads to utilize UPS batteries, we tackle the cooling system overload by exploiting the *cold air* inside the data center as a heat buffer that absorbs transient heat spikes. Concretely, we use the cold supply air (e.g., at 27°C) within the data center as an energy storage by exploiting its temperature difference from the server safe operating limit (e.g., 32°C) to temporarily hold the extra heat generated due to the battery usage.

However, developing a power management solution that effectively exploits both the data center cold air and UPS battery is challenging. First, both the energy stored in the batteries and the temperature of the cold air have *memories*. That is, using the battery now to supplement the power capacity diminishes the available battery for future use. Similarly, an increase in cold air temperature due to absorbing heat spike will leave less room for temperature increase to handle future heat spikes without overheating (e.g., temperature exceeding 32°C). Hence, our power management incorporating the battery and cold air needs to consider future demand in its decisions. Second, to allow safe heat spikes due to battery usage, we also need to accurately predict the impact of heat spikes on cold air temperature by extracting the data center's thermal dynamics. However, thermal modeling for every edge data centers individually is impractical due to their large number and diverse physical locations and operating environments. Not to mention, such thermal models need to be updated every time there is a change in the data center environment (e.g., data center/server layout).



(a)

(b)

Figure 2. (a) Edge data center power delivery system. (b) Edge data center cooling system with cold and hot aisle containment. ①: server inlet temperature T_{in}, ②: server outlet temperature T_{out}.

To solve the aforementioned challenges, we propose a deep reinforcement learning (DRL) based algorithm — DeepPM. We are motivated to use a DRL based solution since it can autonomously learn and incorporate both the future requirements (e.g., capacity boosts from the battery) and dynamics of the data center environment (e.g., temperature change) in its decisions. Moreover, using DRL's model-free approach we can capture greater details of our problem and overcome simplifying assumptions made in existing model-based data center management approaches [30]. In DeepPM, we formulate the power management problem as a Markov decision process (MDP) where the power demand for incoming workload, battery energy, and cold air temperature at the server inlet constitute different MDP states while the action space is the total power allocation. We formalize a parameterized reward function that penalizes for the increase in latency, cold air temperature, and battery energy usage. We use deep Q-learning with long short-term memory (LSTM) network to learn the optimum action at each MDP state. Our MDP formulation together with the LSTM network allows DeepPM to make a decision based on only the current state (i.e., power demand, battery energy level, and cold air temperature), while the deep Q-learning allows us to learn and utilize on-the-fly the data center thermal dynamics in a model-free manner.

To evaluate DeepPM, we compare it with three other baseline algorithms. Our results show that DeepPM can effectively exploit the data center cold air and the UPS battery to provide more than 50% improvement in latency performance while keeping server inlet temperature within 32°C. We also conduct a sensitivity study to see how different settings affect DeepPM's performance.

II. Preliminaries

Power infrastructure. A typical edge data center's capacity ranges from a few kilowatts to a few tens of kilowatts. The data center connects to the power utility and typically also has a backup generator that serves as a secondary power source. An automatic transfer switch (ATS) reroutes the data center power connection from the utility to the generator during a power outage. However, it may take few tens of seconds to a few minutes to bring the generator online [12].

Hence, to support uninterrupted operation during the power switchover, the data center is equipped with a UPS with battery backup. The server racks get power from a power distribution unit (PDU). For small edge data centers, the PDU can be collocated with the UPS and ATS. Finally, the power the distributed to each server form their respective rack PDUs. We show a generic power infrastructure hierarchy for an edge data center in Fig. 2.

Cooling infrastructure. Data centers need dedicated cooling systems to remove the heat generated by servers. Since almost the entire power consumption of the servers converts into heat, data center cooling systems are provisioned with capacity matching the power infrastructure capacity. Due to smaller size, edge data center usually uses a computer room air conditioner (CRAC) as the cooling system. Fig. 2(b) illustrates a typical CRAC cooling system in edge data centers. Here, the CRAC supplies cold air at temperature T_{in} to the server inlet and collects hot exhausting air at temperature T_{out} from the server outlet. To improve CRAC's efficiency, the hot and cold aisle containment can be installed to avoid heat pollution (i.e., hot air mixing with the cold air) [25]. The hot air from the hot aisle is recirculated through the CRAC unit which removes the heat and cool it down to supply cold air to the cold aisle at temperature T_{in}. For improved cooling efficiency, the server inlet temperature is typically conditioned at 27°C, as recommended by ASHRAE [29].

III. Power Management using DeepPM

A. Problem Definition

In this work, we focus on an edge data center hosting multiple server racks, a UPS with battery backup, and a CRAC cooling system. We consider the data center has a total power capacity of C_0. The cooling system capacity is provisioned for the designed power capacity and can supply cold air at temperature T_0 (e.g., 27°C) when the data center power consumption does not exceed C_0. The UPS battery has a maximum recharge rate of R_{max} which is imposed to safeguard against damaging the battery cells. We consider the data center capacity C_0 can be supplemented by discharging the UPS battery using techniques similar to prior work [12], [13].

We use a discrete-time model with a time step Δt (e.g., 10 seconds) where the power management decisions are updated at the beginning of each time step. The server power allocation decision is made based on the power demand from the incoming workloads/requests and the available energy in the battery. Whenever the power allocation is lower than the power demand, the data center utilizes power capping to curb the server power consumption. The decision also takes into account the server inlet temperature to avoid overheating the data center. At time step t, we denote the power demand as p_D^t, the power allocation as p_a^t, battery energy level as b^t, and server inlet temperature as T_{in}^t.

Objective. The target of the data center operator is to dynamically allocate power to improve the data center's overall performance (e.g., latency/response time) without overheating the data center. We formalize the power management as the following optimization problem **OPA** (**O**ptimum **P**ower **A**llocation).

$$\textbf{OPA} : \underset{p_a^t}{\text{minimize}} \quad \sum_t L(p_D^t, p_a^t) \tag{1}$$

$$\text{subject to} \quad T_{in}^{t+1}(p_a^t, T_{in}^t) \le T_{th} \tag{2}$$

$$p_a^t - C_0 \le b^t \tag{3}$$

$$\tag{4}$$

Here, $L(p_D^t, p_a^t)$ is the total latency increase due to a power allocation p_a^t against a demand of p_D^t and T_{th} is the overheating threshold for server inlet temperature. Constraint (2) restricts data center from overheating and constraint (3) limits using the battery beyond its current energy. In addition, whenever the power allocation is less than the capacity, the battery is recharged at the rate of $\min(R_{max}, C_0 - p_a^t)$.

Challenges. Solving **OPA** is challenging because the battery energy and the cold air temperature both have *memories*. Specifically, the change in battery energy and/or cold air temperature due to the power allocation decision made in the current time slot affect the available battery and/or heat absorption capacity of cold air in future time slots. Further, constraint (2) requires that we estimate the server inlet temperature T_{in}^{t+1} in the next time slot based on power allocation (p_a^t) and server inlet temperature (T_{in}^t) of the current time slot. However, modeling the edge data center's thermal behavior using techniques like computational fluid dynamics (CFD) is exhaustive since it requires CFD analysis of a large number of edge data centers operating in diverse environments. Not to mention the update required every time the data center's environment (e.g., server layout) changes.

In what follows, we formulate our problem using a MDP followed by a deep Q-learning based algorithm to solve **OPA** in a model-free manner.

B. MDP Formulation

As the foundation of reinforcement learning-based solution, we first model our problem using a discrete-time MDP where the entire time horizon is divided into time slots $t = 0, 1, \cdots, \infty$. Our system state s^t at time t includes the power demand p_D^t, battery state b^t, and the server inlet temperature T_{in}^t while the action is defined by the power allocation p_a^t. The MDP formulation of our problem can be summarized as follows:

- System state: $s^t = (p_D^t, b^t, T^t) \in \mathcal{S}$
- Action: $p_a^t \in \mathcal{A}(s^t)$
- State transition probability: $P(s^t, p_a^t, s^{t+1})$
- Reward function: $r^t = R(s^t, p_a^t, s^{t+1})$
- Discount factor: $\gamma \in (0, 1)$

Here, the action space $\mathcal{A}$ is determined by the current state s^t. The tuple (s^t, p_a^t, s^{t+1}) means that the system transitions from state s^t to s^{t+1} when action p_a^t is taken. State transition probability function $P(s^t, p_a^t, s^{t+1})$ describes the probability that the system state moves from s^t to s^{t+1} given action p_a^t is taken at s^t. The reward function $R(s^t, p_a^t, s^{t+1})$ defines the immediate reward under state-action tuple (s^t, p_a^t, s^{t+1}).

Action Space. In our MDP formulation, we consider a continuous action space for power allocation with two distinct cases. First, when the power demand is below the data center power capacity, we allocate power for the entire demand, i.e., $p_a^t = p_D^t$. Second, when the demand exceeds capacity, we may supplement the power allocation with battery up to its maximum available capacity, i.e., $p_a^t \leq C_0 + \min(p_D^t - C_0, b^t)$. The eligible action space $\mathcal{A}$ for the current system state s^t can be defined as

$$\mathcal{A} = \begin{cases} p_a^t = p_D^t, & \text{when } p_D^t \leq C_0 \\ p_a^t \leq C_0 + \min(p_D^t - C_0, b^t), & \text{when } p_D^t > C_0 \end{cases} \tag{5}$$

State transitions and Markovian assumptions. For our problem, the battery level b^t and server inlet temperature T_{in}^t both evolve based on the power allocation action taken. The battery level b^t perfectly follows the Markovian process since it changes only when the power allocation action p_a^t exceeds the capacity C_0 and requires battery supplement for catering the demand. However, since the temperature change is a slow process, the time granularity Δt in our problem formulation needs to be sufficiently large for temperature changes to take place to satisfy the MDP assumption. Nonetheless, such restriction on Δt can be lifted at the expense of a larger state-action space by using an augmented multi-level MDP where the next state depends on both the current and recently visited states and actions [15].

The changes in the power demand p_D^t, on the other hand, mainly depends on user behavior and may not correlate with the current state and action, thereby violating the MDP assumption. To facilitate the MDP formulation, we consider that the power demand of next time slot p_D^{t+1} is known at time t through workload estimation [5]. This enables our solution to determine the next state based on current state and action. We add an LSTM network in our design to automate the power demand estimation process.

Reward function. To attain the optimization goals of **OPA**, we devise our reward function as follows:

$$\begin{aligned} r^t = R(s^t, p_a^t, s^{t+1}) = &- L(p_D^t, p_a^t) - \beta_1 (T_{in}^{t+1} - T_{th})^+ \\ &- \beta_2 (b^t - b^{t+1}) \end{aligned} \tag{6}$$

where $(T_{in}^{t+1} - T_{th})^+ = \max(T_{in}^{t+1} - T_{th}, 0)$ is the temperature violation, $b^t - b^{t+1}$ is the battery usage in time slot t, and β_1 and β_2 are wight parameters.

In (6) a decrease in latency is positively rewarded as in **OPA**'s optimization objective. Also, since the battery energy and cold air temperature both have memories, we penalize for battery usage and temperature violation to incorporate their impacts on future decisions. In addition to acting as unit conversion coefficients[1], values of β_1 and β_2 can be tuned to change the emphasis of the optimization goal. For instance, larger values of β_1 will be more restrictive of the temperature violation while increasing β_2 will result in more conservative use of the battery. Note that, the power allocation constraints (3) is satisfied by our action space $\mathcal{A}(s^t)$. The objective of the MDP problem is to find the optimal action policy $\mathcal{A}^*$ for maximizing the long term reward $\sum_t \gamma \cdot r^t$ with a discount factor $\gamma \in (0, 1)$. The discount factor γ is introduced here to have a tractable problem.

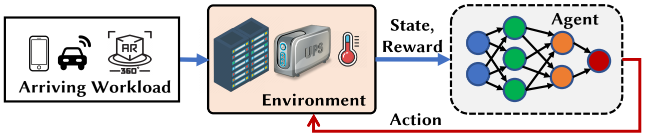


Figure 3. Reinforcement learning system: the agent observes system state, makes action, and receives reward from the environment.

C. *Reinforcement Learning and* DeepPM

Reinforcement learning is a widely used approach to solve MDP problems. Q-learning is one of the reinforcement learning algorithms for solving problems with an unknown environment [31], [34]. Fig. 3 shows the building blocks for reinforcement learning in the context of data center power management. Here, the reinforcement learning agent takes input from the environment to determine its current state which evolves based on the action taken and arriving workload. Next, we briefly discuss Q-learning and then introduce the DRL based power allocation algorithm DeepPM which is implemented with a deep neural network [24].

Q-learning. It is an off-policy reinforcement learning algorithm that can solve model-free MDP problems. In other words, Q-learning can effectively learn the optimal strategy without any prior knowledge of the environment. The learned strategy is represented as a discrete Q value table, which stores Q values for all possible state-action pairs. Then the Q policy π^Q can be extracted as choosing an action with the highest Q value in Eqn. (7).

$$\pi^Q(s^t) = \operatorname*{argmax}_{p_a^t \in \mathcal{A}(s^t)} Q(s^t, p_a^t) \tag{7}$$

The critical task for Q-learning focuses on the estimation of Q values from the environment response. Typically, the Q values can be trained offline with a fixed-point iteration of Bellman equation (Eqn. (8)) for MDP with a known environment. Then, the conventional Q-learning with a learning rate α can be presented as

$$Q(s^t, p_a^t) = Q(s^t, p_a^t) + \alpha[r^t + \gamma Q(s_{t+1}, \pi^Q(s_{t+1}))] \tag{8}$$

[1] Units of battery level and temperature violation are converted to the unit of latency by multiplying with β_1 and β_2, respectively.

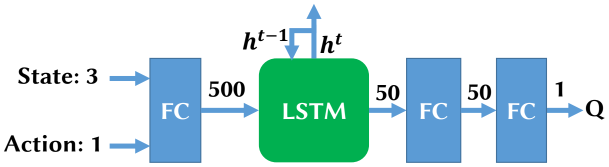


Figure 4. DeepPM using deep neural network. Here, FC stands for fully-connected layer, LSTM stands for Long Short-Term Memory cell, and h_t is the history states of LSTM. It takes the state and action as input and provides the corresponding Q value.

Other methods (e.g., batched Q-learning [36]) have also been investigated to solve specific model-free MDP problems and accelerate the convergence of Q-learning. Usually, Q-learning works well with a small state-action space. It becomes intractable when the state-action space is large or continuous because of either ultra-large Q table or rarely visited state-action pairs. In our power allocation problem, both the states (p_D^t, b^t, T_{in}^t) and action (p_a^t) occupy a continuous space, resulting in an infinitely large Q table. Since the purpose of the Q table is to provide the Q values for a given action, the Q table can be replaced by a deep neural network that acts as an estimator for the Q values.

DeepPM. The basic idea of DeepPM is shown in Fig. 4 where a feed-forward neural network is used to approximate the Q table. The neural network takes the three state parameters (power demand p_D^t, battery state b^t, and server inlet temperature T_{in}^t) and the action (power allocation p_a^t) as the input. Now, as discussed in Section III-B, the power demand p_D^t depends on the incoming user request and may well evolve independently from the state and action. To circumvent the non-MDP nature of the power demand changes, we add an LSTM layer in the feed-forward neural network with a vector h^t that encodes the history states [5]. We initialize h^0 to an all-zero vector. The LSTM layer acts as a predictor for future power demand using history h^{t-1} and allows DeepPM to navigate the state transitions for its actions. In the implementation of DeepPM, we have one fully-connected layer with 500 hidden neurons, one LSTM layer with 50 hidden neurons, one fully-connected layer with 50 hidden neurons and one fully-connected layer with Q value as the output.

The deep neural network is also called Deep-Q-Network (DQN), which can estimate the Q value as $Q(s, p_a|\theta)$, where θ is the weight parameters. Owing to the significant recent developments in deep learning, we can learn the weight parameters θ with high precision using a well-developed gradient descent optimizer (e.g., Adam optimizer [17]). The input of DQN includes the state and action where the output of DQN provides the corresponding Q value. According to (8), the loss functions of DQN are defined in (9) and (10), on training data set $\mathcal{D}$.

$$L(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s^t, p_a^t, s^{t+1}) \in \mathcal{D}} \left(y^t - Q(s^t, p_a^t|\theta)\right)^2 \qquad (9)$$

Algorithm 1: DQN Training Algorithm

```
Input: Greedy exploration parameter $\epsilon$, and
  mini-batch size $B$, $M$, and $N$.
Randomly initialize DQN network $Q(s, p_a|\theta)$ with
  weights $\theta$.
Initialize an empty replay buffer $\mathcal{R}$.
for Epoch = 1 to M do
    Randomly initialize the battery state $b^1$ and
      server inlet temperature $T^1$.
    for t=1 to N do
        1. Select one power allocation $p_a^t$ based on
          current system state $s^t$ according to
          $\epsilon$-greedy policy
            a) randomly select $p_a^t \in \mathcal{A}(s^t)$ with
          probability $\epsilon$.
            b) otherwise, select $p_a^t$ such that
          $p_a^t = \mathrm{argmax}_{p_a \in \mathcal{A}(s^t)} Q(s^t, p_a|\theta)$
        2. Update battery state $b^{t+1}$ and server inlet
          temperature $T^{t+1}$ from sensors.
        3. Calcuate reward $r^t$ based on Eqn. 6.
        4. Store the simulation experience
          $(s^t, p_a^t, r^t, s^{t+1})$ into replay buffer $\mathcal{R}$.
        5. Randomly sample $B$ experience from
          replay buffer $\mathcal{R}$ as the training dataset $\mathcal{D}$.
        6. Update the DQN network $\theta$ via
          minimizing total loss $L(\theta)$ in Eqn. 9.
    end
end
return DQN network with learned weights $\theta$.
```

$$y^t = r^t + \gamma \max_{p_a^t \in \mathcal{A}(s^t)} Q(s^{t+1}, p_a^t|\theta) \qquad (10)$$

To calculate the network target y^t for each gradient descent training iteration, we need to find the optimal action p_a^t (i.e., power allocation) for new state s^{t+1}. We can calculate the optimal action p_a^t for the discrete action space using, for example, exhaustive comparison. However, continuous action space is challenging to deal with because of its complexity in finding the action to maximize Q values. One straightforward approach is to discretize the continuous action space and apply standard DQN. An alternative approach, called Deep Deterministic Policy Gradients(DDPG), is to use a critic-actor policy [18] with two neural networks – a DQN and a deterministic policy gradient (DPG) network. DQN approximates the Q function $Q(s, p_a)$, with state-action as the input and Q value as the output. Whereas DPG approximates the policy function $p_a = \mu(s)$, with states as the input and actions as the output. DDPG is typically applied with high dimensional continuous action space. Since we have only one continuous action variable p_a, we use the discretization approach and split the power allocation into 200 discrete values with reasonably high precision for our problem.

On the other hand, to ensure fast convergence during the DQN training we use experience replay buffer, mini-batch gradient descent, and ϵ-state-action exploration. The experience replay buffer stores all history state-action pairs and can be used in DQN training. Instead of using the entire replay buffer, a smaller subset of transitions is used by the mini-batch gradient descent to minimize the loss in Eqn (9). This is done to avoid using strongly correlated transitions which make the training process unstable [5]. In addition, since the LSTM model is being updated continuously, we cannot directly use saved history h^{i-1} for each picked transition i. Instead we use all the historical states leading up to transition i as input to our LSTM model to get the correct h^{i-1} [5], [28]. For action selection, ϵ-greedy exploration policy is utilized to balance the dilemma between exploration and exploitation. Specifically, with a probability of ϵ, a random power allocation $p_a^t \in \mathcal{A}(s^t)$ is chosen. Otherwise, the action is chosen following (7).

In our implementation, we initialize an empty replay buffer $\mathcal{R}$ at the beginning of training. Then the agent explores a preset number of state-action pairs with a greedy parameter ϵ at each time slot t, observes the environment responses, and then stores the experiences $(s^t, p_a^t, r^t, s^{t+1})$ into the replay buffer $\mathcal{R}$. In contrast to traditional Q-learning, the agent of DRL updates weight parameter θ with a mini-batch sampling from the replay buffer $\mathcal{R}$. The detail of the DQN learning algorithm is presented in algorithm 1.

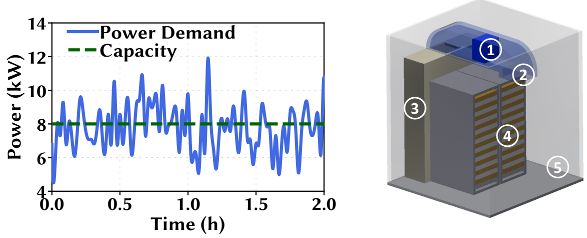
IV. Evaluation Methodology

In this section, we present our default edge data center settings, thermal dynamics, battery energy model, and performance model. We then describe the implementation and parameters for DeepPM. We finally introduce three benchmark algorithms for the evaluation of DeepPM.

A. Settings

Due to limited access to a real edge data center, we resort to a simulation-based evaluation for DeepPM where the power management decisions are updated every 10 seconds.

Data center infrastructure. We consider an edge data center with two server racks each with 20 servers and a designed capacity (C_0) of 8kW. The data center has a UPS backup with battery capacity (C_B) of 0.2kWh which can provide 1.5 minutes of backup power at its maximum discharge rate of 8kW. The maximum recharging rate of the battery (R_{max}) is set to 0.5kW. The data center is cooled using a CRAC (computer room air conditioning) system with a matching designed capacity of 8kW. The CRAC can supply cold air at $T_0 = 27$°C when the data center power consumption remains within its capacity of 8kW. We set the threshold temperature (T_{th}) to 32°C. We also consider the data center employs hot aisle containment to improve cooling efficiency and avoid heat recirculation.



(a) (b)

Figure 5. (a) 2-hour snapshot of workload/power demand trace. (b) CFD simulation model ① Air conditioner (AC). ② Supply air duct. ③ Heat containment. ④ Server racks. ⑤ Server room.

Workload trace. We use real-world traces to perform training and evaluation of DeepPM. We use the number of requests from two popular rideshare applications, Uber and Lyft, in the Boston, Massachusetts area as our workload [32]. Because of their large geographic service areas, customer requests from such rideshare applications are good examples of edge data center workloads. We convert the number of requests to power demand using the server power consumption model of [33] and scale the power trace to have a peak demand of 12kW. Fig. 5(a) shows a 2-hour snapshot of the workload trace used in our simulation.

B. Environment Models

For training and evaluation of DeepPM agent, we simulate the edge data center which is accessed by the DRL for learning the state transitions for its state-action pairs.

Thermal model. We use CFD which is one of the most widely used approaches to analyze the thermal behaviors of data centers (e.g., Google [11]). However, transient CFD simulation is slow and computationally exhaustive. Hence, as outlined in [30], we adopt a short-term CFD approach using Autodesk CFD where the thermal environment is modeled using impulse response by creating power spikes. The 3D model used in Autodesk CFD is shown in Fig. 5(b). As an illustration, in Fig. 6(a), we show the temperature change for our data center with a 1.5kW cooling overload for 10 minutes using our thermal model.

Battery Energy Model We consider a liner battery charging/discharging model where the battery energy state b^t is adjusted by subtracting discharge power and adding recharge power as follows:

$$b^{t+1} = \begin{cases} b^t + \min(C_0 - p_a^t, R_{max}), & \text{for } p_a^t < C_0 \\ b^t - (p_a^t - C_0), & \text{otherwise} \end{cases} \quad (11)$$

Note that, a nonlinear battery model incorporating charging/discharging loss and leakage can also be considered in the state transition without loss of generality.

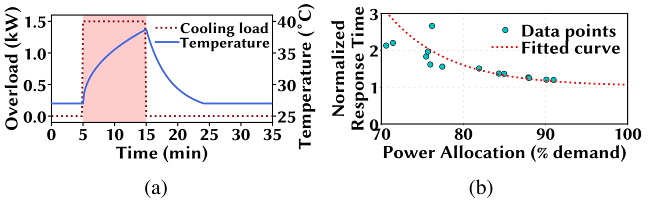


Figure 6. (a) Thermal model illustration with 10-minute cooling overload. (b) Latency model for partially satisfied power allocation.

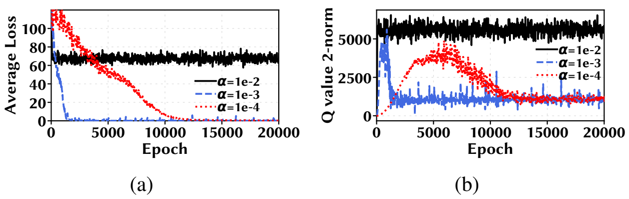


Figure 7. Convergence curve for DQN training: (a)Average training loss. (b) 2-norm of Q values.

C. Latency Model

We determine the latency increase due to the power capping based on web search application experiments in [16]. Specifically, we run the web-search benchmark from CloudSuite [10] for different workload levels and CPU speeds (using DVFS) and collect the 99-th percentile response times.Then, considering the power consumption at full CPU speed as the power demand for a given workload, we model the performance impact (change in response time/latency) for change in the power allocation as a percentage of the power demand. We show the performance for different power allocations in Fig. 6(b) where the response time is normalized to that of without any power capping. Note that, here we use web-search as an example of performance-power trade-off while many other applications also have similar performance-power relationship [16]. Hence, our proposed approach can be used for a wide range of applications.

D. DRL Parameters

For DRL, we implement the DQN training algorithm using *"Tensorflow"* [7]. The DQN ($Q(s, a|\theta)$) is implemented with a four-layer fully-connected feed-forward neural network, which includes 500 nodes in the first hidden layer, and 50 nodes in the second and third hidden layer. The *"ReLU"* is utilized as the activation function for the three hidden layers to achieve nonlinearity.

For the training process of DQN, we use Adam optimizer with a learning rate of $\alpha = 0.001$, and the mini-batch size is set at $B = 1024$. The discount factor is set at $\gamma = 0.9$. The weight factors in reward Eqn. (6) are set as $\beta_1 = 100$ and $\beta_2 = 0.1$ for the penalty of overheating and battery usage, respectively. We perform 6000 training epochs for DQN.

E. Benchmark Policies

We evaluate DeepPM against three benchmark policies – PowerCap, Greedy, and GreedyT, described as follows.

PowerCap. It does not utilize the UPS battery. When demand exceeds the capacity, it caps the power at capacity (i.e., $p_a^t = C_0 < p_D^t$). Otherwise, it allocates power for the full demand (i.e., $p_a^t = p_D^t$). In our evaluation, PowerCap is the baseline policy since it does not employ any performance improvement strategy such as utilizing the UPS battery.

Greedy. It uses the battery greedily without considering its impact on temperature. When demand exceeds the capacity, it uses the battery to cover the deficit amount up to the available energy in the battery, i.e., $p_a^t - C_0 \leq b^t$.

GreedyT. It has the same power allocation strategy as Greedy, except, it only uses the battery if he server inlet temperature is less than 31.5°C.

When the power demand is lower than the capacity, both Greedy and GreedyT satisfy the demand. They also recharge the battery with the unused capacity, $C_0 - p_D^t$.

V. Evaluation Results

In this section, we present results from DQN training and compare DeepPM with the three benchmark algorithms. Then we conduct sensitivity studies on DeepPM.

A. DQN Training

We use the Uber and Lyft workload traces and utilize the thermal model to train DeepPM for 20,000 epochs ($\sim$ 55 hours or little over 2-days with 10 seconds time slots). We use three different learning rates: $\alpha = 0.01$, $\alpha = 0.001$, and $\alpha = 0.0001$ with a mini-batch approach. We sample the average loss for every ten epochs and show the convergence curves of in Fig. 7(a). We also show the evolution of 2-norm of the Q values in Fig. 7(b). We can see that while training does not converge for $\alpha = 0.01$, it converges after $\sim$2000 epochs ($\sim$ 6 hours) for $\alpha = 0.001$ and $\sim$13,000 epochs ($\sim$ 36 hours) for $\alpha = 0.001$. We chose $\alpha = 0.001$ as our default learning rate since it results in a earlier convergence.

B. Illustration of Power Allocation

We show a 4-hour snapshot of the power allocation under the three different policies (Greedy,GreedyT, and DeepPM) in Fig. 8. We include the power demand due to the incoming workloads, the power allocation, battery energy level, and server inlet temperature variation. The snapshot starts with a fully charged battery with 100% energy and a server inlet temperature of 27°C. Whenever the power allocation is above the capacity of 8kW, the data center uses its battery power and the server inlet temperature goes up due to the extra heat generated beyond the cooling capacity. A gap between demand (blue line) and allocation (red lines) indicates that the power allocation does not meet the power demand, resulting in processing latency.

First, we look at Greedy policy which allocates power from battery whenever the demand is higher than the capacity. It disregards the temperature increase and ends up with

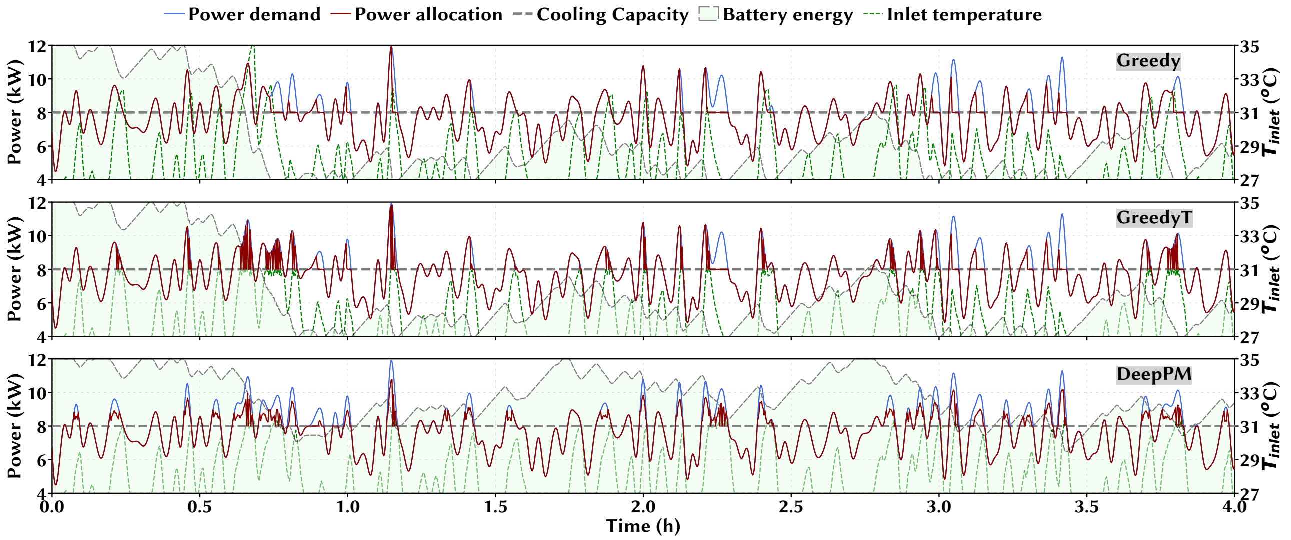
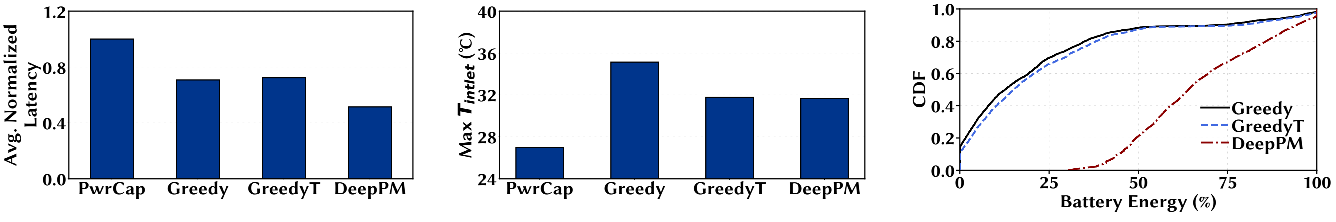


Figure 8. Policy illustration: 4-hour snapshot of workload allocation and environment response.



(a)

(b)

(c)

Figure 9. Performance evaluation with different algorithms.

server inlet temperature going over 32°C many times (e.g., multiple times near 0.5 hours). For DeepPM, on the other hand, we see that it does not allocate power for the full demand when the server inlet temperature approaches the threshold of 32°C, for example near 0.5 hours. Restricting itself within the temperature limit is the cause of a slight increase in latency performance for DeepPM.

We also see in Fig. 8 that the battery energy is frequently replenished due to the rapid fluctuations in power demand. Also, DeepPM uses the batteries more conservatively and can maintain a high battery level most of the time while Greedy nearly depletes it around 0.5 hours.

C. Performance Evaluation

Here we evaluate DeepPM for a period of 24-hours with a trained DQN discussed in Section V-A.

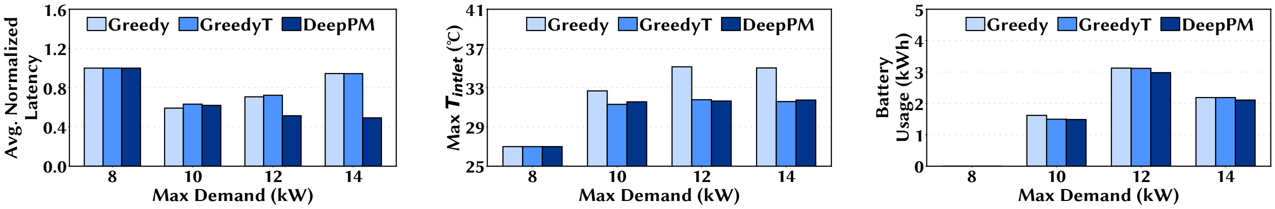
Average Latency. We calculate the latency following our latency model in Section IV-C and normalize to that of PowerCap. We take the average latency performance for the overload time slots to emphasize the true impact of DeepPM. We show the average latencies for the four policies in Fig. 9(a). We see that DeepPM, as compared to the baseline PowerCap, achieves more than 50% lower latency. Greedy and GreedyT also enjoy over 30% performance improvements because they use the battery. Nonetheless, both have a significantly less improvement since they greedily consumes the limited battery capacity, as opposed to DeepPM which takes the future needs into account.

Server inlet temperature. Next, we look at the maximum server inlet temperature resulting from the different policies in Fig. 9(b). We see that both DeepPM and GreedyT manage to maintain a server inlet temperature below the threshold limit of 32°C as opposed to Greedy which results in a maximum inlet temperature of more than 35°C. This is because Greedy does not take the temperature into account during power allocation. PowerCap, on the other hand, has the lowest inlet temperature as it never uses the battery.

Battery usage. Next, we look at how the three battery using policies utilize the battery. We show the CDF of battery energy levels in Fig. 9(c). We see that nearly 20% of the time both Greedy and GreedyT run without any available battery to supplement the power allocation. DeepPM, on the other hand, barely drops below a 40% energy level and therefore retains the battery energy to use when the power demand exceeds the data center capacity

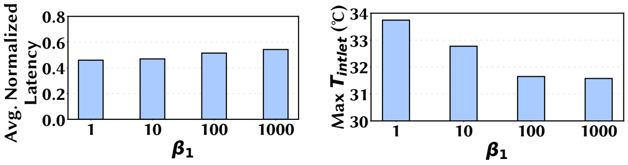
D. Sensitivity Analysis

Here we examine the impacts of peak power demand and weight parameters in the reward function on DeepPM.



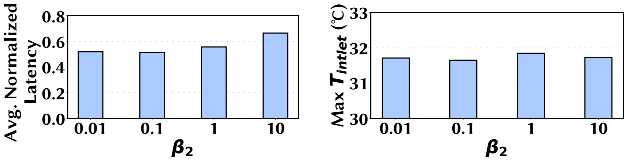
(a) (b) (c)

Figure 10. Impact of peak power demand on DeepPM.



(a) (b)

Figure 11. Effects of weight parameters β_1.



(a) (b)

Figure 12. Effects of weight parameters β_2.

1) Impact of power demand: We vary the peak power demand from 8kW to 14kW by scaling the data center dynamic power. We keep the data center power and cooling capacity at 8kW and the battery capacity at 0.2kWh. The results are shown in Fig. 10 where we omit PowerCap which does not use the battery.We see in Fig. 10(a) that at power demand of 8kW, the latency of all three benchmark algorithms are the same as PowerCap since there is no need for a capacity boost. However, as the demand increases, using the battery becomes useful in improving the latency performance. In Fig. 10(b) we see that the maximum server inlet temperature increases as the peak demand increases while both DeepPM and GreedyT maintain a temperature below 32°C. Finally, Fig. 10(c) shows that at 8kW peak demand (i.e., no underprovisioning) no battery is used while the battery usage increases with peak demand. Interestingly, the battery usage goes down for 14kW peak demand. This is because of the limited recharge opportunity due to higher power demand.

2) Impact of weight parameters β_1 and β_2: The weight parameters in DeepPM's reward function (6) play important roles since they determine how much DeepPM emphasizes on using the battery and violating the temperature constraint as opposed to allowing an increase in latency due to power capping during overloads.

First, we vary β_1 from 1 to 1000 while β_2 is kept constant at its default value 0.1. Since β_1 determines the relative weight of the data center temperature violation, a lower value allows DeepPM to violate the temperature constraint more and vice versa. Consequently, we see in Fig. 11(b) that the the maximum inlet temperature increases with decreasing β_1. Likewise with the temperature violation constraint relaxed at lower values of β_1, DeepPM uses more battery and can result in lower latency (Fig. 11(a)).

Next, we vary β_2 from 0.01 to 10 while β_1 is kept constant at its default value of 100. Since, β_2 is the weight for battery usage, increasing β_2 results in a decrease in the battery usage and vice versa. We see in Fig. 12(a) that the latency performance β_2 has marginal impact of latency performance. On the other hand, as shown in Fig. 12(b), the decrease in battery usage with the increase in β_2 leads to reduction in server maximum inlet temperature.

The take away from our evaluation is that, *edge data center operation can be supplemented with UPS batteries for significantly improving performance by exploiting the rapid fluctuations in workloads and cold air as a thermal buffer.*

VI. Related Work

Data center management. Managing the data center infrastructure with limited resources has received significant attention from the research community in the past decade. Various techniques have been proposed to aid data center management such as improving the energy proportionality [19], [23], jointly managing servers and non-IT support infrastructure (e.g., power/cooling) [19], [23], and exploiting geographical diversity to minimize data center operation cost [26]. Likewise, infrastructure oversubscription is exploited in other works to improve the data center utilization [35]. [21], [38] use energy storage devices to temporarily increase the data center power capacity. Such performance-boosting techniques allow the power consumption to temporarily exceed the data center capacity to offer a performance lift. Constrained by the thermal design power (TDP), [8] proposes temporary power/performance boosts at the microprocessor level by utilizing phase change material and heat absorption of thermal packages. As opposed to prior works, we focus on emerging edge data centers and, instead of exploiting microprocessor-level thermal inertia, utilize the data center level thermal mass in coordination with the UPS battery and propose a DRL-based solution.

Reinforcement learning for resource management. The autonomous learning and online decision capability make reinforcement learning a prime choice for solving data center resource management problems. Due to the recent advances in deep neural network-based learning, recent works focus on DRL based approaches. For instance, [37] proposes a DRL based algorithm to schedule compute-intensive workloads for energy minimization, while [2] designs a task scheduler and resource provisioning system for large cloud service providers. In [22], DRL is used for the job and virtual machine allocation in the cloud data center and [5] uses DRL to capture user behavior for profit maximization of a cloud service provider. Our key novelty is that we focus on edge data centers with rapidly fluctuating workloads, for which energy storage – both thermal and battery – is exploited for performance maximization subject to capacity constraints.

VII. Conclusion

In this work, we developed a novel power management algorithm, DeepPM, for edge data centers that exploits the data center cold air and workload fluctuations to use UPS batteries for capacity boosting without overheating the data center. We used DRL to estimate the data center thermal behavior in a model-free manner and utilized it for deciding power allocation to improve latency performance while keeping server inlet temperature within safe operating limits. We showed that DeepPM can achieve a performance improvement of more than 50% compared to the power capping baseline.

Acknowledgment

This work was supported in part by the NSF under grants CNS-1551661, ECCS-1610471, and CNS-1910208.

References

[1] Luiz André Barroso and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 4(1):1–108, 2009.

[2] Mingxi Cheng, Ji Li, and Shahin Nazarian. DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In *ASP-DAC*, 2018.

[3] M. Chiang and T. Zhang. Fog and IoT: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

[4] VNI Cisco. Cisco visual networking index: Forecast and trends, 2017–2022. *White Paper*, 2018.

[5] Bingqian Du, Chuan Wu, and Zhiyi Huang. Learning resource allocation and pricing for cloud profit maximization. In *AAAI*, 2019.

[6] Dell EMC. idrac 8/7 v2.40.40.40 user's guide. 2017.

[7] Abadi Martin et, al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[8] Songchun Fan, Seyed Majid Zahedi, and Benjamin C. Lee. The computational sprinting game. In *ASPLOS*, 2016.

[9] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ISCA*, 2007.

[10] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Acm sigplan notices*, 47(4):37–48, 2012.

[11] Google. Google's Data Center Efficiency. http://www.google.com/about/datacenters/, 2019.

[12] Sriram Govindan, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Benefits and limitations of tapping into stored energy for datacenters. In *ISCA*, 2011.

[13] Sriram Govindan, Di Wang, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters. In *ASPLOS*, 2012.

[14] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing - a key technology towards 5g. *ETSI white paper*, 2015.

[15] Hyeong Soo Chang, P. J. Fard, S. I. Marcus, and M. Shayman. Multitime scale markov decision processes. *IEEE Trans. on Automatic Control*, 48(6):976–987, 2003.

[16] Mohammad A. Islam, Xiaoqi Ren, Shaolei Ren, and Adam Wierman. A spot capacity market to increase power infrastructure utilization in multi-tenant data centers. In *HPCA*, 2018.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[19] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. In *INFOCOM*, 2011.

[20] Paul Lin, Simon Zhang, and Jim VanGilder. Data center temperature rise during a cooling system outage. *White Paper of Schneider Electric Data Center Science Center*, 2013.

[21] Longjun Liu, Chao Li, Hongbin Sun, Yang Hu, Juncheng Gu, Tao Li, Jingmin Xin, and Nanning Zheng. HEB: Deploying and managing hybrid energy buffers for improving datacenter efficiency and economy. In *ISCA*, 2015.

[22] Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yanzhi Wang. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *ICDCS*, 2017.

[23] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. In *ISCA*, 2011.

[24] 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. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[25] John Niemann. Hot aisle vs. cold aisle containment. *American Power Conversion, West Kingston, RI, APC White Paper*, 2008.

[26] Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. Cutting the electric bill for internet-scale systems. In *SIGCOMM*, 2009.

[27] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[28] D. R. Song, C. Yang, C. McGreavy, and Z. Li. Recurrent deterministic policy gradient method for bipedal locomotion on rough terrain challenge. In *ICARCV*, 2018.

[29] Robin A. Steinbrecher and Roger Schmidt. Data center environments: Ashrae's evolving thermal guidelines. *ASHRAE Technical Feature*, pages 42–49, December 2011.

[30] Qinghui Tang, Sandeep Kumar S. Gupta, and Georgios Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Trans. Parallel Distrib. Syst.*, 19(11):1458–1472, 2008.

[31] John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 1994.

[32] Uber and Lyft. Uber and lyft dataset boston - from 11-26-2018 to 12-18-2018. https://www.kaggle.com/brllrb/uber-and-lyft-dataset-boston-ma, 2019.

[33] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely. Dynamic resource allocation and power management in virtualized data centers. In *NOMS*, 2010.

[34] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.

[35] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Raymond Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. Dynamo: Facebook's data center-wide power management system. In *ISCA*, 2016.

[36] Jie Xu, Lixing Chen, and Shaolei Ren. Online learning for offloading and autoscaling in energy harvesting mobile edge computing. *IEEE Trans. on Cognitive Communications and Networking*, 3(3):361–373, September 2017.

[37] D. Yi, X. Zhou, Y. Wen, and R. Tan. Efficient compute-intensive job allocation in data centers via deep reinforcement learning. *IEEE Trans. on Parallel and Distributed Syst.*, 31(6):1474–1485, 2020.

[38] Wenli Zheng, Kai Ma, and Xiaorui Wang. Exploiting thermal energy storage to reduce data center capital and operating expenses. In *HPCA*, 2014.