Scheduling in Data Centers Running on Renewable Energy with Deep Reinforcement Learning

Vanamala Venkataswamy
Computer Science, University of Virginia
Charlottesville, Virginia.
Email: vv3xu@virginia.edu

Andrew Grimshaw
Computer Science, University of Virginia
and Lancium Inc., Houston, TX.
Email: grimshaw@virginia.edu

***Abstract*—Most data centers operate on electricity generated using non-renewable sources (brown energy). Electricity cost is a significant part of the operational expense in data centers. Higher electricity costs directly translate to the end-users paying higher prices to use the cloud services or reduced service providers' profits. To reduce the cost to end-users and make higher profits, cloud service providers need to reduce their operational expenses. Renewable energy sources are increasingly becoming a viable electricity source that can dramatically lower electrical power costs and achieve dramatic reductions in climate impact. The green data centers are co-located at the sources and powered by renewable energy. The green data centers are not restricted to using a single source of renewable energy; instead, they can utilize multiple renewable energy sources (like wind and solar).**

Using renewable energy sources to power the data centers has challenges. For instance, wind flow is not continuous and not uniform across all regions, time of day, or all seasons. The data centers running on renewable energy sources need smart systems and system-software that adapt to the power variability to ensure that cloud services are available even when there is transient power unavailability in these data centers. Three significant issues that need addressing are 1) Meeting Service Level Objectives (SLOs), 2) Managing the resources, and 3) Adapting to power variability.

Hand-engineering domain-specific heuristics-based schedulers to meet specific objective functions is time-consuming, expensive, and needs extensive tuning in this dynamic environment. We applied Deep Reinforcement Learning (DRL) to automatically learn effective job scheduling policies while continuously adapting to the complex dynamic environment. The DRL based scheduler's objective function is to maximize the total value from jobs (maximizing service providers' revenue).

I. Introduction

Data centers today are powered by electricity generated using primarily non-renewable energy sources (brown energy). Electricity cost is a significant contributor to the total operational cost in data centers [1][2]. Data centers in the U.S use 1.8% of total electricity [3]. One way to reduce the costs is to reduce the operational expense in the data centers. Since electricity is a significant piece of operating expense in a data center [2][4], using renewable energy sources (green energy) to run data centers is becoming a necessity and a reality [5][6][7].

Besides reducing the operational cost, renewable energy sources achieve dramatic reductions in climate impact. Wind energy is one of the lowest-priced and cleanest energy sources available today [9]. Electricity generation from wind farms in the U.S increased from $\sim$ 6 billion kilowatt-hours (kWh) to $\sim$ 275 billion kWh between 2000 and 2018 [8]. By 2050, wind energy could avoid 12.3 gigatons of greenhouse gases [10].

All of the renewable energy generated is not consumed effectively, leading to over-generation problems [11]. A practical way to use renewable energy in a cost-efficient manner is by colocating the data centers right in place with or close to the energy generation, eliminating the associated transmission costs and risks. Green data centers like [25][26][27] use excess power for computing, solving the over-generation problems caused by renewable energy sources.

Green data centers co-located at renewable energy generation facilities draw load directly from the source (e.g., wind farms, solar farms) when power prices are low and drop consumption when power is unavailable or expensive (i.e., during demand peaks). This revolutionary data center model results in tremendous cost savings in terms of power utilization and operational costs. The green data centers can provide cloud services at a lower cost to users.

In this setting, the challenge is to meet user expectations (as expressed by SLOs) with the available resources subject to power availability at the data center and energy production forecast [34] inaccuracy. The degree of inaccuracy varies from one renewable energy source to another requiring smart system software to carefully balance the compute load and available power from multiple sources. The two challenges that need addressing while scheduling jobs are: ***1)*** *Initial task placement, and* ***2)*** *Based on predicted power availability and SLOs deciding which jobs to suspend/restart, migrate (and where to migrate), or terminate.*

Optimizing the value for users means to optimize what the users are willing to pay, expressed via SLOs, and therefore maximizing the revenue for the service provider. The scheduler's goal is maximizing the value subject to resource constraints driven by inaccurate power availability at the sites. In summary, value (*revenue*) is an increasing function of the SLOs; minimizing SLO violations leads to maximizing customer value expressed by the QoS level(§III-B2).

Traditionally, dynamic, online scheduling problems have used heuristics-based scheduling techniques where system engineers design algorithms to capture multiple and diverse problems. Reasoning about various heuristics' interactions is complicated and becomes intractable as the number of vari-

ables and heuristics increases. The goal of autonomic resource management is to reduce the degree of human involvement in managing complex and dynamic infrastructure in a data center. Preferably, a human would only specify a broad, high-level objective as input to the system's resource management algorithms. While the system runs, the management algorithms would continually sense the system state and execute management actions that optimally achieve the specified high-level objective. We duly note that the design and implementation of complex computing systems' accurate performance models can be highly knowledge-intensive and labor-intensive.

Recent works [12][13][14][15] propose resource management using DRL to learn resource management policies automatically. A "policy" is a mapping from system states to management actions. In the most basic form, RL provides a knowledge-free trial-and-error methodology in which a learner (agent) tries various actions in numerous system states and learns from the consequences of each action[16]. There have been many notable successful applications of RL over the last decade in real-world problems ranging from helicopter control to financial markets trading to world-championship game playing [17][18][19].

Given a set of jobs and resources, scheduling jobs optimally is an NP-hard problem. Scheduling in green data centers encounters additional complexity because the availability of resources is not constant due to intermittent and varying power availability. The related work in [12][13][14][15] focuses on resource management in typical data centers where power is constant (brown energy).

This paper demonstrates that DRL provides a promising approach to resource management in green data centers that differ from standard heuristics approaches that use explicit system models. DRL can automatically learn high-quality resource management policies without an explicit performance model. We have implemented and tested our approach in a simulated green data center setting, in which resources are dynamically allocated among multiple online jobs with varying resource and QoS requirements to maximize the expected sum of SLO payments in each job. We tested our approach for varying job arrival rates and varying power availability at the green data center. In each case, we find that DRL trained models provide better performance compared to heuristics-based algorithms in the dynamic green data center environment.

Contributions: We show that,

- A DRL based scheduler generates efficient scheduling policies in a complex dynamic green data center environment.
- The obtained scheduling policies perform 30%–70% better than traditional and greedy heuristic algorithms.
- The DRL scheduler adapts efficiently to the power variability in the green data center.

The remainder of this paper is organized as follows: In §II, we present the closely related works, and in §III, we present the proposed strategy to obtain online scheduling policies using DRL. We present the main results in §IV and the conclusions in §V.

II. Related Work

The first RL formulation of job scheduling was proposed in [20]. The authors describe a local, static, central, suboptimal RL based scheduling in this work. The scheduler is a repair-based in that it starts with a critical-path schedule and repairs constraint violations incrementally to find the shortest conflict-free schedule. The authors propose temporal difference (TD) leading algorithms to train a neural network that learns a heuristic evaluation function over states. The TD evaluation function is a one-step look-ahead search procedure to find reasonable solutions to scheduling problems. A small number of tasks for NASA space shuttle payload processing were evaluated, and their results confirmed that RL could provide a new approach for constructing high-performance scheduling systems.

DeepRM [14] is a global, dynamic, central, suboptimal (approximation) scheduler implemented using a DRL framework. This work describes the design and demonstrates a simple multi-resource cluster scheduler. DeepRM operates in an online setting where jobs arrive dynamically and not-preempted once scheduled. DeepRM learns to optimize various objectives, such as minimizing average job slowdown or completion time. DeepRM employs a standard Policy Gradient (REINFORCE [30]) algorithm to learn from experience.

Minerva [15] is a global, dynamic, central, suboptimal scheduler for bottleneck detection in distributed factory settings. The work uses a similar technique as [18] in that it schedules jobs for a fixed interval, starting with scheduling bottleneck jobs and then rearranging the jobs that violate constraints until no more violations exist in the schedule. This work implements a neural network-based Q-function approximation and Q-learning algorithm.

cuSH [13] is a High-Performance Cluster Scheduler using RL to schedule jobs on heterogeneous clusters. The work is primarily influenced by [14] and implements similar DRL (CNN) techniques for finding near-optimal scheduling policy. This scheduler addresses scheduling for a single HPC cluster and not geographically distributed clusters.

Decima [12] uses DRL and neural networks to learn workload-specific scheduling algorithms without any human instruction beyond a high-level objective, such as minimizing average job completion time. Decima demonstrates new representations for jobs' dependency graphs to support jobs running on the Spark framework. Decima focuses on jobs with dependency graphs on in Spark framework [31].

Harmony [21] is a DRL cluster scheduler that places Machine Learning (ML) training jobs to minimize interference and maximize performance (training completion time). The DRL employs the actor-critic algorithm to stabilize training and improve convergence, job-aware action space exploration, and experience replay. Harmony employs an auxiliary reward prediction model trained using historical samples to produce a reward for unseen placement. Harmony focused exclusively on scheduling ML jobs.

All the systems above assume that power is constant in the data centers, i.e., server and network failures are the only

failure modes. While those failures still exist in the green data center environment, we focus on dynamically scheduling jobs considering power variability in the green data centers.

III. Job Scheduling with Simulation and DRL

Scheduling jobs optimally, given a set of jobs and resources, is an NP-hard problem. Scheduling in green data centers encounters additional complexity because the availability of resources is not constant due to intermittent and varying power availability. Carefully designing and implementing accurate job scheduling policies for such a complex environment can be highly knowledge-intensive and labor-intensive.

We tackled the online problem of scheduling a set of jobs in a single data center setting. This work's general idea is to employ DRL to observe the scheduling behavior with varying power availability and job arrival rates. We present the scheduling agent's performance for various power availability and job arrival rates, where the objective function maximizes the overall value for jobs.

A. General Scheduling Problem

The job scheduling problem is determining when and where to schedule jobs. Given a set of jobs, J, and the set of available resources, R, the scheduler's task is timing and ordering J jobs on R resources. The cardinality of jobs, $|J|$, and resources, $|R|$, change over time. New jobs can be added to or removed from set J, and resources can be added to or removed from set R. There are many combinations of assigning jobs to resources, and the scheduler's job is to find the best possible schedule, such that a predefined objective function is satisfied. At any given time, a job may or may not be mapped on to a resource.

For this work, based on [22], we categorize the DRL based scheduler as global, non-optimal, online, dynamic, and centralized scheduler with an assumption that job preemption (work-preserving) and migration are possible.

B. Deep Reinforcement Learning and DRL based Scheduler

The fundamental principle of RL is that an agent tries to maximize a reward signal by trial and error. Fig. 1. shows a general DRL setting where the Scheduler agent interacts with the environment. The environment or system state consists of available resources and jobs. At each time step t, the agent observes the environment s_t and chooses an action a_t. For the chosen action, the state of the environment transitions from s_t to s_{t+1}, the agent receives a reward r_{t+1} for that action. The state transitions and corresponding rewards are stochastic and have the Markov property - the state transition probabilities and rewards depend on the state of the environment s_t and the agent's action a_t at time t.

1) Policy space: The act of selecting an action a_t in each state s_t is called "policy" denoted as π. The agent selects the next actions based on a policy (π). Policy is a probability distribution over actions $\pi : \pi(s, a) \rightarrow [0, 1]$. Thus $\pi(s, a)$ is the probability that an action a is taken in state s. There are many possible (s, a) pairs, exponential in our case. Therefore, it is impractical to store the policy in a tabular

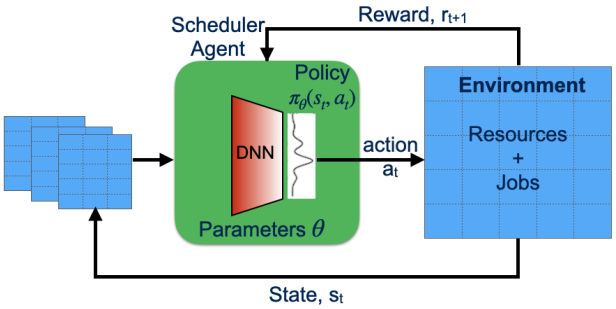

Fig. 1. DRL scheduler environment and agent interaction

format. Instead, we use a function approximator, Deep Neural Network (DNN). A function approximator has considerably fewer number of parameters, θ, represented as $\pi_\theta(s, a)$. The idea is by approximating the policy; the agent will take similar actions for similar states. The policy-gradient method gives the directions in which the parameters must be adjusted to improve a given policy's performance. In essence, policy gradient is increasing the probabilities for good actions and decreasing those of "bad" actions in the policy distribution. For this work, we applied the Advantage Actor-Critic (A2C) policy gradient method, a synchronous version of A3C [31]. The process of training DRL agents is just optimizing the objective function. The objective for the DRL agent is to get the maximum expected cumulative discounted reward:

$$E[\sum_{i=1}^{\infty} \gamma_t r^t] \tag{1}$$

by taking the gradient of the objective function, where γ is the discount factor.

2) Scheduler Agent's Environment: The DRL Scheduler Agent's environment includes available resources and jobs. We consider the data center as constituted by a set of homogeneous resources connected by network interconnection topology, and the jobs arrive over time (i.e., on-line arrival) in a centralized job pool.

A job, J_i, is some workload with the following meta-data:

- *Resource_request* $(type\text{–}1, type\text{–}2)$. The number of units of $(type\text{–}1, type\text{–}2)$ resource sub-types.
- *Enter_time,* e_t. The time when a job entered the system.
- *Start_time,* s_t. The time when the job first started execution.
- *Expected_runtime,* r_t. Job's total runtime as specified by the user.
- *Current_runtime*. Duration for which the job was running on some resources.
- *Max_allowed_runtime*. The maximum allowed amount of time the job can take.
- *QoS_level*. The percentage of wall-clock time the user-specified for this job.
- *Job_value*. The measure of total dollar value when the job completes without violating the job's SLO.
- *Remaining_slack_time*. The remaining time before the job violates the SLO, thus reducing the value of the job.

QoS_level: Users may have different utility functions, i.e., users are willing to pay different amounts for different jobs based on their importance. When a user submits a job, he picks the required QoS for that job based on the user's willingness to pay for the job. The QoS, specified as a percentage of the time the user wants his job to run. For example, if the user specifies a QoS level of 50% and the expected job completion time is $10hours$, then in order to meet the specified QoS, his job must finish within $20hours$ after submission. The higher the QoS percentage, the higher the job's value and the job completion time is closer to the actual time required. If a user wants 95% QoS and specified job completion time as $10hours$, the job must be completed within $10.5hours$. Expressing QoS with percentages gives an upper bound of when a user can expect his job to finish. The idea is similar to that of Least Attained Service (LAS) proposed in [29] in that, if preempted, a job that has received more service is suspended and migrated without violating SLOs.

Jobs can have additional information, e.g., input/output data. The additional meta-data may affect a job's completion time, but we do not account for additional overhead in our environment. For instance, if a job requests a huge input dataset, the data transfer time must reflect the overall execution time.

A resource can have sub-types, i.e., CPUs, GPUs, and memory. We consider two sub-types for this work, and a job could request one or more of these sub-types. The resources are either in the available, occupied or unavailable state. A resource is available if there are no jobs scheduled on it. A resource is occupied if jobs are scheduled on it. A resource is marked as unavailable (to run a job) when power (from renewable energy sources) becomes unavailable.

The available resource pool, R, varies based on the amount of power available to the data center at any given time. We assume that the critical infrastructure (head nodes, network, storage nodes) are always up to facilitate job checkpointing and migration.

3) Training the Scheduler Agent: The environment or the state encoded in matrix (image) format (e.g., available resources; duration of availability of resources; jobs waiting and running). The policy, represented as a neural network, a.k.a policy network, takes a collection of images described above as input and gives a probability distribution over possible actions. Algorithm 1 shows the general Policy Gradient training algorithm used to train the scheduler agent.

Algorithm 1 Policy Gradient training for the Scheduler Agent

```
for iteration = 1, 2, ... do
    Δθ ← 0
    for episode = 1, 2, ..., N do
        Run complete trajectory in the env for T time steps
        Δθ ← Δθ + α∇θ log πθ(s_{i_t}, a_{i_t})
    end for
    θ_new ← θ_old + Δθ #update policy parameter
end for
```

In the current implementation, the actions are: 1) schedule a job, 2) Suspend a job, or 3) Do nothing. The action set is extendable to accommodate other actions in the future. The complete action set for our environment will include, in addition to the above actions, resume a suspended job, freeze a job, unfreeze a frozen job and migrate a job (from one machine or subcluster to another).

During training, the agent perceives a state, takes action, and records the reward information for all time steps of each episode, i.e.,

$$\tau = < s_0, a_o, r_1, s_1, a_1, r_2, s_2, a_2, r_3, ... > \quad (2)$$

The discounted cumulative reward is computed from the previously recorded values for each time step t for every episode. Various job arrival sequences during training (Algorithm 1) ensures the policy network generalizes for future job arrival rates.

4) Reward Shaping: In RL, the *Reward Hypothesis* [16] states that "all of what we mean by goals and purposes" can be described with reward functions. The reward function, a single scalar function, is a combination of the positive reward or associated cost for action in a given state. A reward function is configurable, allowing rewards to be either sparse (rewards only obtained on task completion) or dense (smaller rewards to guide the agent after each step). In our environment, we defined a dense reward function to ensure that jobs make progress at every time step. For instance, the agent gets a small positive reward every time a job, j_i, is scheduled and running. If the agent's action is to suspend a job, it incurs a small negative cost, and resuming a job incurs a small positive reward. All rewards and costs are proportional to the job's value. The final value after each training episode is cumulative of the values of all the jobs that finished:

$$Total\_Value = \sum_{i=1}^{|J\_finished|} j_i.value * scaling\_factor \quad (3)$$

where *scaling_factor* is used to scale down the value to a tractable number.

In the current implementation, the reward function maximizes the cumulative value of the jobs. Ideally, we want the scheduler agent to optimize for multiple objectives, maximizing the total value while minimizing the number of SLO violations, referred to as Multi-Objective Reinforcement Learning (MORL) [28][35]. In MORL, the agent's goal is to learn policies over multiple competing objectives, whose relative importance is unknown to the agent. We are exploring multi-objective reward function implementation in the green data center environment.

C. Resource Pool Management (RPM) based on power availability

In green data centers, it is not only essential to schedule jobs effectively to maximize total value but also schedule jobs such that the power availability is accounted for when scheduling jobs. For instance, the power production for wind

and solar energy varies at different times of the day. Depending on the predicted power, the machines can be in any of the following power states: 1) turned-off (no power production), 2) idle (low power production), or 3) running full throttle (full power production).

Maintaining an optimal resource pool based on power availability, such that cumulative value is maximized, is a separate NP-hard problem. Although the objective function for job scheduling and RPM could be the same (e.g., maximize the total job value), in practice picking optimal job sets to maximize total value is distinct from picking the optimal set of operational machines to maximize total value. Combining the two problems into a single scheduler agent can lead to a state space explosion. The scheduling agent is now responsible for picking a set of jobs from $|N|$ jobs to run and picking a set of $|M|$ machines from the resource pool $|R|$ to transition between power states. Consequently, the scheduler agent's policies must be conditioned on multiple state descriptions resulting in distinct state representations and learning goals.

In our current implementation, we are focusing on job scheduling problem assuming the resources are managed, based on tunable parameters, indicated as the percentage of systems to turn on/off at specific times. The percentage of time a resource is available depends on the predicted power availability. For instance, assume a wind farm generates a power prediction for 24–$hours$ into the future ($12noon$ to $12noon$ next day), and the power prediction indicates "90% power" between $10am$–$12noon$, i.e., 22^{nd} and 24^{th} hour interval. During the 2–$hour$ period of "90% power availability", a percentage of resources proportional to power availability (i.e., 90% of the resources) are marked available, and 10% marked unavailable. The scheduler agent should learn scheduling policies such that jobs either finish by that period or scheduled after resources become available.

In a realistic setting, the wind pattern is not binary (wind or no wind resulting in power or no power mode). The power generation varies throughout the day. We must alter the resource pool such that at any point, the resources available will consume, by switching between the power states, only as much power as generated at that point in time. The control system for selecting a set of resources to shutdown (or resume or idle) based on power availability and running jobs can be treated as a separate agent whose worldview is slightly different from the scheduling agent's environment. The RPM agent's environment includes power prediction from wind farms, machine states (shutdown, idle or throttle), and jobs running on the machines. The agent's goal is to decide a set of machines to alter their power state such that the total value from jobs, currently running, is maximized.

We describe two agents: job scheduling agent and RPM agent, and their respective goals. The two agents work in tandem, i.e., the job scheduler will generate a $job \rightarrow machine\_mapping$ and convey the information to the RPM agent. The RPM agent will map $power\_prediction \rightarrow machine\_states$ and cycle the information back to the job scheduler agent. The two agents' combined actions will generate the system's overall reward, quantifying the executed actions' quality. Since each agent's world view is slightly different, the state becomes Partially Observable MDP (POMDP)[24]. A promising approach that we plan to explore is Multi-View RL [23], a case of Multi-Task RL, described in [23]. This approach allows for decision making when agents share common dynamics but adhere to different observation models.

IV. Results

This section presents the main results obtained by scheduling jobs in a simulated green data center. The green data center environment, as seen by the scheduler agent, is described in §III-B2. We first describe the workload used and evaluate the DRL scheduler agent's performance for varying job arrival rates and power availability. Next, we compare the results of the DRL scheduler agent's performance with heuristics policies. The metrics used for evaluating the performance is the average reward obtained by the scheduler agent and the total value from running the jobs. The training and green data center's environment simulation was performed using PyTorch [32] in the rlpyt framework [33].

1) Workload Model: In this subsection, we aim to answer the following question: How well does the DRL scheduler agent perform with various workloads?. For all the experiments in this section, we used a synthetic workload where each job consists of meta-data described in §III-B2. The online job scheduling works as follows: jobs arrive in a centralized job pool, and the scheduler performs an action – using a scheduling policy – on the jobs present in the job pool.

When a job J_i is selected for execution and if the requested resources ($Resource\_request$ vector) is lower than the total amount of resources available, then requested resources are reserved for this job, and those resources become unavailable. These resources will become available again when $Expected\_runtime$ units of time have passed since the start of the execution of the job J_i or when the job is suspended. If there are not enough resources to process a job J_i, then the scheduler agent continues to process other waiting jobs if available.

We used the A2C policy gradient algorithm with a minibatch-based stochastic gradient update to train the DRL scheduler agent. The policy and value networks are combined into a single Convolutional Neural Network (CNN). The CNN network takes the environment (resources + jobs) encoded as a matrix and outputs action probabilities based on the state information. The agent was trained for $250e4$ steps (Fig.2 through Fig. 4) with $learning\_rate$ of $1e$–3.

In the following subsections, the term "average reward" means the rewards accumulated by the DRL scheduler agent during training, which includes a small positive reward for running the jobs and small negative cost for suspending and delaying jobs. The reward and cost are proportional to the job's value. The reward signal is generated at every time step encouraging or discouraging the agent's actions based on the reward. The term "total value" means the final cumulative

value obtained from running a set of jobs (§III-B4). The total value can be interpreted as the revenue obtained from running a set of jobs.

2) Scheduling with various job arrival rates: In this section, the DRL scheduler agent's training performance is measured as the job arrival rates vary. The goal is to verify that the scheduler agent can handle realistic workloads where jobs arrive at varying intervals. Jobs arrive in an online fashion, which means that the scheduler does not know the job information a priori. The jobs arrive at varying intervals; for instance, 50% means a new job arrives every other time step. The jobs vary in length, resource requirements, and value (depending on the user's QoS level). For our experiments, we generated a synthetic workload where jobs can request one or more resource types (CPU and GPU) with different runtimes, r_t (§III-B2). The DRL scheduler can handle job arrival modeled as a *Poisson* process and real job traces instead of synthetic workload.

We measure both the total value generated by the scheduler agent and the job slowdown as the job arrival rate varies. Job slowdown for a set of jobs defined as:

$$Job\_Slowdown = \frac{1}{|J|}(\sum_{i=1}^{|J|}(j_i.f\_t - j_i.e\_t)/j_i.r\_t) \quad (4)$$

where, f_t is j_i's finish time, e_t is j_i's enter time and r_t is the expected runtime specified by the user.

Fig. 2. shows the average reward ($y–axis$) earned for 20%, 40%, 60%, 80%, and 100% job arrival rates. The $x–axis$ shows the number of training iterations that the agent trained for. The experiments assume no power outage (100% power availability). At job arrival rates of 20% and 40%, the number of jobs arriving is less than the number of available resources. Since the scheduler has less number of jobs to run, the overall reward earned is less. As the job arrival rate increases, the average reward increases since additional jobs get scheduled on available resources. Initially, up to $30K$ steps, the DRL scheduler agent's rewards are closer to zero, indicating that the agent is still learning good policies via trial-and-error. After $50K$ steps, the reward values converge, indicating that that is the best the scheduler agent can do for that workload.

Fig. 3. shows the average slowdown ($y–axis$) for various job arrival rates. The $x–axis$ shows the number of training steps that the DRL scheduler agent is trained for. The job slowdown increases as the number of jobs (based on job arrival rate) increase. For instance, the job slowdown with a 20% job arrival rate is significantly lower than job slowdown with a 100% job arrival rate. We note that initially when the agent is still exploring via trial-and-error, more jobs suspensions causing lower slowdown (also causing low rewards compared to Fig. 2.), but as the scheduler agent learns that suspending the jobs incur an additional cost (cost applied in the reward function), the scheduler agent learns to reduce the number of jobs suspended. Lower job suspension increases the job slowdown because the scheduler agent is not consuming new jobs. After 70K steps, the job slowdown converges, indicating that the agent reached an acceptable policy that maximizes its rewards.

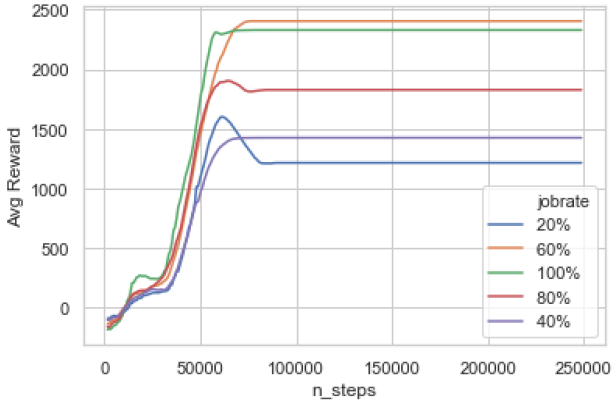

Fig. 2. Average reward for various job arrival rates

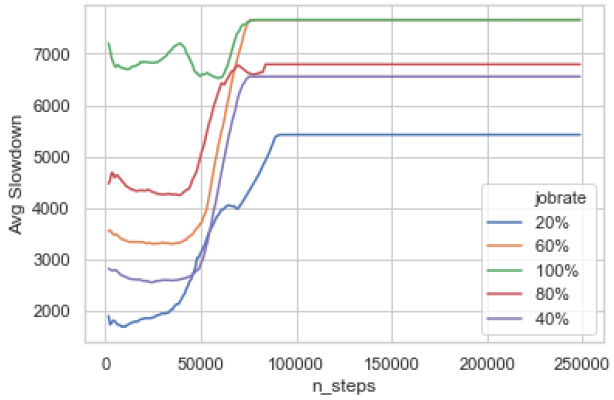

Fig. 3. Average slowdown for various job arrival rates

3) Adaptability to fluctuating power availability: The electricity generated by renewable sources is not always constant, which means some data center resources must switch between power states (off, idle, or full throttle). The scheduler should adapt to the fluctuations in resource availability and schedule the jobs around the highs and lows such that total job value is maximized. In our experiments, the scheduler agent was trained with various power availability levels and measured the average reward and total value from finished jobs. The various power availability levels are 90% (power unavailable 90% of the time, which means only 90% of are available during that period), and 50% (power unavailable 50% of the time). The power availability is controlled via a tunable parameter that dictates the percentage of resources in on/off state; the percentage of available resources proportional to the amount of the power available. The job arrival rate is kept constant at 100% for all the power availability levels measured.

Fig. 4. shows the scheduler agent's average reward ($y–axis$) during training. The $x–axis$ shows the number of training steps that the agent was trained for. Initially, the agent's average reward is meager (below zero) for both the power availability levels. At this point, the scheduler agent is ex-

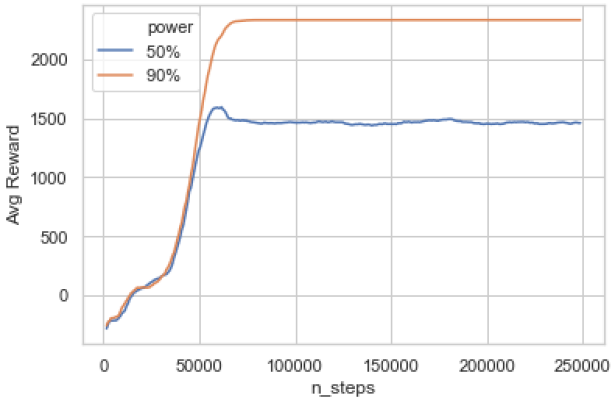

Fig. 4. Average reward for 90% and 50% power availability

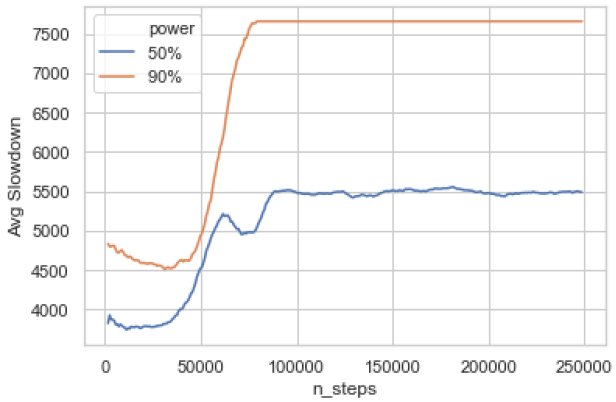

Fig. 5. Average slowdown for 90% and 50% power availability

ploring the various states and actions, possibly generating suboptimal actions, leading to low average reward. Gradually, the agent learns the states and associated policies to follow in order to increase the average reward. From Fig. 4., the average reward during power availability of 90% is much higher than during the power availability of 50% as expected. That said, the agent's average reward is 60%, at power availability of 50%, as that of during power availability of 90%, 10% more considering the agent has 50% fewer resources. This indicates that the agent did indeed learn good scheduling policies to maximize the total job value even when considerably fewer resources are available. In practice, the power availability, with solar and wind combined, is usually much higher than 50%. We used 50% power availability to demonstrate the DRL scheduler agent's adaptability to extreme situations.

Fig. 5. shows the total job slowdown ($y–axis$) during training with 90% and 50% power availability. The $x–axis$ shows the number of training steps that the agent trained for. When the power availability is 90%, the job slowdown is similar to that of power availability at 100% (Fig. 3). That is, even with only 90% of total resources available, the job slowdown is close to 100% of resources availability. When the power availability drops to 50%, naturally, more jobs get delayed because not enough resources are available to run the

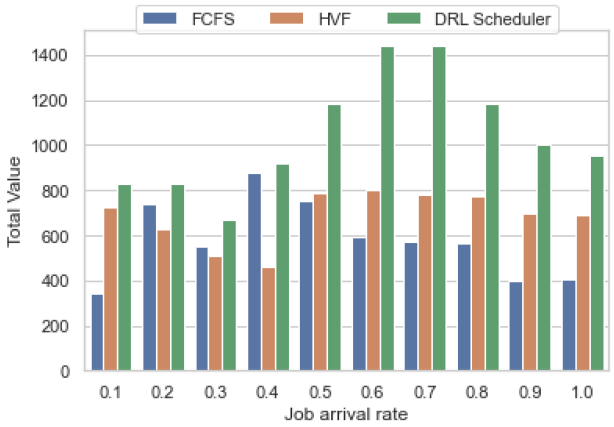

Fig. 6. Total job values, DRL scheduler agent vs. heuristics scheduling policies

jobs.

4) Evaluating DRL scheduler agent and heuristics scheduling: Fig. 6. shows the total value of the DRL scheduler agent compared to the traditional scheduling policies: 1) First Come First Serve (FCFS) and 2) Highest Value First (HVF). The job arrival rates, ranging between 10% and 100%, and the corresponding total job values generated by FCFS, HVF, and DRL scheduler agents are plotted in Fig. 6. The FCFS policy always picks jobs that arrived first to schedule, and HVF picks the highest value job to schedule on the available resources. The FCFS policy serves as a baseline for comparison with greedy HVF and DRL scheduler agent. When the job arrival rates are below 50%, the heuristics policies and DRL scheduler agent perform (in terms of total job value) similarly, except at 40% where HVF performs poorly. For job arrival rates above 50%, the DRL scheduler agent outperforms (between $\sim$ 38% and $\sim$ 70%) the greedy heuristics HVF policy. When there are more jobs than available resources, greedy HVF will not schedule jobs of lesser value until the higher value jobs are scheduled, leading to a reduction in the total value obtained. On the other hand, the DRL scheduler agent's policy schedules jobs such that the reward is maximized irrespective of whether the next job picked is of the highest value or not.

V. Conclusion

Today's data centers operate on electricity generated using non-renewable sources (brown energy). Electricity cost is a significant portion of the operational expense, rendering end-users paying higher prices for the cloud services or reduced service providers' profits. Green data centers can significantly lower the overall operational expenses by employing renewable energy sources. An additional benefit of using renewable energy is a dramatic reduction in climate impact. The most practical way is to colocate the green data centers right in place with or close to the energy generation, eliminating the cost and risks associated with power transmission.

Using renewable energy sources to power the data centers has challenges. For instance, renewable sources' power pro-

duction is not continuous and not uniform across all geographical regions, time of day, or all seasons. The data centers running on renewable energy sources need smart systems and system-software that adapt to the power variability to ensure that cloud services are available during transient power unavailability in these data centers. Three significant issues to address are 1) Meeting Service Level Objectives (SLOs), 2) Managing the resources, and 3) Adapting to power variability.

Hand-engineering domain-specific heuristics-based schedulers to meet specific objective functions is time-consuming, expensive, and needs extensive tuning in this dynamic environment. We applied Deep Reinforcement Learning (DRL) to automatically learn effective job scheduling policies while continuously adapting to the complex dynamic environment. The DRL based scheduler's objective function is to maximize the total value from jobs (maximizing service providers' revenue). Our results demonstrate that the DRL based scheduler generates efficient scheduling policies in a complex dynamic green data center environment adapting to various job arrival rates. The DRL scheduling agent performs 30%–70% better in maximizing total job value than traditional heuristics algorithms. Finally, the DRL scheduler efficiently adapts to the power variability by automatically learning good scheduling policies that maximize jobs' total value.

As future work, we could improve on the current work in two directions. The first would be to modify the DRL scheduler's objective function to satisfy two constraints, i.e., maximize total value from jobs while minimizing SLO violations. Optimizing for multiple constraints can be achieved using the MORL technique described in §III-B4. The second direction is separating the job scheduling and resource pool management functionalities. This can be achieved by applying MVRL techniques (§III-C) where the scheduling agent and RPM agent will have a slightly different world-view, but their combined action will generate the overall reward signal.

ACKNOWLEDGMENT

We thank the MLCS reviewers for their valuable feedback. We thank Dr. Yanjun Qi in the UVA CS department for her constructive feedback on DRL frameworks and Lancium Inc. for financial support. We thank Dr. Alan Sill and Dr. Sandeep Nimmagadda at Texas Tech University for sharing insights on renewable energy management at the GLEAMM data center.

REFERENCES

[1] https://perspectives.mvdirona.com/2008/11/cost-of-power-in-large-scale-data-centers/
[2] https://www.osti.gov/servlets/purl/1372902/
[3] https://www.osti.gov/biblio/1372902/
[4] https://www.sciencedirect.com/science/article/pii/S1876610215009467
[5] https://lancium.com/
[6] https://www.data centerknowledge.com/inside-facebooks-lulea-data-center
[7] https://www.technologyreview.com/s/613779/icelands-data-centers-are-booming-heres-why-thats-a-problem/
[8] https://www.eia.gov/energyexplained/wind/electricity-generation-from-wind.php
[9] https://www.energy.gov/eere/wind/advantages-and-challenges-wind-energy
[10] https://www.energy.gov/eere/wind/wind-vision
[11] https://www.nrel.gov/docs/fy14osti/60983.pdf
[12] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh. Learning scheduling algorithms for data processing clusters. arXiv preprint arXiv:1810.01963, 2018.
[13] Domeniconi, G., Lee, E.K. and Morari, A. CuSH: Cognitive ScHeduler for Heterogeneous High Performance Computing System. 2019. KDD.
[14] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource Management with Deep Reinforcement Learning. 2016. In HotNets. ACM.
[15] T. E. Thomas, J. Koo, S. Chaterji and S. Bagchi. Minerva: A reinforcement learning-based technique for optimal scheduling and bottleneck detection in distributed factory operations. 2018. 10th International Conference on Communication Systems Networks (COMSNETS), Bengaluru, 2018, pp. 129-136.
[16] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. 2nd Edition. 2015.
[17] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). https://doi.org/10.1038/nature16961.
[18] T. E. Thomas, J. Koo, S. Chaterji and S. Bagchi. Minerva: A reinforcement learning-based technique for optimal scheduling and bottleneck detection in distributed factory operations. 2018 10th International Conference on Communication Systems Networks (COMSNETS), Bengaluru, 2018, pp. 129-136.
[19] Gamble and J. Gao. Safety-first AI for autonomous data centre cool- ing and industrial control, 2018. URL safety-first-ai-autonomous-data-centre-cooling-and-industrial-control/.
[20] Zhang, W. Dietterich, T. G. A reinforcement learning approach to job-shop scheduling. In Proc. 14th Int. Jt Conf. Artif. Intell. 1114–1120 (1995).
[21] Y. Bao, Y. Peng and C. Wu. Deep Learning-based Job Placement in Distributed Machine Learning Clusters. IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, Paris, France, 2019, pp. 505-513.
[22] Casavant, T. L. and Kuhl, J. G. 1988. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Trans. Softw. Eng. 14, 2 (Feb.), 141–154.
[23] Minne Li, Lisheng Wu, WANG Jun, and Haitham Bou Ammar. Multi-view reinforcement learning. In Advances in Neural Information Processing Systems, pages 1418–1429, 2019.
[24] Dietterich T. G. An overview of MAXQ hierarchical reinforcement learning. In: Choueiry BY, Walsh T, Lecture notes in computer science, vol. 1864. Springer, Berlin, pp 26–44.
[25] https://lancium.com
[26] https://gleamm.org/
[27] https://static.googleusercontent.com/media/www.google.com/en//green/pdf/achieving-100-renewable-energy-purchasing-goal.pdf
[28] Yang, R., Sun, X., and Narasimhan, K. A Generalized Algorithm for Multi- Objective Reinforcement Learning and Policy Adaptation. In Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 14610– 14621. 2019.
[29] Pamela Delgado, Diego Didona, Florin Dinu, and Willy Zwaenepoel. 2018. Kairos: Preemptive data center Scheduling Without Runtime Estimates. In Proceedings of the ACM Symposium on Cloud Computing, SoCC 2018, Carlsbad, CA, USA.
[30] R. S. Sutton, D. A. McAllester, S. P. Singh and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation", Proc. of NIPS, 2000.
[31] Apache Spark. 2018. Spark: Dynamic Resource Allocation. (2018). http://spark.apache.org/docs/2.2.1/job- scheduling.htmldynamic-resource-allocation Spark v2.2.1 Documentation.
[32] Mnih, V . et al. Asynchronous methods for deep reinforcement learning. In Proc. 33rd Int. Conf. Mach. Learn. Vol. 48 (eds Balcan, M. F. Weinberger, K. Q.) 1928–1937 (2016).
[33] https://pytorch.org
[34] Adam Stooke and Pieter Abbeel. rlpyt: A research code base for deep reinforcement learning in pytorch. arXiv preprint arXiv:1909.01500, 2019.
[35] https://www.nrel.gov/docs/fy12osti/52233.pdf
[36] Adam Stooke, Joshua Achiam, Pieter Abbeel. Responsive Safety in Reinforcement Learning by PID Lagrangian Methods. arXiv:2007.03964. 2020.