Adaptive Control of Data Center Cooling using Deep Reinforcement Learning

Heimerson, Albin; Sjölund, Johannes; Brännvall, Rickard; Gustafsson, Jonas; Eker, Johan

Published in:
2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)

DOI:
10.1109/ACSOSC56246.2022.00018

2022

Adaptive Control of Data Center Cooling using Deep Reinforcement Learning

Albin Heimerson, Johannes Sjölund, Rickard Brännvall, Jonas Gustafsson, Johan Eker

March 21, 2023

Abstract

In this paper, we explore the use of Reinforcement Learning (RL) to improve the control of cooling equipment in Data Centers (DCs). DCs are inherently complex systems, and thus challenging to model from first principles. Machine learning offers a way to address this by instead training a model to capture the thermal dynamics of a DC. In RL, an agent learns to control a system through trial-and-error. However, for systems such as DCs, an interactive trial-and-error approach is not possible, and instead, a high-fidelity model is needed. In this paper, we develop a DC model using Computational Fluid Dynamics (CFD) based on the Lattice Boltzmann Method (LBM) Bhatnagar-Gross-Krook (BGK) algorithm. The model features transient boundary conditions for simulating the DC room, heat-generating servers, and Computer Room Air Handlers (CRAHs) as well as rejection components outside the server room such as heat exchangers, compressors, and dry coolers. This model is used to train a RL agent to control the cooling equipment. Evaluations show that the RL agent can outperform traditional controllers and also can adapt to changes in the environment, such as equipment breaking down.

This work was supported by Vinnova grant ITEA3-17002 (AutoDC).

1 Introduction

According to a 2021 study by Koot and Wijnhoven [1] the total worldwide energy consumption of Data Centers (DCs) is expected to grow from 286 TWh in 2016, meaning around 1.15% of global energy consumption, to about 321 TWh or 1.86% in 2030. To describe how efficiently a DC uses energy, a metric called Power Usage Effectiveness (PUE) is commonly used, which is the ratio of total facility energy usage over Information Technology (IT) equipment energy. In a 2016 study by Ni and Bai [2] where 100 DCs were examined, their PUEs were found to range from 1.33 to 3.85, with an average of 2.13, thus showing an overhead of more than 100%. The same study also found that the air conditioning systems used on average 38% of the total energy, showing that optimization of cooling systems is important for achieving more sustainable DC operations.

The thermodynamic behavior of DC and the impact of cloud services are extremely complex. Often, traditional air conditioning systems are tuned using heuristics that may not capture the full dynamics of the systems. This may result in inefficient operations and suboptimal cooling end energy usage. In this paper, we propose an adaptive air conditioning control strategy for DCs using Reinforcement Learning (RL).

Reinforcement Learning (RL) is an area of machine learning where an agent is interacting with an environment by selecting an action and receiving a reward. The objective is to learn a policy that maximizes the cumulative reward. RL agents learn by doing, and often fail multiple times. Exploration is an important element of training an RL agent. In this process, the agent tries out different actions, many leading up to bad system states. Deploying an RL agent on a real DC is typically not feasible since it could cause real damage. To this end we created a realistic digital twin of the DC based on Computational Fluid Dynamics (CFD) for training of the RL agent. The goal is to create a controller with more thermal awareness, allowing for more energy-efficient operation.

Major challenges in implementing an RL agent for DC control are the large state space created by the number of individual server temperatures, the delayed rewards, the noise from server load balancing, and the changing

outdoor temperatures. The challenges are approached by both allowing more sensing, but then also introducing initial structure and transformations in the networks to reduce the data complexity.

The main contribution of this paper is the design of an RL agent that outperforms current best practices. It also shows the benefit of an approach that can learn over time and be adaptive to changing circumstances. Furthermore, the viability of using complex simulation models for training is explored, although verifying the agent on a real system remains as future work.

The rest of the paper is laid out as follows. Section 2 describes previous work in both DC modeling and controller optimization of DC cooling systems. The simulation model used for training is presented in Section 3. Section 4 gives a brief introduction to RL, the specific algorithm used in this work. Section 5 explains the experimental setup for the model and agents used. Results are presented in Section 6 and compared to two baseline algorithms. In Section 7, we reflect on what benefits this method could bring, as well as possible directions for future work.

2 Related work

2.1 DC modeling

A compact model for the heat flow in a DC with a chilled-water cooling system was proposed by VanGilder et al. in 2018 [3]. It included discretized numerical models for heat sources (servers) and cooling by assuming a simple counter-flow heat-exchanger, which alternatively can be replaced by a quasi-steady-state model for applications when accuracy can be traded for model simplicity.

This compact model is developed further by Healey et al. in 2018 [4] that adds components for the room, plenum, walls, floor, and ceiling to better represent the complete thermal mass of a DC.

Training RL agents on real systems often pose many problems and is commonly done in simulation instead. EnergyPlus [5] is one such simulation tool that can be used to simulate Heating, Ventilation and Air-Conditioning (HVAC) systems in buildings as an aid in training RL agents [6]. It has also been used to simulate DCs for training RL agents to minimize cooling energy by finding optimal facilities setpoints [7].

The proposed CFD method used in this paper allows for fast simulations in an interactive fashion, which is suitable for use in an RL training scenario. The method can also handle transient changes in boundary conditions, which was not possible with most earlier approaches.

This work is similar to the work of Van et al. [8] in that both use complex models to train an RL agent offline (i.e. not on the real process), and the goal is to optimize energy consumption. The setups are slightly different, in [8] a solely free-cooling approach is used, while this paper employs a model which switches between compressor-cooling and free-cooling depending on setpoints. This paper also utilizes RL methods designed to do updates in a more stable and robust way with the intention to continue updating the agent when running on a real process, which should allow for a more adaptive agent.

2.2 DC control

With the increasing number of DCs in the world, the interest in optimizing energy consumption is growing. Cooling systems, though not the largest contributor, use a considerable amount of energy and is one area that has seen a lot of work lately.

It is not uncommon to find DC operators relying upon manually tuning the cooling systems and running a simple feedback loop for the fans to keep the cold aisle temperature at a constant value.

Using Machine Learning (ML) for DC control is not a novel idea, and has been used both as an aid to the human operators or as an independent controller. In work from Google presented in [9] a supervised learning approach is designed to predict the PUE based on multiple different sensors in the DC. This is then used by the operators to predict which configuration would result in the lowest PUE. However, it does not explicitly select and apply actions, as done in this paper.

In recent literature, there have been two main approaches for the cooling optimization problem when it comes to actual control of the DC. These are either using Model Predictive Control (MPC) and data-driven modeling or training an RL agent. In 2017, Lucchese et al. [10] created a CFD based model for which they found the parameters using data-driven methods. In 2018 Lazic et al. [11] estimated a linear model from data with only small amounts of knowledge imposed to create sparsity in the model. Both the above works implement an MPC using their model to control the cooling equipment. The proposed RL based method offers an adaptive approach that will learn and improve over time.

In a 2021 survey by Zhang et al. [12] they compare methods for MPC with RL. The main differences between

these come down to MPC needing a model and not being adaptive by itself, while RL is often model-free, adaptive, but often not as stable and can converge slowly.

In 2020, Van et al. [13] model the gas-vapour mixtures of a DC. This was combined with a Neural Network (NN) model that estimates power consumption and is trained to match real data. The combined model is then used to train an RL agent on a discretized subset of the action space. In previous work by the same authors [8] they use the same model and compare a MPC approach to the training of an RL agent. They conclude that one big disadvantage of the MPC approach is the much higher computational cost for evaluating which action to take.

Townend et al. [14] presented a scheduler that takes multiple levels of infrastructure into account when distributing containers on a Kubernetes cluster. Baek et al. [15] use RL to balance workloads over heterogeneous servers while optimizing throughput and energy usage. This agent work does not explicitly issue actions for the cooling equipment. In [16] we used holistic control strategy which combined cloud workload scheduling with cooling system control to minimize energy usage was presented. The difference compared to the work in this paper is that a much simpler simulation model was used. However, this simple model was also used in this paper to search for suitable hyperparameters for the training of the RL agent.

3 The DC simulation model

The DC CFD model used in this work was first presented by Sjölund [17] and defines three-dimensional boundary conditions for the room, servers, and Computer Room Air Handler (CRAH) units. A conceptual schematic of the heat rejection is shown in Fig. 1 where the different parts are modeled using different strategies. The "IT space" in Fig. 2 is modeled with an CFD method called Lattice Boltzmann Method (LBM), using the Single Relaxation Time (SRT) algorithm [18].

Boundary conditions of the servers and cooling systems are based on mathematical modeling of hardware such as the power used by the IT load (cloud services), the fan speeds, the vapor compression, and the heat exchangers, which all affect the temperatures and air velocities.

3.1 Lattice Boltzmann method

The LBM in its simplest form is based on a uniform grid of statistical distribution functions (DFs), called lattice sites, representing density and velocity of fluid particle groups affected by different forces. Evolution of fluid flow over time is computed using the discrete lattice Boltzmann equation

$$f(\vec{x} + \vec{e}\Delta t, t + \Delta t) = f(\vec{x}, t) + \Gamma(f(\vec{x}, t)) + F\Delta t, \quad (1)$$

where f is the distribution function of density, representing convection and diffusion, $\vec{x}$ is displacement in space, Δt is the time step, Γ is the collision term and F is a body force perturbation.

In the model, the density and velocity directions in each lattice site are discretized in 3D space, see Fig. 3. The collision term Γ in (1) the Bhatnagar-Gross-Krook (BGK) employs a single relaxation time τ to return the perturbed system into equilibrium, by

$$\Gamma(f(\vec{x}, t)) = -\frac{\Delta t}{\tau}\left(f(\vec{x}, t) - f^{eq}(\vec{x}, t)\right). \quad (2)$$

The function f^{eq} is the equilibrium distribution for velocity.

$$\rho(\vec{x}, t) = \sum_{i=0}^{Q} f_i(\vec{x}, t) \text{ and } \vec{u}(\vec{x}, t) = \frac{1}{\rho}\sum_{i=0}^{Q} f_i(\vec{x}, t)\vec{e_i}.$$

Buoyancy effects from natural convection are implemented using the Boussinesq approximation,

$$F_i = \pm\frac{\vec{g}\beta(T - T_0)}{2},$$

where the thermal expansion coefficient β is constant at reference temperature T_0 and $\vec{g}$ is gravitational acceleration. This force works along the directions aligned with gravity ($\vec{e_5}$ and $\vec{e_6}$ in Fig. 3).

For simulating thermal evolution, a separate lattice T_i is used. The velocity lattice affects the temperature lattice through advection, while the temperature affects velocity through buoyancy. Its evolution is described as [18]

$$T_i(\vec{x} + \vec{e}_i\Delta t, t + \Delta t) = T_i(\vec{x}, t) - \frac{\Delta t}{\tau_T}\left(T_i(\vec{x}, t) - T_i^{eq}(\vec{x}, t)\right),$$

where $T_i^{eq}(\vec{x}, t)$ is the equilibrium DF and τ_T the relaxation time.

3.2 Turbulence modeling

Turbulence was modeled by Large Eddy Simulation (LES) and the turbulent eddy viscosity is calculated as

$$\nu_t = \frac{1}{6}\sqrt{\nu_0^2 + 18C_s^2\Delta^2\sqrt{\mathcal{S}_{\alpha\beta}\mathcal{S}_{\alpha\beta}}}$$

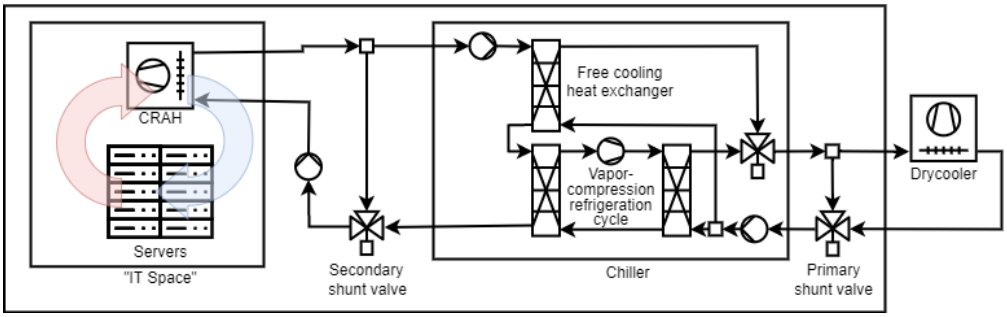

Figure 1: Conceptual heat rejection schematic of the physical model.

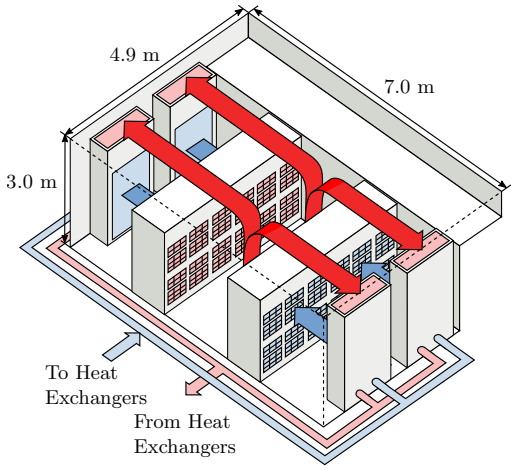

Figure 2: The idealized hot and cold flows in the DC. This corresponds to the "IT space" in Fig 1.

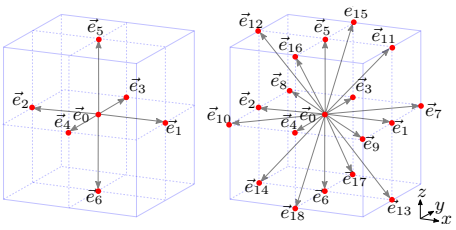

Figure 3: The lattice sites are used to discretize the room in Fig. 2 in order to model flow along the directions $\vec{e}_i$.

where ν_0 is the kinematic viscosity for no turbulence model, $C_s = 0.1$ was chosen as the Smagorinsky constant, filter cutoff length Δ is set to unity and $\mathcal{S}_{\alpha\beta}$ is the local stress tensor [18].

The relaxation time in (2) is then replaced with

$$\tau = 3\nu + 0.5 = 3(\nu_0 + \nu_t) + 0.5.$$

3.3 Boundary conditions

Three different types of boundary conditions define the three-dimensional model. The solid surfaces are defined using a no-slip condition that imposes zero velocity. This is implemented by reverting particles leaving the domain in the opposite directions.

The inlet/outlet conditions, which specify a constant flow velocity $\vec{u}_0$ or temperature T_0 in the direction of the surface normal $\vec{n}$, were set using the corresponding equilibrium DF.

Finally, the airflow into CRAHs and frontal server air intakes were modeled using a zero-gradient boundary condition, where the air was effectively removed from the simulation by setting the velocity and temperature gradients to zero.

A model for the transient effects of a server's thermal mass is implemented based on VanGilder et al. [19]. The effective temperature in the server is

$$T_{eff} = \left(\frac{\tau_1}{\tau_1 + \Delta t}\right) T_{eff}^{old} + \left(\frac{\Delta t}{\tau_1 + \Delta t}\right) \left[T_{amb} + \Delta T_{IT}\right],$$

which along with ambient temperature T_{amb} and IT load

ΔT_{IT} it determines server exhaust temperature.

$$T_{ex} = T_{amb} + \Delta T_{IT} + \left(\frac{\tau_2}{\tau_1 + \Delta t}\right)\left[T_{eff}^{old} - T_{amb} - \Delta T_{IT}\right] \quad (3)$$

Time constants τ_1 and τ_2 are approximated by

$$\tau_1 = \frac{Mc_{p_{eff}}}{hA} \text{ and } \tau_2 = \frac{Mc_{p_{eff}}}{Q_{avg}c_{p_{air}}},$$

where $M = 8$ kg is the server mass, $A = 0.15$ m^2 server internal surface area, $c_{p_{air}} = 1000$ J/(kg·K) specific heat capacity of air at 30°C, while $c_{p_{eff}} = 400$ J/(kg·K) is overall specific heat and $h = 40$ J/(s·m^2·K) overall effective heat transfer coefficient. This resulted in time constants of approximately 10 minutes and $\tau_2 < \tau_1$.

3.4 Server, CRAH, and compressor models

The models for the servers and cooling equipment are based on similar ideas as in our previous work [16].

Individual servers s have IT loads generating heat with an effect of p_s, and a flow Q_s cooling the server. Using the volumetric heat capacity $C_v = 1183$ J/(m^3·K), the IT load referenced in (3) and (4) is

$$\Delta T_{IT_s} = \frac{p_s}{C_v Q_s}.$$

The CPU is the main part that needs to be cooled, and its temperature is modeled to depend on the inlet temperature T_{IN_s} as well as the IT load ΔT_{IT_s}. The new temperature of the CPU is calculated as

$$T_{CPU_s} = T_{IN_s} + R\Delta T_{IT_s} \quad (4)$$

where $R \approx 3$ is describes heat is removal from the CPU and was estimated to make the simulations behave realistically. CPU temperature is used to update the future flow for the using an integral controller

$$Q_s^{targ} = Q_s + K_i(T_{CPU_{SP}} - T_{CPU_s})$$

where Q_s^{targ} is constrained by $0.001 < Q_s^{targ} < 0.04$. The controller was tuned to $K_i \approx -8{\cdot}10^{-5}$ m^3/(s·K) for a reasonably quick response, and $T_{CPU_{SP}} = 60$°C was chosen as a stationary target temperature.

The cooling is modeled as shown in Fig. 1, where a free-cooling heat exchanger is in series with a compressor. If the outside air is cold enough, the dry-cooler is sufficient, and otherwise the compressor is needed. The dry-cooler allows removing heat corresponding to the difference in temperature between the CRAH inlets and the outdoor temperature, and the remaining heat is then removed using the compressor.

All the flows should follow fan affinity laws derived from dimensional analysis, so the power will be cubic with respect to the flow [20]. The flows in the cooling chain are assumed to be proportional to the flow through the individual CRAH units c, which then makes the total power used for flows

$$p_{flow} = K_{srv}\sum_{s=1}^{360} Q_s^3 + K_{cool}\sum_{c=1}^{4} Q_c^3 \quad (5)$$

where $K_{srv} = 7.9 \cdot 10^5$ Ws^3/m^9 is calculated to match the specification on the fans and $K_{cool} \approx 80$ Ws^3/m^9 is estimated to cover all the flows in the cooling chain.

The compressor in the chiller will need to lower the heat from the outdoor temperature T_{AMB} to the setpoint temperature T_{SP}, and will do so with a coefficient of performance $K_C \approx 3$. The total power used in the compressor is then a sum over the power used to reduce each CRAH to the corresponding setpoint temperature,

$$p_{comp} = \frac{C_v}{K_C}\sum_{c=1}^{4} \max(0, Q_c(T_{AMB} - T_{SP_c}). \quad (6)$$

4 Reinforcement Learning

RL is an area of machine learning concerned with how to find optimal ways of interacting with an environment. The RL agent selects what action to take based on the state of the system and observes how the environment changes. It receives a reward signal for each applied action. The goal is for the agent to learn to estimate the value of different states and learn a policy that selects actions that maximize the cumulative reward [21]. RL has shown great promises in very complex scenarios, for example, playing the board game Go [22], which were considered too computationally complex. Two distinguishing features of RL are the trial-and-error search and the delayed reward. Some environments are more suitable for trial-and-error than others. When it is not possible to perform trial-and-error on a real system, like for a data center, a high-fidelity simulation model is needed. The delayed reward comes from the fact the value of an action, for example, the value of placing a stone in Go, is not evident until the game is finished. This means that an

agent has to play multiple games (episodes) to learn the value of each state and action. The delayed reward makes learning complex in RL. When controlling a datacenter, there is no terminal state and no final accumulated reward. We must instead rely on techniques that make no assumptions about episodes.

Today, DC cooling systems are typically controlled using standard algorithms, e.g., PID controllers. A challenge in DC control design is the complexity of the datacenter itself, composed of servers, routers, power distribution units, etc., and the modeling of the thermal environment. Machine learning offers the means to address this by instead training a model. Using RL, it is possible to allow the agent to interact with the environment to learn which actions to take. By giving the controller more information and making it more adaptive, it is possible to find improved control strategies. In addition, an RL agent can learn dynamically while interacting with the environment and can thus be able to adapt to changing conditions, for example as the seasons change or as hardware breaks down.

We will below give a brief background to our RL system, and then later provide details on our implementation in Section 5.

Fig. 4 visualizes the agent's interaction with the environment. The agent observes the state s_t and reward r_t, after which the agent's policy π generates the next action to take, $a_t = \pi(s_t)$. The environment is stepped to forward to $t+1$, resulting in a new observation of s_{t+1} and r_{t+1}. The state s_t corresponds to temperature readings and the actions a_t control the cooling equipment, i.e., the CRAH units.

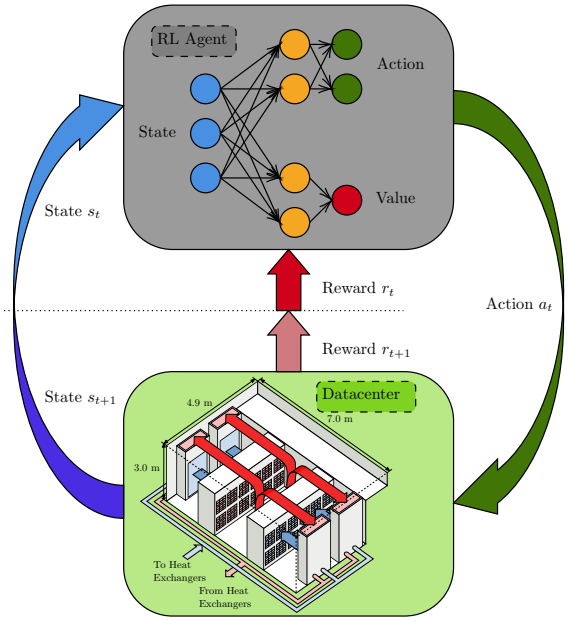

Figure 4: The RL agent observes the state and receives a reward for time t, after which it decides on an action. In the next step, it receives the new observation as well as a reward describing the value of the new state of the DC.

4.1 Reinforcement Learning Algorithm

Actor-Critics are a family of common algorithms in RL where the agent consists of two parts, the *actor* that is used to select an action and the *critic* that evaluates the current state based on the expected reward. The actor has a policy that generates an action distribution based on the current state, from which the action is sampled $a_t \sim \pi(\cdot \mid s_t)$. The critic estimates the value function $V(s_t) = \mathbb{E}_\pi \left[\sum_i \gamma^i r_{t+i}\right]$, which is the discounted future value from a state, where $\gamma \in [0, 1]$ is a discount factor placing less weight on future rewards since they are probably not as reliable [21].

The Bellman equation is a recurrent relationship over the value function, and this can be used for training. The goal is that $V_\phi(s_t) = r_{t+1} + \gamma V_\phi(s_{t+1})$ should hold, where V_ϕ is the value function approximation parametrized by ϕ. To achieve this the loss is defined as the mean squared error of the difference between the two sides, also known as the mean squared bellman error [21], which is approximated by

$$L_V(\phi) = \frac{1}{N} \sum_{i=1}^{N} (r_{t_i+1} + \gamma V_\phi(s_{t_i+1}) - V_\phi(s_{t_i}))^2. \quad (7)$$

The policy is commonly trained to increase the probability of advantageous actions, but how this is achieved varies a bit between algorithms. One way is to take the gradient of $\pi_\theta(a_t \mid s_t)$ with respect to θ, a parametrization of the policy function. The gradient is then weighted based on both the probability of taking the action $\pi_\theta(a_t \mid s_t)$, and the advantage $\hat{A}_t$ of the action [21]. The advantage specifies how much better than expected an action turned out to be, and can be estimated with $\hat{A}_t = V_\phi(s_{t+1}) + r_{t+1} - V_\phi(s_t)$. Updating the policy using this method will increase the probability of actions that are estimated to be better than expected $\hat{A}_t > 0$, while decreasing the probability when $\hat{A}_t < 0$.

This work employs Proximal Policy Optimization (PPO) [23], a type of actor-critic algorithm designed to

constrain the policy updates in an effort to make learning safer. It is implemented using constraints on $q_t(\theta)$, the ratio of the action probabilities between the updated policy and the old policy.

$$q_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

The objective of the policy is to increase the probability of taking good actions while decreasing the probability of bad ones. Doing this without constraints would look like

$$L_1(\theta) = q_t(\theta)\hat{A}_t.$$

To also implement constraints on the policy, a second objective is defined to enforce $1 - \epsilon \leq q_t(\theta) \leq 1 + \epsilon$ for some ϵ.

$$L_2(\theta) = \max(1 - \epsilon, \min(q_t(\theta), 1 + \epsilon))\hat{A}_t \tag{8}$$

The actual objective for PPO is the minimum of these two,

$$L^{PPO}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(L_1(\theta), L_2(\theta)\right)\right], \tag{9}$$

which should then stop gradients w.r.t to θ if $q_t(\theta)$ is too far from one, i.e. the policy changed too much.

The advantage $\hat{A}_t$ is estimated using Generalized Advantage Estimation (GAE) [24] which utilizes $V(s_t)$ to find a weighted sum of advantages over different horizons.

The end goal is to find a policy that maximizes the probability of taking actions with high advantage.The policy should be improved with each update, but it should not change too much before the updated policy has been tested in the environment.

A good change in the policy would be one where either the action is better than expected $\hat{A}_t > 0$ and the chance of taking this action was increased $q_t(\theta) > 1$, or the action is worse than expected $\hat{A}_t < 0$ and the chance of taking it was decreased $q_t(\theta) < 1$. If the policy was changed in such a way for a batch of data, and $q_t(\theta)$ differs from 1 by more than ϵ, then $L_2(\theta) < L_1(\theta)$ and the objective becomes $(1 \pm \epsilon)\hat{A}_t$ for that batch. The gradient of this objective is zero, so this batch will not affect the policy update. If, on the other hand, the policy has regressed in performance, then $L_1(\theta) \leq L_2(\theta)$ and the full gradient step will always be taken. This results in a larger effort towards fixing the policy for data that was made worse compared to the effort for further improving the policy for data that is already improved.

The value function used to estimate $\hat{A}_t$ is trained using a loss similar to the one in (7).

Figure 5: The experimental pod at RISE that was modeled in the simulations.

5 Evaluation

In this section, the proposed approach is evaluated using simulations of a DC pod. The modeled DC pod contained 360 servers distributed inside twelve racks placed in a hot aisle configuration depicted in Fig. 2. The pod has two pairs of CRAHs on opposite sides of the room, supplying the servers with cold air. The actual experimental DC pod is found at the RISE ICE facility in Luleå, Sweden, and is shown in Fig. 5. All the code, except for the CFD simulations, are publicly available and can be found at Github [25].

The actual training was done using a set of Nvidia V100 GPU clusters running Ray [26]. The GPUs were used to speed up both the training of the RL agents as well as the CFD simulations. Ray has excellent support for large numbers of parallel jobs on distributed platforms, as well as managing resources such as Graphical Processing Units (GPUs).

5.1 The simulation setup

The simulations are configured to trigger interesting scenarios. The outdoor temperatures and workloads are chosen to put the DC in a state where there are interesting decisions to make. This happens when the outdoor temperature is high enough that the compressor is often needed, but the workload is not high enough to require constant max cooling.

Fig. 6 has a linear interpolation over an hourly average of the temperature in Luleå, Sweden, retrieved from the local weather forecast SMHI [27] and is the temperature used for the simulation.

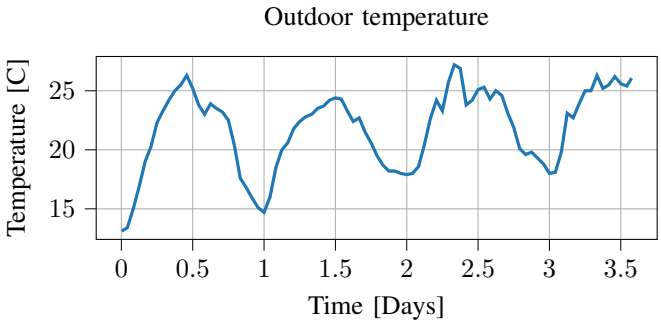

Figure 6: Temperature data for Luleå from SMHI [27] used in simulation.

The workload model is designed to represent different kinds of services found in a datacenter. This includes different kinds of cloud services, e.g., IaaS with virtual machines or containers being allocated or more short running jobs in the shape of FaaS or serverless, but also more traditional HPC style jobs. The workload model consists of a job type with an expected duration, which can be either static or given as a statistical distribution. Each job is associated with a given workload, expressed in terms of power consumption. The energy requirement of each job is static in the evaluation, but this is trivial to extend. In the simulation, each job has a constant load of 20 W and a duration of 540 seconds, and for each step of the simulation, there is a 50% probability of a job arriving.

Jobs are modeled to be received by a load balancer that distributes them over servers by the current lowest load. This will give a reasonably even distribution over the servers. This simple workload model with constant load and duration is motivated by job scheduling not being part of the algorithm at the moment.

The specific values for the jobs in the simulation were chosen to keep the probability and load large enough for changes to happen and matter, while also having an average load that put the DC in a state where there were interesting decisions to make. This choice was made to help the agent learn faster by exciting the state of the environment.

The state consists of room measurements of the outlet temperatures T_{OUT_s} and loads p_s over the 360 servers, as well as the outdoor temperature T_{AMB}.

$$s_t = [T_{OUT_1}, \ldots, T_{OUT_{360}}, p_1, \ldots, p_{360}, T_{AMB}] \tag{10}$$

The actions are continuous flow and temperature set-points for each of the 4 CRAH.

$$a_t = [Q_1, \ldots, Q_4, T_{SP_1}, \ldots, T_{SP_4}] \tag{11}$$

The reward from the environment is defined as a weighted sum over all the negative costs in the environment. The costs are the cooling energy and an artificial cost based on a common standard to keep cold isles below 27°C.

$$r_t = -C_1 \underbrace{(p_{flow} + p_{comp})\Delta t}_{\text{Cooling energy}} - C_2 \underbrace{\sum_s^N \max(0, T_{IN_s} - 27)}_{\text{Cold aisle threshold}} \tag{12}$$

The effects p_{flow} (5) and p_{comp} (6) gives the energy, where $\Delta t = 1$ since the environment is simulated in steps of one second. The weights $C_1 = 10^{-1}$ J^{-1} and $C_2 = 1$ K^{-1} were calibrated so that not staying under the threshold for the cold aisle is punished quite hard. This will help the agent to learn this objective first, and once this objective is (mostly) fulfilled it will have very little effect on the final reward, so the agent can focus on the energy expenditure. These weights are also used to scale the reward to "reasonable" values, a common trick that can have a large effect on training.

The simulation model is quite resource intensive, and it is here the GPU requirement comes in. Having a cluster to run multiple environments in parallel to collect data for the agent made the training run much faster.

5.2 Baseline agents

As mentioned previously, it is not uncommon to find very simple strategies that only aim to keep a constant cold aisle temperature without taking any other external factors into account.

This strategy was used as a baseline to compare against, with two slightly different versions of it to capture the different objectives. Both keep a constant flow and temperature coming out of the CRAH with the single difference that the temperature set-point is either 18°C (B18) or 22°C (B22).

The baseline with 18°C set-point is very good at keeping it cold, and will never break the cold-aisle threshold, but it will run the compressor more often.

The baseline with 22°C set-point is much more energy-efficient, but will sometimes fail to adhere to the cold aisle threshold.

Neither of these is likely an optimal constant valued strategy, but they are each very good at one of the two

Table 1: Parameters for PPO training and model, see Fig. 7 for model layout.

Name	Value
Adam stepsize	$5 \cdot 10^{-5}$
Train batch size	1600
Minibatch size	128
Number of epochs	30
Discount (γ)	0.99
GAE parameter	1.0
Clipping parameter	0.3
VF clipping	1000.0
KL target	0.01
Hidden layer size	64
Activation function	elu

objectives while still being very similar strategies. Using both as a benchmark for the RL agent should then give a good comparison of how well it fares in each objective.

5.3 Reinforcement Learning agent

The agent is an implementation of PPO [23] from Ray's RLlib [28]. PPO was chosen partly for its stable policy updates, something that is desirable for an adaptive algorithm that should be trained and updated while running on a real system. PPO is an on-policy algorithm that commonly is trained by having it interact with multiple parallel environments to collect more and varied data. We use eight parallel environments running with slightly different initial conditions.

Successfully training RL agents can require a bit of trial-and-error. It can for example be beneficial to start in a simpler environment to get early signs of if it might work, and is one reason behind the simple environment in [16].

Doing hyperparameter tuning is also important, and the parameters presented in Table 1 are a mix of RLlib defaults, things tuned on the simpler environment (since the simulation was a few orders of magnitude faster) and a few that were tuned in the environment presented here.

All states and actions were also normalized to around the interval $[-1, 1]$, common practice in many ML applications. This can help with gradient updates by making the gradients more similar, and thus the steps will not be dominated by one direction.

The model used for π_θ and V_ϕ is presented in Fig. 7 and is made up of policy and value networks, both using the input defined in (10).

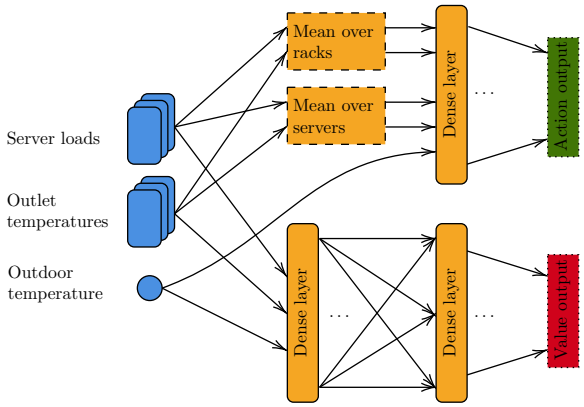

Figure 7: The network structure used for the agent. The same input is used in both networks, see (10). The policy has two layers, the first one with transformations to reduce the space and then a single dense layer before the output layer. The outputs of the policy network are means and standard deviations for normal distributions for each action, which are then used for sampling. The value network has two dense layers and outputs the estimated future reward.

The policy network takes the state and generates an action $a_t = \pi(s_t)$. This network has transformations in the initial layer creating averages over the server states, both over all servers and over servers within each rack. The averages are then fed through a single dense layer of 64 units with `elu` activation. The output layer has two values for each output dimension, this is since the policy is stochastic internally and as such needs a mean and standard deviation to generate each output distribution. The action is then sampled from the resulting distribution.

The value network takes the full state through two dense layers of 64 units, each with `elu` activation functions. The condensed state is not used mainly based on hyperparameter tuning, though it also seems reasonable that the value network might need more precise knowledge over individual servers since, for example, the cubic cost from (5) will allow the distribution of a load within a rack to have an impact on the power used for the server fans which affects the reward.

The RL agent starts interacting with the environment without any previous training and continues to learn throughout the simulation.

Entropy regularization is not used, so the agent will explore a lot in the beginning, since the policy is stochastic by design, but later converge towards something more

Figure 8: The reward is a combination of the cooling energy cost and the cold aisle loss, and is the optimization target for the RL agent.

deterministic where the collected knowledge can be exploited.

6 Results

In Fig. 8 the reward for all three algorithms are compared, and the RL agent generally achieves the highest reward. When the outdoor temperature in Fig. 6 is below 18°C, and hence allows for running the CRAH at minimum temperature with no compressor, the 18°C baseline algorithm is more efficient. After more training, the RL agent might also learn to reduce the CRAH temperature to 18°C in these cases, but it is not unexpected that it is still a bit careful since lower temperatures can incur a large energy cost from the compressor.

The total reward is based on both the cooling power and the cold aisle loss. The RL agent matches up quite well with the 22°C baseline while outperforming the 18°C baseline when it comes to energy utilization, see Fig. 9. But looking at the cold aisle loss in Fig. 10, it is apparent that the 22°C baseline is not managing to keep the cold aisle below the threshold for all servers as well as the other two algorithms.

One thing to note is that the peaks in cooling power from Fig. 9 happen in conjunction with the peaks in temperature, Fig. 6. These peaks are from the compressor needing to run more when the outdoor temperature is high relative to the CRAH setpoint, thus adding a substantial energy cost.

In Fig. 11 the inlet temperatures for all 360 servers are plotted for the three algorithms. Here we see how the RL agent tries to optimize against the boundary that was set,

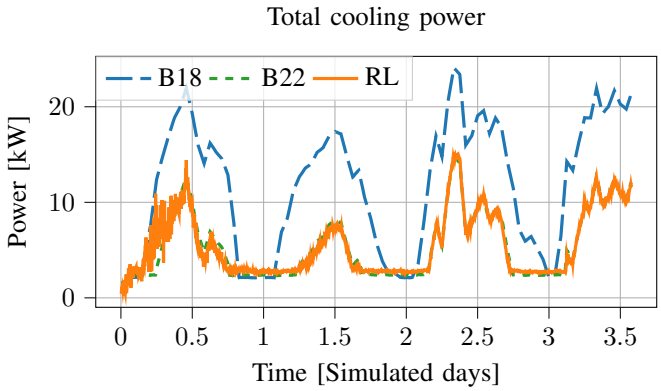

Figure 9: The total cooling power is the combination of the power consumption of all flows (5) and the power consumption of the compressor (6). The RL agent is similar to the 22°C baseline, while managing to keep the cold aisle below the threshold much better in Fig. 10.

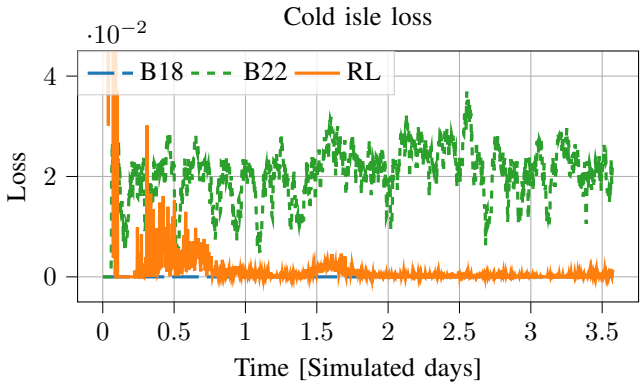

Figure 10: The cold aisle loss for each agent as defined in (12), the 22°C baseline does not manage to keep the server inlets below the threshold.

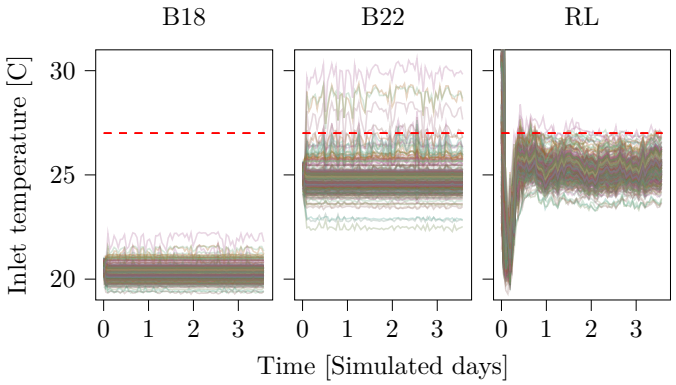

Figure 11: Temperature distribution of server inlets over the three algorithms. The red line is the 27°C threshold used in the loss calculations.

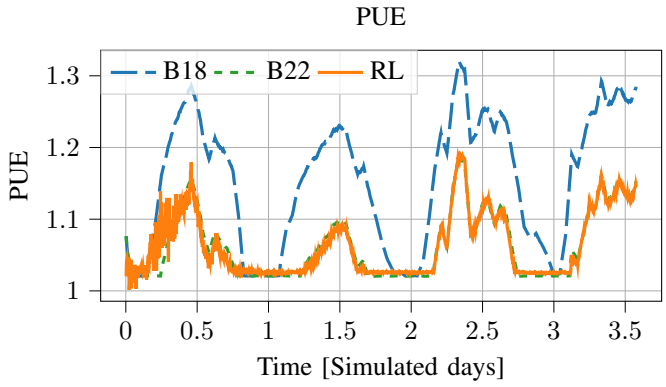

Figure 12: PUE for the RL agent and the two baselines. The RL agent is similar to the 22°C baseline, while managing to keep the cold aisle below the threshold much better in Fig. 10.

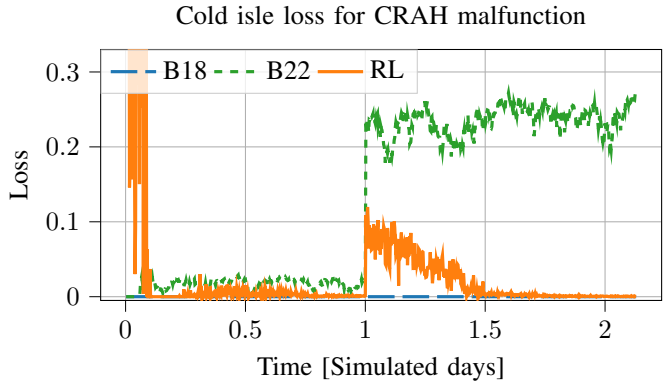

Figure 13: CRAH_0 loses efficiency after one day and the RL agent adapts to the changing conditions.

and while the 22°C baseline sits in a very similar space it has a few servers that violate the threshold by a couple of degrees. These few warmer servers turn out to be in the upper parts of the rack, which seems reasonable, since if there is any re-circulation of hot air from the server it will come from above in this setup. The RL agent manages to balance the CRAH setpoints better to avoid these outliers and make the server inlets stay under the threshold.

The PUE is plotted in Fig. 12 and given that we are running with a constant arrival rate for the workload, it is no surprise that it will have a similar shape to the cooling power in Fig. 9. Again, it is clear that the 18°C baseline is using most power while the other two have similar efficiency, but the RL agent manages to do so while also keeping to the cold aisle thresholds much better than the 22°C baseline.

In addition to running the DC under normal conditions, experiments were also conducted to show how the agent continuously adapts to disturbances. In this case, an inefficiency was introduced in CRAH_0, resulting in the CRAH operating with only 80% of the original airflow while using the same power. Fig. 13 shows how the cold aisle threshold is temporarily broken by the RL agent when the inefficiency is introduced at day one, and how it manages to come back to a similar state as before within half a day. How fast this is can be tuned with learning rates for the agent, and here we went for a rather slow and careful approach to ensure the agent didn't change too quickly with the new environment. Fig. 14 shows how CRAH_0 loses efficiency, and how the flow setpoint is in-

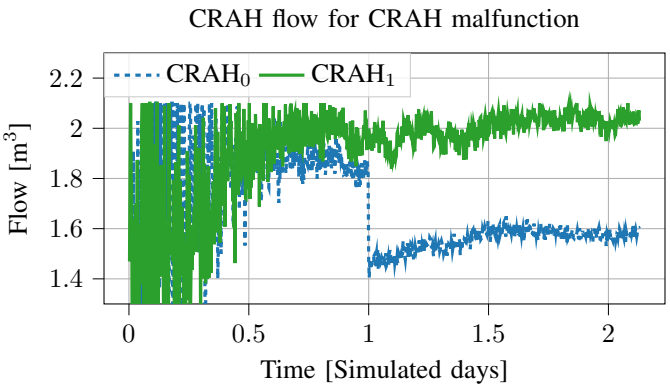

Figure 14: $CRAH_0$ loses efficiency after one day, and the RL agent increases the flow of both $CRAH_0$ and $CRAH_1$ to make up for parts of the lost efficiency. This is without the agent being told that those CRAHs share an aisle.

creased to make up for parts of the lost flow. There is also the neighboring $CRAH_1$, sharing the same cold aisle, which also increases its flow a bit to support the slightly broken $CRAH_0$. This is done without the RL agent getting any information that these CRAH units are in the same aisle and can support each other, and can be done thanks to adaptive controllers that are aware of more of the surrounding state than traditional controllers are.

7 Conclusion

This work shows that an RL agent can learn well in the complex environment of a DC and illustrates how complex models could be used for initial training to not put a DC under strain in the initial phase. It also shows that the RL agent can adapt to problematic situations that occur over time.

The agent is also interacting with the environment every second, this is a high rate compared to many other works in similar fields where the sampling rate can be anywhere from tens of seconds to tens of minutes. One could argue that updating the CRAH every second is certainly not needed, but the problem with slower sampling is that then something might happen right after taking an action and the algorithm will not react to that for a full step. So it is advantageous to have a faster interaction to allow the agent to react faster, even if most of the interactions will just repeat the last action. This is also a reason to use RL agents compared to MPC, since running an MPC on a model of this complexity every second would likely not be a very viable solution.

The weighting between the two objectives, cooling energy and cold aisle loss, could be changed and would have changed the final reward. What is shown is that based on the current weighting, the RL agent finds a policy that is more optimal than the baseline algorithms, one of which is perfect in cold aisle temperature and another which is very good with energy consumption.

7.1 Future work

The next step is to extend the agent to also handle the load-balancing to further improve the energy efficiency. This approach was demonstrated for a smaller DC model in [16].

Validating the agent on a real DC after pre-training on a simulated model of said DC would be the next step. The model used in the simulation is of a real module located at RISE in Luleå. This module is equipped with sensors to allow detailed monitoring of temperatures.

References

[1] Martijn Koot and Fons Wijnhoven. Usage impact on data center electricity needs: A system dynamic forecasting model. *Applied Energy*, 291:116798, 2021.

[2] Jiacheng Ni and Xuelian Bai. A review of air conditioning energy performance in data centers. *Renewable and Sustainable Energy Reviews*, 67:625–640, 2017.

[3] James W. VanGilder, Christopher M. Healey, Michael Condor, Wei Tian, and Quentin Menusier. A compact cooling-system model for transient data center simulations. In *2018 17th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*. IEEE, may 2018.

[4] C.M. Healey, J.W. VanGilder, M. Condor, and W. Tian. Transient data center temperatures after a primary power outage. pages 865–870, 2018.

[5] Drury B. Crawley, Linda K. Lawrie, Frederick C. Winkelmann, W. F. Buhl, Y. Joe Huang, Curtis O. Pedersen, Richard K. Strand, Richard J. Liesen, Daniel E. Fisher, Michael J. Witte, and Jason Glazer. EnergyPlus: Creating a new-generation building energy simulation program. *Energy and Buildings*, 33(4):319–331, April 2001.

[6] Bocheng Li and Li Xia. A multi-grid reinforcement learning method for energy conservation and comfort of HVAC in buildings. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 444–449, August 2015.

[7] Yuanlong Li, Yonggang Wen, Dacheng Tao, and Kyle Guan. Transforming Cooling Optimization for Green Data Center via Deep Reinforcement Learning. *IEEE Transactions on Cybernetics*, 50(5):2002–2013, May 2020.

[8] Duc Van Le, Yingbo Liu, Rongrong Wang, Rui Tan, Yew-Wah Wong, and Yonggang Wen. Control of Air Free-Cooled Data Centers in Tropics via Deep Reinforcement Learning. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, BuildSys '19, pages 306–315, New York, NY, USA, November 2019. Association for Computing Machinery.

[9] Jim Gao. Machine learning applications for data center optimization, 2014.

[10] Riccardo Lucchese, Jesper Olsson, Anna-Lena Ljung, Winston Garcia-Gabin, and Damiano Varagnolo. Energy savings in data centers: A framework for modelling and control of servers' cooling. *IFAC-PapersOnLine*, 50(1):9050–9057, 2017.

[11] Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. Data center cooling using model-predictive control. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[12] Qingxia Zhang, Zihao Meng, Xianwen Hong, Yuhao Zhan, Jia Liu, Jiabao Dong, Tian Bai, Junyu Niu, and M. Jamal Deen. A survey on data center cooling systems: Technology, power consumption modeling and control strategy optimization. *Journal of Systems Architecture*, 119:102253, October 2021.

[13] Duc Van Le, Rongrong Wang, Yingbo Liu, Rui Tan, Yew-Wah Wong, and Yonggang Wen. Deep Reinforcement Learning for Tropical Air Free-Cooled Data Center Control. *arXiv:2012.06834 [cs, eess]*, December 2020.

[14] P. Townend, S. Clement, D. Burdett, R. Yang, J. Shaw, B. Slater, and J. Xu. Invited paper: Improving data center efficiency through holistic scheduling in kubernetes. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 156–15610, 2019.

[15] J. Baek, G. Kaddoum, S. Garg, K. Kaur, and V. Gravel. Managing Fog Networks using Reinforcement Learning Based Load Balancing Algorithm. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–7, April 2019.

[16] Albin Heimerson, Rickard Brännvall, Johannes Sjölund, Johan Eker, and Jonas Gustafsson. Towards a Holistic Controller: Reinforcement Learning for Data Center Control. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, 2021.

[17] Johannes Sjölund. Real-time thermal flow predictions for data centers: Using the lattice boltzmann method on graphics processing units for predicting thermal flow in data centers, 2018.

[18] Nicolas Delbosc. Real-time simulation of indoor air flow using the lattice boltzmann method on graphics processing unit. University of Leeds, September 2015.

[19] James Vangilder, Christopher Healey, Zachary Pardey, and X. Zhang. A compact server model for transient data center simulations. *ASHRAE Transactions*, 119:358–370, 01 2013.

[20] E. Buckingham. On physically similar systems' illustrations of the use of dimensional equations. *Physical Review*, pages 345–376, 1914.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[22] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

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

[24] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv:1506.02438 [cs]*, October 2018.

[25] Albin Heimerson. Rldc rafsine, December 2021. `https://github.com/albheim/rldc_rafsine/tree/ccgrid22`.

[26] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A Distributed Framework for Emerging AI Applications. *arXiv:1712.05889 [cs, stat]*, 2018.

[27] Smhi (swedish meteorological and hydrological institute). `https://www.smhi.se/data/meteorologi/ladda-ner-meteorologiska-observationer#param=airtemperatureInstant,stations=all,stationid=162870`. Accessed: 2021-09-28.

[28] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for Distributed Reinforcement Learning. *arXiv:1712.09381 [cs]*, 2018.