A Reinforcement Learning Approach for Performance-aware Reduction in Power Consumption of Data Center Compute Nodes

Akhilesh Raj
Department of Electrical Engineering
Vanderbilt University
Nashville, Tennessee, USA
akhilesh.raj@vanderbilt.edu

Swann Perarnau
Mathematics and Computer Science Division
Argonne National Laboratory
Lemont, Illinois, USA
swann@anl.gov

Aniruddha Gokhale
Department of Computer Science
Vanderbilt University
Nashville, Tennessee, USA
a.gokhale@vanderbilt.edu

***Abstract*—As Exascale computing becomes a reality, the energy needs of compute nodes in cloud data centers will continue to grow. A common approach to reducing this energy demand is to limit the power consumption of hardware components when workloads are experiencing bottlenecks elsewhere in the system. However, designing a resource controller capable of detecting and limiting power consumption on-the-fly is a complex issue and can also adversely impact application performance. In this paper, we explore the use of Reinforcement Learning (RL) to design a power capping policy on cloud compute nodes using observations on current power consumption and instantaneous application performance (heartbeats). By leveraging the Argo Node Resource Management (NRM) software stack in conjunction with the Intel Running Average Power Limit (RAPL) hardware control mechanism, we design an agent to control the maximum supplied power to processors without compromising on application performance. Employing a Proximal Policy Optimization (PPO) agent to learn an optimal policy on a mathematical model of the compute nodes, we demonstrate and evaluate using the STREAM benchmark how a trained agent running on actual hardware can take actions by balancing power consumption and application performance.**

***Index Terms*—HPC, Power Management, Energy issues in data centers, Reinforcement Learning, RAPL.**

I. Introduction

Cloud data centers are increasingly providing a variety of high performance computing nodes including hardware accelerators to support the needs of a range of compute-intensive applications including various machine learning training/inferencing and scientific applications. In these setups, a large number of compute nodes are networked together via high-speed networks to provide high performance computing capabilities. This growth, however, has given rise to several challenges, notably in the areas of energy efficiency. For instance, Darrow et al. in [1] pointed out that data centers consume 10-50 times the energy per floor space compared to any other commercial building and the consumed energy constitute 2% of the total US energy consumption. Multiple surveys and studies [1]–[5] have shown that the global power consumption for computing is expected to increase in the upcoming years.

Consequently, research on data center power consumption has gained considerable attention since the publication of initial statistics on its energy usage, which illuminated the rapidly increasing demands of servers worldwide [2]. A more recent survey shows that the growing demands of these data centers are not in proportion with their efficiency despite the technological innovations [3]. In other words, the rate at which the power consumption increases across the generations of data centers are higher than the rate of growth in their efficiency. Within the data center, almost 86% of the total power consumption is equally shared between the cooling system and servers [6], which makes the power consumption of compute nodes an important issue that needs to be addressed.

In the context of reducing compute node power consumption, various methods have been proposed that can be broadly classified into two categories: those that focus on minimizing power dissipation and those that focus on minimizing power supply. These approaches aim to reduce the overall energy consumption of compute nodes, while maintaining their performance and functionality. The former is strictly a Very Large Scale Integration (VLSI) design problem, while the latter is a cyber-physical systems problem, which can be dealt with using efficient power management algorithms (i.e., control system design), which is the focus of this paper.

Specifically, we design a method to reduce the consumed power without any noticeable impact on the execution time of a workload. Through our work, we achieve regulation of the average supplied power using a reinforcement learning (RL) agent, trained using a mathematical model which relates the progress made by the application towards the completion of its scientific-goal to the power cap (pcap) of RAPL actuators. Additionally, the goal is to not let the execution time vary significantly when compared to that with maximum power execution.

To that end, we present a real-time, dynamic power management scheme that is based on observing both time-aware and power-aware variables as a solution to the power management problem of the high performance compute nodes. In our work, the RL agent observes the progress made by the application towards completing its scientifc goal at each instant and makes a decision on the power requirement for the next time step based on this observation. The action, which is based on this decision, is taken by setting the running average power limit (RAPL) power caps at the computed value. RAPL is an interface available on modern Intel processors to monitor and control the energy consumption of various power domains. Our approach is incorporated in the Argo Node Resource Management (NRM) stack [7].

The key contributions of this paper are the following:

- A novel RL-based algorithm that relies on the system model for training.
- An RL-based architecture employing the trained model to control the compute node within a desired operational region, while utilizing real-time progress measurements.
- An implementation of the algorithm with an open-source repository for researchers to test and extend the results.

The remainder of this paper is structured as follows: In Section II, we review existing algorithms in power and performance optimization. Section III presents the necessary background for this paper, including the RL algorithm, hardware and software stacks, and packages used in the experiments. Section IV outlines the proposed algorithm and workflow in the context of compute nodes. In Section V, we present the experimental design and implementation, along with the results and analysis. Finally, Section VI provides conclusions including a brief summary, the impact of this work and future opportunities.

II. Related Work

Prior methods for regulating the performance on HPC nodes can be classified either as a Power-aware scheme [8], [9] or as a Time-aware scheme [10], [11]. A power-aware scheme is designed with the primary objective of minimizing power consumption while still achieving acceptable performance levels. The main focus of this scheme is to optimize energy usage and reduce power consumption in computing systems. Power-aware schemes may employ techniques such as dynamic voltage and frequency scaling (DVFS), power capping, power gating, and other power management methods to control and regulate power usage based on workload and system conditions. The goal is to maximize energy efficiency and reduce the overall carbon footprint of the computing system. Time-Aware Scheme on the other hand, prioritizes meeting specific time or performance requirements, often associated with critical real-time or time-sensitive applications. The main objective of a time-aware scheme is to ensure that tasks or processes are completed within predefined time constraints or deadlines. In time-aware schemes, performance and response time take precedence over power efficiency. These schemes may use techniques like aggressive clock frequency scaling, parallelism, or task prioritization to meet the time requirements of critical applications. This section provides an overview of some of these methods which also explains why there is a growing demand for machine learning based methods for the power and performance control that has lead to the proposed work.

Our prior work [12] researched software approaches for power optimization using duty cycle, voltage and frequency scaling, wherein we used machine learning techniques to model the application performance and hardware characteristics, and use these models to guide the placement decisions. But, the choice of method highly depended on the hardware under consideration. Similarly, Jung et al. [13] proposes a supervised learning based power management for multi-core processors. However, such algorithms are not always reliable due to the need for training data prior to the implementation of software-based optimization.

On the other hand, methods that deal with system architecture that set the input power based on the system requirements converges faster. For example, a RAPL-based scheme supports a variety of algorithms that relies on hardware-based control for power capping on Intel processors [14]. But the automation/feedback required for the processor to determine the power requirements based on the application it runs is not detailed in the paper. Similarly Raghavendra et al. [15] utilizes a controller to stop the supplied power that is being delivered to the subsystems, by detecting the no-load zones in the application during its execution. Utilizing the technique, they are able to regulate the power to finer granularities. But, the efficiency of detecting the workload characteristics is not at the best, since the research focuses only on the hardware centric approach which is the main contribution of the paper.

Zhang et al. [16] later came up with a hybrid method, combining software and hardware based methods to control the power supply, which showed significant improvement over the other methods. The authors combined the efficiency of the software-based approaches like DVFS, Task Scheduling etc. and the timeliness of the hardware approaches like Power Gating, Voltage regulators etc. together by reducing the time it takes to implement a power cap on the RAPL sensor from the moment its value is set. But a workload-based adaptive control of power was not considered here. Petoumenos et al. [17], through their research give a comparative analysis of the software, hardware and hybrid approaches towards the power optimization in compute nodes.

Cerf et al. in her recent research [18] proposed a control theoretical approach to understand the possible operating regions of a compute node at which the performance decay is minimized under a reduced power supply. A classical proportional-integral-derivative (PID) controller was designed to impose power cap (PCAP) for RAPL actuators on an Intel(R) Xeon(R) Gold 6126 CPU by using performance feedback from the compute nodes [19]. The RAPL actuators comprising a control knob or the PCAP knob and a time-window knob were used to regulate the average power given between two sampling intervals to a compute node. A mathematical model formulated using static characterization was used for the design of the controller. The controller was shown to be working for the compute nodes by making it track a user-computed set-point for the desired performance. While an adaptive controller [20] addressed model inaccuracies, reliance on a user-defined set point calls for an automated approach.

Considering the limitations in prior work, we emphasize the design of an optimal controller by solving the underlying power and performance optimization problem. We propose a hybrid approach for determining the best operating power cap for a compute node by relying on the instantaneous performance analysis that is obtained using the state-of-the-art algorithms.

III. Background

To make the paper self-contained, this section provides the necessary background on the concepts and technologies used.

A. Reinforcement Learning

Reinforcement Learning (RL) iteratively improves a sequence of actions or strategy through continuous interactions with the environment over a finite or infinite amount of time traversed using discrete time steps. In this work we use an RL agent to learn an optimal control policy by constantly generating an action and evaluating it using a mathematical model that describes the relation between power cap and progress made by the compute node while running an application. The RL problem can be mathematically formulated as a Markov Decision Process (MDP), where the agent interacts with an environment that is modeled as a set of states, actions, and rewards. At each time step, the agent observes the current state, selects an action based on a policy, and receives a reward from the environment. The goal of the agent is to learn a policy that maximizes the expected cumulative reward over time. In this paper we use a model-based algorithm by employing a Proximal Policy Optimization (PPO) [21] agent to learn the state transition and the reward functions, thereby learning the policy.

B. Performance Measurement

Instantaneous measurement of the controlled variable is an important aspect of feedback control problems. The RL algorithm for maximizing performance under a controlled power cap being an optimization problem, also requires the measurement of instantaneous performance. During the measurement, care must be taken that it does not interfere with the overall working and thereby the performance of the system. Therefore, we use a lightweight instrumentation library [19] and follow the instrumentation steps outlined in [18]. We formally re-define the progress measurement equation introduced in the paper [19] considering the number of messages received between two sampling instants. Therefore in this paper the progress at the time instant t_i is given by:

$$progress(t_i) = \underset{\forall k, t_k \in [t_{i-1}, t_i]}{median} \left(\frac{N}{t_k - t_{k-1}} \right). \tag{1}$$

where N is the number of messages received between the time instants t_k and t_{k-1}.

C. Intel RAPL

The running average power limit (RAPL) is an interface available with modern Intel processors for power monitoring and controlling making it unavoidable in energy efficient computing. It also allows users to specify a power cap on available hardware domains using model-specific registers or the associated Linux sysfs subsystem. The RAPL interface uses two knobs, the power-limit knob and a time window knob to supply the average power for a user-defined period of time. The internal controller then guarantees that the average power over the time window is maintained. This mechanism also offers sensors through the same interface to measure the total energy consumed since the processor was turned on. Consequently, RAPL can be used for both measuring and limiting power usage [22].

IV. Methodology

This section describes details of our model-based reinforcement learning (RL) which relies on a mathematical model representing the relation between the progress and the power cap (PCAP) in a compute node for training the RL agent. In our previous work [18], where the performance of a compute node under a given application was controlled using a suitable PID controller running alongside the application, itself utilized all the available cores of the Skylake processor. However, a similar approach is not possible in the current work for training the RL agent because of the time and computational complexity involved. Therefore, we rely on the mathematical model for training the RL agent.

Figure 1 depicts our methodology while comparing it with the steps involved in the design of the PI controller in our prior work [18]. The processes involved in the two approaches beginning from the formulation of problem statement and extending till the evaluation are shown in the block diagram in Figure 1. It can be observed that the initial problem statement, the steps leading to the control formulation, and the system analysis are same in both the approaches. But, when it comes to the controller block, the PI controller relies on standard tuning methods from the classical control theory to determine its control gains, (i.e tuning the proportional, integral and derivative constants, K_p, K_d and K_i). In contrast, the RL method uses an iterative, policy improvement and policy update process by using an actor-critic network. It is only in this stage of the controller design that the two algorithms take different routes in the implementation. The rest of this section details the steps used in our approach, i.e., the branch shown in red color.

A. Problem Definition

The first step is to define the objectives and the constraints involved in the optimization problem. Our objective is to maximize performance within a given power cap (PCAP) constraint, while taking into account the relationship between progress and PCAP as determined by the node dynamics. In other words, we aim to improve the execution time of the application while still using the minimum allowed energy. A

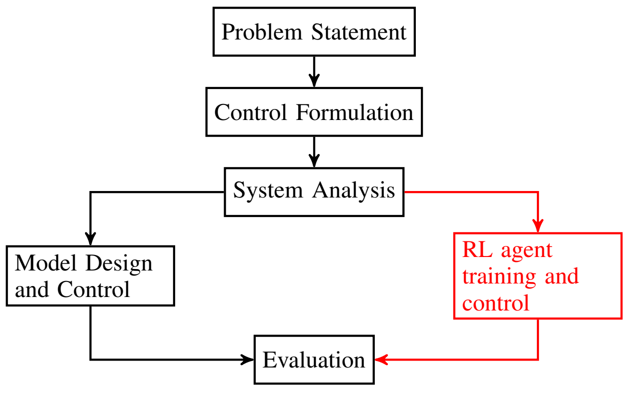

Fig. 1. A pictorial representation showing the differences in methodologies implementing a classical control and an RL based control. The RL method discussed in this paper is colored in red.

simple way of ensuring this behavior is by making sure that the computational progress is at its peak under the given PCAP.

B. Mathematical modeling and MDP identification

Given that the current progress of our problem is dependent on both the previous progress and the current power cap, it is expected that the node dynamics follow the characteristics of a Markov Decision Process (MDP). Thus, to effectively apply a reinforcement learning to a problem, it is crucial to identify the MDP tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$ representing the state space, a set of actions available to an agent, the unknown transition kernel, the reward function and the discount factor, respectively. In addition to identifying the MDP, we also need to derive a mathematical model that represents the node dynamics so as to train the RL agent. To support the claim that the node dynamics have MDP characteristics and to derive the mathematical model, we use a static characterization approach and utilize line-fitting algorithms from SciPy [23].

The experiment, which is part of the model building process, begins by collecting data points in the form of input-output pairs, where the input (i.e., PCAP) is provided at certain pre-defined instances and the output (i.e., progress made by the compute node on the workload) is calculated using Equation (1). These data points correspond to an entire benchmark execution where a constant PCAP is applied, and the progress signal is averaged. To model the time-averaged relationship between power cap and progress, we consider only stabilized situations of the progress value generated using PCAPs allowed by the RAPL actuators. At least 20 experiments are run for each compute node to generate enough data points. Based on these experiments, a static model is obtained by linking the time-stabilized power cap to the progress using line-fitting algorithms. We observed that the PCAP-progress relation follows an exponential relationship, as shown by Equation (2):

$$progress = K_L(1 - e^{-\alpha(a.PCAP+b-\beta)}), \tag{2}$$

where a and b parameters represent RAPL actuator accuracy of the node (slope and offset, respectively). Therefore, the actual power applied on a node can be approximated as: $a.PCAP+b$ which is also the measured power. α and β characterize the benchmark-dependent power-to-progress profile and K_L is the linear gain and is both benchmark- and node-specific. The effective values of the fitting coefficients given by α, β, a, b, K_L and τ are obtained using the above mentioned steps, where the line fitting problem follows a non-linear least squares optimization. The linear characterization of dynamics can be then computed using least square optimization [18] and is given as

$$\begin{aligned} progress_L(t_{i+1}) = &\frac{K_L \Delta t_i}{\Delta t_i + \tau}.PCAP_L(t_i) \\ &+ \frac{\tau}{\Delta t_i + \tau} progress_L(t_i), \end{aligned} \tag{3}$$

where $\Delta t_i = t_{i+1} - t_i$ is the control interval and τ is the time constant. The values of Δt_i and τ are chosen as 1s and $\frac{1}{3}$s, respectively, for the experiments. We can use either of these models for training.

After defining the Markov Decision Process (MDP), we proceed to assign distinct characters to each role in the MDP, which include the state, action, and observation spaces that were previously established. In this study, the current state, denoted as $s^{'} \in \mathcal{S}$, is represented by the progress value at time t, $progress_t$. On the other hand, the action, denoted as $a \in \mathcal{A}$, taken by the agent, is determined by the $PCAP$ value at time t based on the observation $progress_{t-1}$ $(s \in \mathcal{S})$, which represents the progress value at the time $t-1$.

C. Reward function

We now explain how the reward function is chosen to facilitate fast and accurate learning. The reward at a given time t reflects the quality of the control action applied to the system at that moment in time indicating how well the action helped achieve the goal or optimize the system's performance. The instantaneous reward, i.e., reward at a moment in time, is calculated by designing an appropriate reward function that maps the state-action space into the set of real numbers $\mathbb{R}$. The choice of an appropriate reward function depends on how effectively it can discriminate the changes happening in the environment when a control action is applied. As explained in the Subsection IV-A, our aim is to maximize the performance under a given PCAP. By negating the power consumption measured at time t during each of the sampling instance, we can easily take care of the power reduction i.e., $R(s, a) = -measured\ power(t)$. This reward function penalizes an action that can lead to an increased power consumption, but at the same time ignores its effect on performance.

An alternate choice could be $R(s, a) = -measured\ power(t) + progress(t)$, however, this is not a good choice for a reward function since it fails to record the trends in the power and performance. For example: A higher progress at a lower PCAP cannot be differentiated from another that may return a lower progress at a higher PCAP. Therefore, a suitable choice of reward function that can differentiate these modes of power and performance is $R(s, a) = \frac{progress}{measured\ power}$.

To give extra emphasis on the reduction of power, we decided to consider the following reward function consisting of a linear combination of the total measured power and the instantaneous progress to PCAP ratio:

$$R(t) = \underbrace{-c_1 * PCAP}_{\text{minimize consumed power}} + \underbrace{c_2 * \frac{progress(t)}{measured\ power(t)}}_{\text{maximize performance per watt}} \quad (4)$$

where c_1 and c_2 are scaling factors which we will determine by observing a large number of training performances. In the mathematical model developed using the static characterization in Equation 2, the measured power is computed as $measured\ power = e^{-\alpha(a.PCAP+b-\beta)}$ which we will be using for the learning process. We tested our algorithm with a variety of values for c_1 and c_2 and observed that more than one value exists as a valid candidate for c_1 and c_2. One interesting combination which we used for the training and testing is $c_1 = 0$ and $c_2 = 4.44$ which focused only on the maximization of the performance under the given power cap. Therefore, under the present simulation conditions, a reward function of $R(s, a) = \frac{progress(t)}{measured\ power(t)}$ is expected to be equally good when compared to an enhanced power reducing reward function given by Equation 4.

D. RL Agent Training

The RL system is trained using the mathematical model given by the Equation 3 which is linear in nature. Here, we follow a model-based training approach that is used in a variety of applications [24]. A model-based training has the advantage of not risking the safety and stability of hardware while evaluating a variety of control actions. Therefore, while training, we used the entropy term of the PPO agent to enable exploration. This helped in learning an optimal policy faster. The flow diagram in Figure 2 shows the general idea employed under the training using the mathematical model.

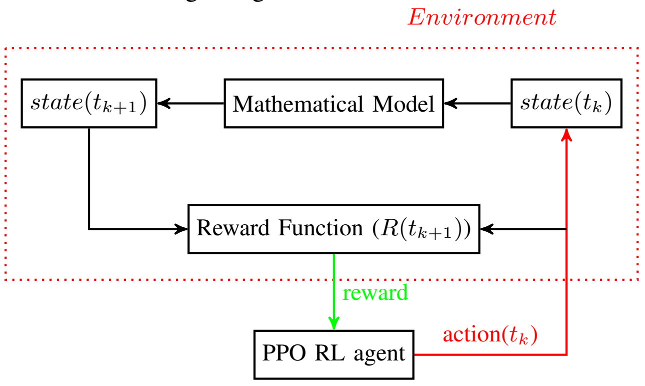

Fig. 2. Training of the RL-agent as described in the paper. The RL-agent receives a reward, for every action it takes, calculated using the values of state variables obtained from the mathematical model.

V. Empirical Evaluation of Reinforcement Learning Approach

This section presents details of evaluating our approach for its efficacy in meeting its objectives. First we present the criteria using which we evaluate the proposed method for controlling a data center compute node. We then present the results and our analysis.

A. Evaluation Strategy

We compare here the total time taken for executing the workload and the total energy consumption during that period, with two standard operating conditions of a data center compute node. The evaluation consists of the following analysis based on which we draw conclusions about the efficiency (percentage value of increase or decrease in the execution time and power consumption) of the proposed method:

- A comparative analysis of the proposed RL-based control method with the PI controller proposed in prior work [18],
- A comparative analysis of the RL-based control with the efficiency of the system with minimum and maximum power caps,
- A repeatability analysis of the experiment on the same compute node, and
- A repeatability analysis involving different compute nodes.

For the comparative analysis with the PI controller, we test our RL agents that are trained using a variety of reward functions and generated using varying values of c_1 and c_2 while running the benchmark application. Similarly we use the optimal controller among these trained models to control the PCAP, while running the benchmark application, and compare it against the maximum and minimum PCAP runs. We repeat this experiment 10 different times on a node to analyze the statistics associated with the results. We also test the algorithm on different nodes and analyze the statistics for the repeatability of the results on varying hardware. Through these evaluations, we also comment on the execution time, total power consumed, and the related statistics for each experiment.

B. Implementation of RL Agent and Experimental Platform

The code for the training and testing the algorithms are written in Python3 with the required support packages installed on the Nix-environment manager [25]. [26] introduces a Python3-based software package called stable-baselines-3 (SB-3) that give users a leverage to code RL algorithms seamlessly. With the support of PyTorch [27], we developed an easy to implement architecture for all the standard RL methods shown in the literature.

The RL agent resides inside the the Argo Node Resource Manager (NRM) stack [7]. Argo NRM is an infrastructure developed as part of the U.S. Department of Energy Exascale Computing Project called Argo, for the design of node-level resource management policies. It provides users an easy-to-use and a unified interface (through Unix domain sockets), to the various monitoring and resource control knobs available on a compute node (e.g., RAPL, performance counters).

All experiments in this paper were performed on Skylake compute nodes using the STREAM benchmark. STREAM [28] is a standard application benchmark comprising a set of simple, portable programs that can be used to measure memory bandwidth on a variety of computer systems. The benchmark measures the memory bandwidth for four different types of operations: copy, scale, add, and triad. STREAM is chosen as it is representative of memory-bound phases of applications and shows a stable behavior. STREAM is also easy to modify into an iterative application: its four kernels are ran a configurable number of times in a loop, with a heartbeat being reported to the NRM each time the loop completes, i.e., after one run of the four kernels.

All four tests are performed on arrays of floating-point numbers and the results are reported in terms of the achieved memory bandwidth (in bytes/second). The problem size is set to 33,554,432 with 10,000 iterations for the experiment during its evaluation. In the context of the STREAM benchmark, the problem size refers to the amount of data in bytes that will be used to execute the benchmark. The problem size can be varied to measure the memory bandwidth at different data set sizes. The iterations refer to the number of times that the benchmark will be executed for a given problem size, and the results of the iterations are averaged to reduce the impact of measurement variability. Typically, a large number of iterations are used to obtain accurate and consistent results.

Note that availability of RAPL actuators and sensors on the compute nodes are important for our experiments. Moreover, each node should have a reasonable range of operation region under the given power cap before it attains saturation so that a number of them can execute the STREAM benchmark seamlessly. We use Chameleon Cloud [29], a server facility made available for researchers to perform experiments that requires low-level access to hardware. Hosted on a variety of locations spanning across the country, it offers hardware components of various specifications. We use the Skylake nodes from the available servers for our experiments that use an Intel Xeon Gold 6126 processor with 12 dual threaded cores. Above all, it belongs to the modern series of the Intel processors with RAPL actuators and sensors.

C. Experiment Details

For the first part of the experiment involving the static characterization, described in the Subsection IV-B, we used a Chameleon Cloud node to run the STREAM benchmark while the daemon collected the progress measurement. We actuated the RAPL power knobs with test PCAP values in the form of step signals. A total of 17 step signals were given and the progress values were calculated using Equation 1 and recorded once the steady state was attained. We then collected these datasets consisting of the PCAP-progress pairs and used the Scipy line fitting algorithm to obtain the best fit equation. Figure 3 depicts the non-linear and linear representations of data points and the curve that was obtained using static characterization, respectively. The parameters obtained after the line fitting are given in Table I.

In the next part of the experiment, the obtained parameters were used in the mathematical model shown in Equation 2 for generating the next state (progress) in response to an action (PCAP). We employed a PPO-RL agent, to generate a policy and execute it on the mathematical model thereby generating the next state followed by the corresponding reward given

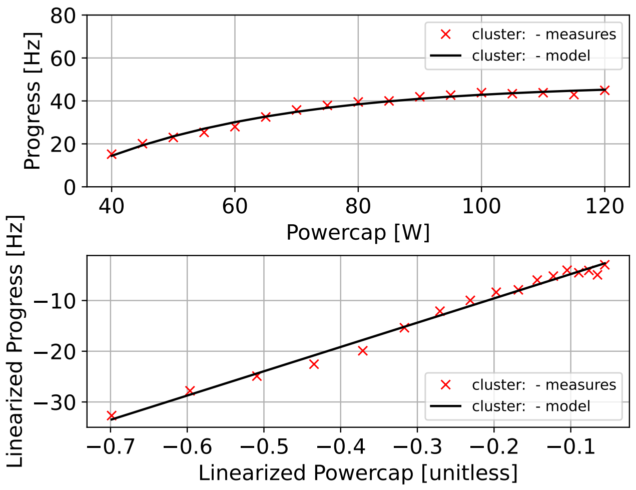

Fig. 3. Mathematical model of an HPC node representing the non-linear and the corresponding linear relations existing between PCAP and progress, obtained using static characterization, showing the data points (red '×') and the best fit curve (black '−').

Description	Notation	Unit	Value
RAPL slope	a	[1]	0.95
RAPL offset	b	[W]	0.15
	α	[W^{-1}]	0.041
power offset	β	[W]	24.3
linear gain	K_L	[Hz]	47.9
time constant	τ	[s]	1/3

TABLE I
MODEL PARAMETERS OBTAINED USING STATIC CHARACTERIZATION OF AN INTEL XEON GOLD 6126 PROCESSOR.

by Equation 4. At this point, we would like to bring to the reader's attention that we are not running the application nor the NRM daemon while using the mathematical model. We can implement this training algorithm on any standard processor and do not require any specific hardware. Therefore a simple Python code deploying the PPO-RL agent evaluated the policy, followed by periodic updates which assured its convergence to an optimal value.

For determining the best reward function that could generate an optimal policy, we performed the training using different reward functions, obtained by varying the c_1 and c_2 in the Equation 4. We then collected all the models and tested it on the mathematical model to determine its impact on the theoretical performance and power, and determined the region of interest. We also tested these models on the hardware running the application and daemon to compare it against the PI controller.

Having determined the optimal reward function, we used the optimal model on the hardware for testing followed by evaluation. For the testing and evaluation, we reintroduced the hardware and the workload to execute, with the help of NRM. This time, the trained RL-agent provided the PCAP for controlling the power knob of the RAPL actuators. Following this, the corresponding progress values were measured and fed back into to the RL network to compute the next PCAP. This cycle is repeated till the entire work load is finished execution. We repeated the experiment for a variety of scenarios described in the SubsectionV-A.

D. Empirical Results and Discussion

In this section we will explain the results obtained after following the experiment design described in the previous section. We will also perform the evaluation of the algorithm followed by the interpretation of the results obtained.

Firstly for choosing an appropriate reward function we trained the RL-agent using the reward function given by the equation (4) with varying values of c_1 and c_2 followed by testing on the mathematical model for its performance. The theoretical values of performance (execution time) and the total energy consumed during the execution, were obtained using the control generated by each of the agents and are as shown in the Figure 4. By using a series of values for c_1 and c_2 starting from 0 followed by increments of 0.1, till 10, we were able to observe the variations in the performance and power for each of the designed reward. As expected we obtained a combination of values for c_1 and c_2 that yielded respective results ranging from the fastest to the slowest performance as shown in the figures 4. Observing the figure we can see that there is a point in the maximum curvature region (marked in blue circle) in figure 4 that can be considered as an ideal region to look for a candidate reward function. This point corresponds to the minimum energy consumption and the maximum performance. Note that, there were a couple of ideal candidates for the reward function that repeated across multiple execution of this experiment, for example, (c_1=1.052,c_2=2.22) and (c_1=0,c_2=4.44) gave the same responses. We chose the model, that was obtained after training using the reward function, $R(t) = -1.052PCAP(t)+2.22\frac{progress(t)}{measured\ power(t)}$, as the RL-agent to control the HPC nodes during the testing phase.

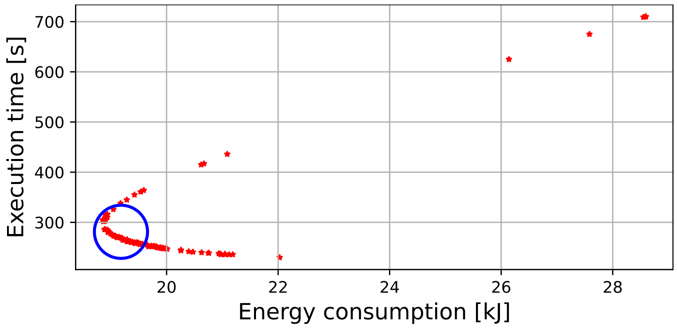

Fig. 4. A Pareto of execution time vs energy consumed, generated by varying the RL-agents trained with different reward functions. The x-axis shows the total energy consumed during the execution, and the time of execution is plotted on the y-axis. The ideal characteristics are inside the blue circle.

E. Analysis of Results

Before we proceeded with the analysis and evaluation of the proposed optimal controller for the PCAP, we needed to verify its performance by comparing it against the existing PI controller. For that, we needed to run both the algorithms on the same machine. We first executed the PI controller on the HPC node with varying set-points as shown in the Figure 5. The pareto of points with gradients of colors ranging from yellow to blue depicts the result obtained for the PI controller with varying setpoints. The setpoint was specified using the parameter ϵ, representing the allowed tolerance in the maximum performance value. The ϵ value was varied from 0 to 0.6 randomly, during the experiments using PI controller. The shortest execution time of 240s (corresponding to faster execution) was obtained at an energy consumption of 36kJ at an ϵ value of 0.1. The setpoint value is expected to change based on the hardware being tested, whose dependency on the controller is being removed while using an RL controller. We then tested the trained RL controllers on the same hardware running the application and created a similar pareto of points. In the Figure 5, the red points represent the response of the HPC node controlled using the trained RL-agent. We observed that the optimal region consisted of points vary close to the best performance obtained using the PI controller. It was observed that the $R(t)$ proposed in the previous subsection fell within a closer range of the best performance.

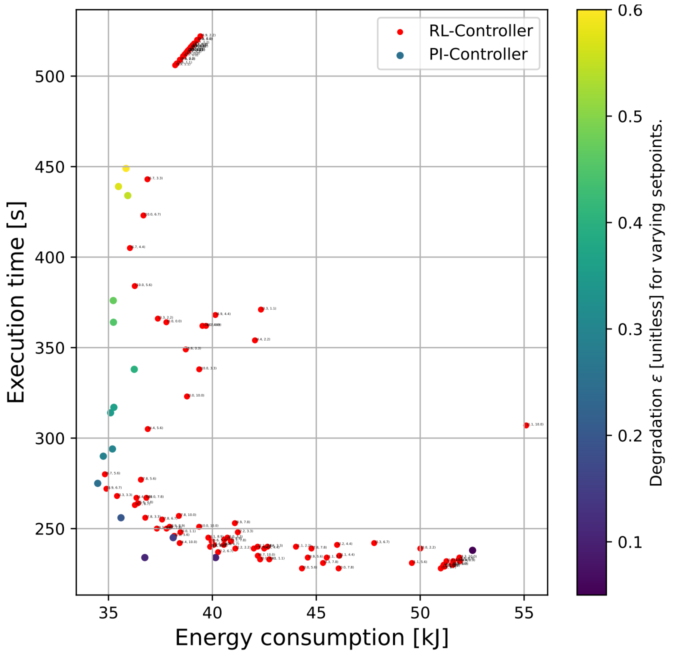

Fig. 5. Plot showing the comparison of PI controller with varying setpoints (values for $1 - \epsilon$) and RL controller for varying reward functions. The PI control objective is given as a degradation factor ϵ, that is, the tolerable loss of performance

Through these comparisons we were able to justify the use of a trained RL-agent as an optimal controller aimed to minimize the power consumption. Using the proposed RL controller, we were able to regulate the HPC node performance around those regions of operation, where a PI controller was regulating the performance, given a user-defined set-point. We also were able to excite the HPC node at some other regions where a slower response was obtained but at a very low energy consumption. We will now look for points where optimal performance and power consumption was recorded and will check for the repeatability of the experiments using the same hardware as well as different hardware.

We used the RL-agent trained using the reward function chosen previously, for the remaining part of the experiment consisting of:

- Repeatability on the same node and
- Repeatability on different nodes.

Within each of these analysis, we considered two different scenarios to compare with the performance of the proposed method, namely:

- Maximum PCAP: allowing the node to utilize the maximum allowed capabilities of the system,
- Minimum PCAP: limiting the performance, by allowing only the minimum power at the actuators.

For the analysis of the repeatability of the experiment on the same node, we conducted a total of 30 executions of the experiment with RL controller being used for 10 of the executions. In the remaining 20 executions we used the maximum allowed and minimum allowed values for the PCAP to record the statistics. We didn't consider the PI controller for the comparison here, since we are only evaluating the repeatability of the proposed method. We then observed the statistics associated with each set of executions and the values are tabulated. Figure 6 depicts the results of the experiment with the execution time plotted against the energy consumed during the each of the experiments. Figure 7 shows the values of instantaneous PCAPs sensed during each time step, by the RAPL sensors for three different executions using maximum, minimum and optimal PCAP respectively. In Figure 6,

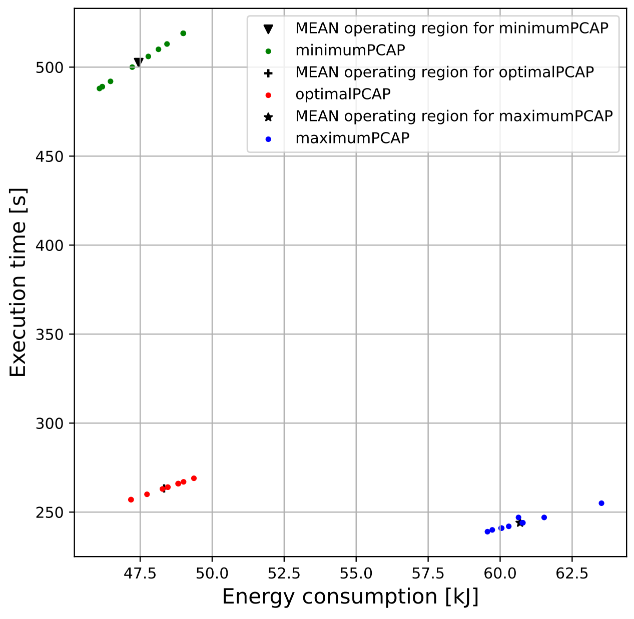

Fig. 6. Comparison of the execution-time and the energy consumed during the execution of the experiments performed on the same hardware employing three different methods. The statistics of the executions are shown in the table II

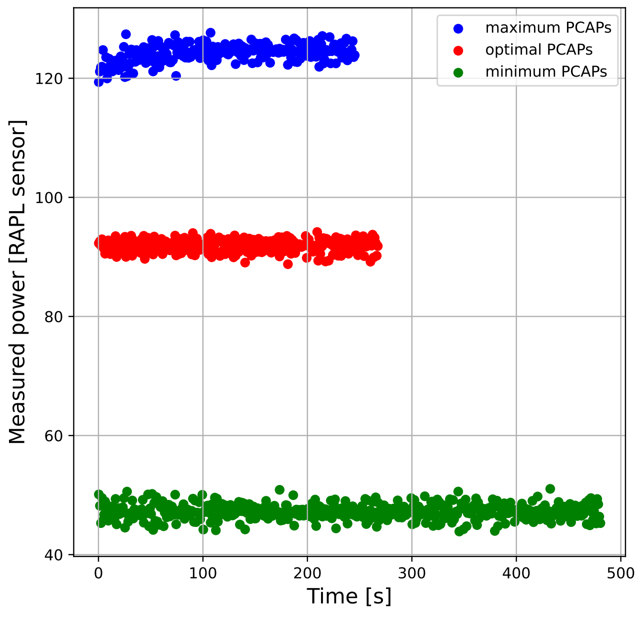

Fig. 7. Instantaneous power vs time-steps plots for a single execution of the experiment using optimal, maximum and minimum PCAP controllers.

ensuring the maximum PCAP for the RAPL actuators, that put no limit on the amount of energy drawn into the system, the workload execution was completed in 245s. But when we look at the total energy consumed over the period it was in the range of an average value of 60.97kJ. On the other hand, enforcing the minimum PCAP on RAPL actuators, the power consumption was minimal, i.e in the range of an average value of 47.49kJ of energy, but compromising on the total execution time which lasted for 501.14s. On the other hand, the RL based controller was able to limit the power consumption and comparable execution time. It was observed that the average power consumption, over 10 experiments using the optimal PCAP, was 48.23kJ, while the average execution time was 261.19s. This shows a reduction of 21% on the consumed power, while the performance was compromised only by 6.5%. The statistics of the experiment are given in the table II

	Minimal	Maximal	Optimal
μ Execution Time [s]	501.14	245.23	261.19.
σ Execution Time	26.20.21	3.6	5.89
μ Energy Consumption [kJ]	47.49	60.97	48.23
σ Energy Consumption	1.98	0.77	0.744

TABLE II
STATISTICS OF REPEATABILITY USING THE SAME NODE OVER 10 EXECUTIONS. μ AND σ DEPICT MEAN AND STANDARD DEVIATION RESPECTIVELY.

For the next part, we performed a total of 21 exections consisting of 7 sets with 3 executions each with maximum, minimum and optimal PCAP. Each set was exectuted on different hardware. We made the reservation for 7 nodes from the Chameleon Cloud using a single lease in-order to assure diversity in hardware. The results of the execution was then gathered for analysis of its statistics. Figure 8 shows

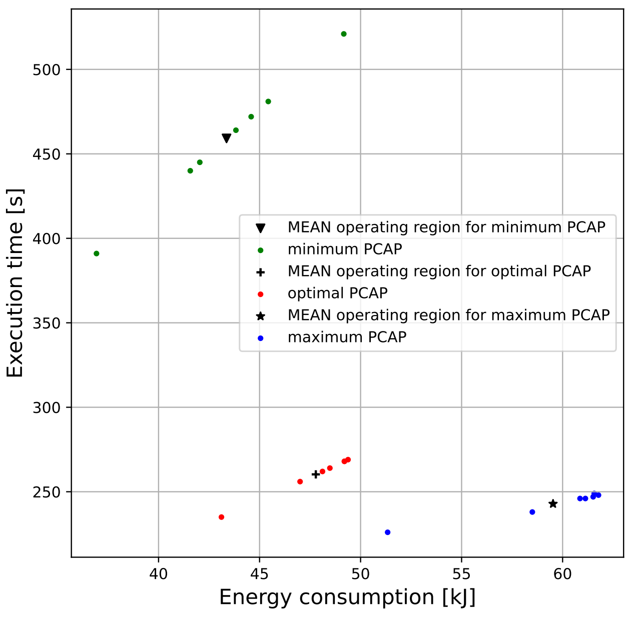

Fig. 8. Comparison of the execution-time and the energy consumed during the execution of experiments, performed on different nodes using the three methods. The statistics of the executions are shown in the table III

the obtained results of the experiment. It was observed that the proposed RL based controller was able to bring down the energy consumption by 19.7% at a cost of 7.3% in performance. It was also observed that the variance among each set of the experiments have gone high due to the changes in hardware, but in effect the optimal controller performed with the minimum variance. The related statistics are presented in the table III. Through the repeatability experiments we were

	Minimal	Maximal	Optimal
μ Execution Time [s]	459.14	242.85	260.28
σ Execution Time	40.21	8.25	12.03
μ Energy Consumption [kJ]	43.3	59.52	47.78
σ Energy Consumption	3.78	3.77	2.22

TABLE III
STATISTICS OF REPEATABILITY OVER 5 EXECUTIONS ON DIFFERENT HPC NODES. μ AND σ HAVE THE USUAL MEANING.

able to prove the reliability of the proposed approach for generating an optimal control for HPC nodes. We were also able to verify that the energy consumption reduction is consistent throughout its execution irrespective of the hardware.

VI. Conclusion

This paper addresses the problem of managing power consumption in heterogeneous data center compute nodes by focusing on the potential of dynamically adjusting power across compute elements to save energy with almost no impact on performance. Our approach involves model-based reinforcement learning that learns using a mathematical model, which is generated using static characterization of a specific hardware node running a standard application benchmark. The control of the hardware, then uses the trained RL agent to ensure optimal performance. Experimental validation shows promising results for systems running a memory-bound benchmark. The repeatability of results showed a standard deviation of 5.89 with a mean energy consumption of 48.23 kJ during 261.19s of mean execution time. These results indicate that the RL-based approach can serve as an alternative to the PI controllers used in our past experiments in order to remove the dependency on operational setpoints. The approach presented in this paper is most suitable for cloud providers and high performance compute node administrators who seek solutions to reduce power consumption of their server resources yet not impact application performance. In our future work, we aim to resolve the dependency of the proposed work on the mathematical model for training. We also seek to design a generalized RL-based controller, controlling PCAPs on nodes executing a variety of applications.

Acknowledgments

Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation. Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computer Research, under Contract DE-AC02-06CH11357. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. The dataset and the codes used for the entire experiment, including the static characterization, training and testing, are hosted on our GitHub repository. We also provide readers with complete access to our Chameleon Cloud image, which will enable them to reproduce the results.

References

[1] K. Darrow and B. A. Hedman, *Opportunities for combined heat and power in data centers*. ICF International Arlington, VA, 2009.

[2] J. Koomey *et al.*, "Growth in data center electricity use 2005 to 2010," *A report by Analytical Press, completed at the request of The New York Times*, vol. 9, no. 2011, p. 161, 2011.

[3] M. Koot and F. Wijnhoven, "Usage impact on data center electricity needs: A system dynamic forecasting model," *Applied Energy*, vol. 291, p. 116798, 2021.

[4] J. Shalf, "The future of computing beyond moore's law," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20190061, 2020.

[5] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 732–794, 2015.

[6] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, "United states data center energy usage report. 2016," *DOI*, vol. 10, p. 1372902, 2016.

[7] B. Pete, D. Nicolas, G. Rinku, I. Kamil, P. Swann, T. John, V. Brice, and Y. Kazutomo, "Argo Node Resource Manager," 2021.

[8] C.-h. Hsu and W.-c. Feng, "A power-aware run-time system for high-performance computing," in *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pp. 1–1, IEEE, 2005.

[9] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. De Supinski, "Practical resource management in power-constrained, high performance computing," in *Proceedings of the 24th international symposium on high-performance parallel and distributed computing*, pp. 121–132, 2015.

[10] M. Sadrosadati, S. B. Ehsani, H. Falahati, R. Ausavarungnirun, A. Tavakkol, M. Abaee, L. Orosa, Y. Wang, H. Sarbazi-Azad, and O. Mutlu, "Itap: Idle-time-aware power management for gpu execution units," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 16, no. 1, pp. 1–26, 2019.

[11] M. Qiu, Z. Ming, J. Li, S. Liu, B. Wang, and Z. Lu, "Three-phase time-aware energy minimization with dvfs and unrolling for chip multiprocessors," *Journal of Systems Architecture*, vol. 58, no. 10, pp. 439–445, 2012.

[12] F. Caglar, S. Shekhar, and A. Gokhale, "iplace: An intelligent and tunable power-and performance-aware virtual machine placement technique for cloud-based real-time applications," in *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pp. 48–55, IEEE, 2014.

[13] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 9, pp. 1395–1408, 2010.

[14] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," in *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pp. 189–194, IEEE, 2010.

[15] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No" power" struggles: coordinated multi-level power management for the data center," in *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pp. 48–59, 2008.

[16] H. Zhang and H. Hoffmann, "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 545–559, 2016.

[17] P. Petoumenos, L. Mukhanov, Z. Wang, H. Leather, and D. S. Nikolopoulos, "Power capping: What works, what does not," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 525–534, IEEE, 2015.

[18] S. Cerf, R. Bleuse, V. Reis, S. Perarnau, and E. Rutten, "Sustaining performance while reducing energy consumption: a control theory approach," in *European Conference on Parallel Processing*, pp. 334–349, Springer, 2021.

[19] S. Ramesh, S. Perarnau, S. Bhalachandra, A. D. Malony, and P. Beckman, "Understanding the impact of dynamic power capping on application progress," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 793–804, IEEE, 2019.

[20] I. Hawila, S. Cerf, R. Bleuse, S. Perarnau, and E. Rutten, "Adaptive Power Control for Sober High-Performance Computing," in *CCTA 2022 - 6th IEEE Conference on Control Technology and Applications*, (Trieste, Italy), pp. 1–8, IEEE, Aug. 2022.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[22] B. Rountree, D. H. Ahn, B. R. De Supinski, D. K. Lowenthal, and M. Schulz, "Beyond dvfs: A first look at performance under a hardware-enforced power bound," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pp. 947–953, IEEE, 2012.

[23] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[24] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721, IEEE, 2017.

[25] E. Dolstra and A. Löh, "Nixos: A purely functional linux distribution," in *Proceedings of the 13th ACM SIGPLAN international conference on Functional programming*, pp. 367–378, 2008.

[26] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, 2021.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[28] J. D. McCalpin, "A survey of memory bandwidth and machine balance in current high performance computers," *Newsletter of the IEEE Technical Committee on Computer Architecture (TCCA)(December 1995)*, 1997.

[29] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, "Lessons learned from the chameleon testbed," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*, USENIX Association, July 2020.