2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 979-8-3503-9566-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/CCGrid59990.2024.00042

Opportunistic Energy-Aware Scheduling for Container Orchestration Platforms Using Graph Neural Networks

Philipp Raith*†, Gourav Rattihalli†, Aditya Dhakal†, Sai Rahul Chalamalasetti†,
Dejan Milojicic†, Eitan Frachtenberg†, Stefan Nastic* and Schahram Dustdar*
*TU Wien, Vienna, Austria
{p.raith, s.nastic, dustdar}@dsg.tuwien.ac.at
†Hewlett Packard Labs, Milpitas, USA
{first.lastname}@hpe.com

***Abstract*—Reducing the energy consumption of data centers is critical to meeting international climate goals and lowering operation costs. Container orchestration platforms can help counteract this trend by optimally placing applications across the infrastructure to increase resource utilization and reduce energy consumption. But platforms in use today are still energy-agnostic and do not offer any insights into energy consumption. In this paper, we present a monitoring framework and a new modeling approach for resource usage in data centers. The model captures heterogeneous hardware and software and acts as input for a Graph Neural Network (GNN) to predict power consumption. Based on this model, we derive a set of container scheduling algorithms that opportunistically schedule applications based on the estimated energy impact of incoming containers. Our results show that the GNN-based prediction model is very accurate and achieves an average RMSE (Root Mean Square Error) of 7.5%. We have implemented a custom scheduler to demonstrate the benefits of using our prediction, and our scheduler can decrease energy consumption on average by 6.2% without any code changes for the application and without increasing workload completion time compared to the default Kubernetes scheduler.**

***Index Terms*—GNN, Energy estimation, energy-aware scheduler**

I. Introduction

Data centers may be responsible for around 3-13% of global energy consumption by 2030 [1]. Therefore, platforms are called to reduce energy consumption and increase sustainability [2], [3]. Especially container orchestration platforms are prevalent in today's cloud and edge-cloud platforms because they offer autonomous application and resource management for large-scale application deployments [4], [5]. These platforms facilitate the deployment in many application domains, such as AI and HPC [6], [7], but also build the foundation for other platform paradigms, such as serverless computing [4], [8]. However, current container orchestration platforms, such as Kubernetes or Apache Mesos, lack autonomous energy-aware management strategies [9], [10]. Therefore, it is imperative to develop more sustainable containerized platforms [3]. This study investigates energy-aware resource management in container orchestration platforms to reduce energy consumption. Specifically, we focus on an energy-aware container scheduler using power prediction to make data centers more sustainable, which has not been integrated yet into existing platforms and remains a relatively unstudied problem in serverless computing [3], [11]–[13]. Energy estimation of future applications can facilitate this development. To this end, we introduce an end-to-end scheduling approach that includes a monitoring system and a Graph Neural Network to estimate energy consumption, which our scheduling algorithms use. We introduce a novel graph model that allows us to perform *what-if* estimations. For example, we can estimate the energy consumption of a host if we schedule a specific container onto it using historical data. Based on that, we implement three energy-aware container scheduling algorithms that use this prediction in a real-world setup. Therefore, the algorithms opportunistically estimate the energy impact of applications and base their decisions on this. However, simply estimating the average power consumption of an application to estimate the total energy consumption may also not be accurate, as resource usage varies over time. Our approach solves these challenges using a flexible graph representation. The graph-based resource usage representation allows the combination of arbitrary applications and hosts, accounts for different resource phases throughout the application's lifecycle, and supports heterogeneous systems. However, integrating a power-prediction model into container-based platforms poses several challenges. While tools exist to energy monitoring tools exist, they vary from platform to platform, making energy estimation a universally applicable approach to estimating energy consumption. Our energy estimation builds the foundation of our energy-aware container scheduler, but it can also facilitate other strategies to reduce a data center's power consumption. For example, estimating the power consumption of the hardware accelerators for specific models before deployment can help select more efficient hardware [12], cooling [14] and facilitate the development of carbon-aware strategies.

Our contributions in this paper are as follows:

- A novel graph model for hosts and a Graph Neural Network to predict the power consumption of hosts in con-

979-8-3503-9566-2/24/$31.00 ©2024 IEEE
DOI 10.1109/CCGrid59990.2024.00042

tainer orchestration platforms. While our custom dataset is comprised of CPU-focused profiling experiments, the graph model can include arbitrary accelerators.
- The integration of the power prediction into Kubernetes which builds on commonly used open-source projects for monitoring resource usage.
- An energy-aware scheduling algorithm based on our power prediction and integration with Kubernetes.

Our results show that the model can account for existing accelerators and has a mean RMSE of 7.5%. On an on-premise cluster, the scheduler evaluation includes hosts with similar hardware specifications but observes different power consumption due to a GPU on one. Moreover, the results show that our energy-aware scheduler can reduce energy consumption by 6.2% while maintaining good performance, demonstrating the viability of our approach in real-world data center management activities.

The remaining work is structured as follows. In Section II, we discuss related work, and in Section III, we explain our approach to solving the energy prediction for a workload, the required monitoring to support the prediction, and energy-aware scheduler algorithms. In Section IV, we talk about the evaluation methodology for generating the training dataset and a use case for our prediction in a Kubernetes-based custom scheduler, and in Section V, we present and discuss the results of our experiments in multiple scenarios. In Section VI, we summarize our work's next steps and conclude in Section VII.

II. Related Work

While energy awareness in cloud applications and resource management has been extensively studied [15]–[18], limited work has been done in exploring energy awareness for container orchestration [3], [13], [19]–[21]. To this end, we review work spanning from carbon- and energy-aware serverless and container-based systems to power-consumption prediction and the use of GNNs in resource management.

A. Carbon- and Energy-Aware Approaches

We consider carbon-aware approaches related as we present an AI model to predict power consumption, which can be easily combined with real-time carbon emission intensities (CEI) to build carbon-aware system management tools. Real-time CEI updates are publicly available through different third-party websites, such as WattTime[1] and electricityMaps[2]. Hanfy et al. [22] present CarbonScaler, a greedy algorithm for carbon-aware scaling that has been integrated into Kubernetes and evaluated using machine learning training and MPI jobs. While our work does not consider carbon emissions, the power consumption estimation we propose can facilitate future carbon-aware strategies. Moreover, our power consumption estimation also differentiates us to other scheduling approaches. By offloading applications to other nodes, Aslanpour et al. [11] explore energy-aware serverless edge computing. Rastegar et al. [13] minimize CPU utilization and introduce an energy model tied to CPU utilization. In contrast, our power-consumption prediction and the scheduler we build atop can model different resources such as CPU, I/O, and hardware accelerators. Caspart et al. [12] explore energy consumption of AI workloads on heterogeneous nodes, highlighting the need for models that predict based on resource usage and the high impact of hardware accelerators on power consumption. Our work aims to address this issue by proposing a flexible model for power consumption prediction.

[1]https://www.watttime.org/

[2]https://www.electricitymaps.com/

B. Power Consumption Prediction

Power consumption predictions have been explored from various perspectives, such as container-based systems and general purpose predictions. Ou et al. [23] present a random-tree-based power consumption prediction in container-based systems. They highlight the importance of container-level predictions, which we implicitly model by estimating the power consumption of the host if a certain container would run on it. Moreover, our approach follows a fine-grained power consumption prediction model that allows to account for different resource phases throughout the application's lifecycle. Other studies have presented linear consumption models for heterogeneous servers [24], reviewed energy consumption models for data centers [25], and surveyed different models [26]. Others explore power consumption estimation using different resource usage characteristics, such as I/O-based features [27], [28] However, the most significant differences with our approach are the use of a graph to model a host, the use of a GNN for predictions, and the inherently flexible definition of heterogeneous hosts with which intrinsic relationships can be captured in contrast to other machine learning methods [29].

C. GNNs in Resource Management

Next, we discuss related work that uses GNNs for scheduling or resource management strategies. Tulli et al. [30] use a GNN to predict the Quality of Service (QoS) in scheduling given a graph of containers and hosts while considering different factors, such as temperature. The main difference between our approach lies in the graph model and the task we solve. We also model hosts at a much finer level. Moreover, our GNN predicts power consumption given an application placement, while theirs predicts QoS. Tam et al. [31] also use GNNs to predict the performance of applications. Specifically, they model microservice graphs as input for GNNs to predict end-to-end latency.

III. Approach

Figure 1 highlights our approach of building an end-to-end framework for energy-aware scheduling. We start by explaining the monitoring setup and graph model definition.

A. Monitoring & Graph Definition

The monitoring infrastructure captures telemetry data for all hosts $h \in H$ in a data center. Hosts expose a set of sensors S_h, each reporting back a single measurement (number) every

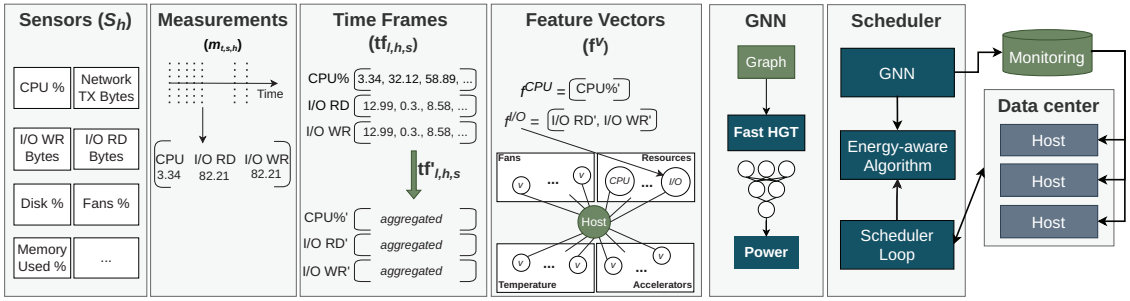

Fig. 1: GNN-based Energy-Aware Scheduling

second, which we denote as $m_{t,s,h}$, where t is the timestamp, s the sensor and h the host. We divide measurements into two types, counter, and gauge, whereas counter measurements can only increase (e.g., CPU time spent) while gauge ones can vary (e.g., RAM usage). Based on this monitoring setup, we now explain how to preprocess the measurements into graphs that act as input for our GNN. We create time frames that group *n* consecutive measurements per sensor and host. Throughout our work, we set *n* to 10, *i.e.,* each frame contains ten seconds of measurements. We denote each time frame as $tf_{l,h,s}$, where *l* is the index of the time frame created by dividing the original measurements $m_{t,s,h}$ of sensor s on host h by *L*, the number of time frames. The measurements in $tf_{l,h,s}$ are aggregated based on their type. In the case of a counter measurement, we calculate the total amount of work done within a time frame (e.g., the sum of differences between measurements). For gauge measurements, we calculate the arithmetic mean value of all measurements in $tf_{l,h,s}$, resulting in $tf'_{l,h,s}$. Next, we can create *L* heterogeneous graphs for each host, each capturing the resource usage of one time frame. A simple way of understanding this step is by thinking of each graph containing the aggregated time frames of each sensor of one host. This approach allows the capturing of varying resource usage over time, and granularity, *i.e.*, the size of time frames is flexible.

A heterogeneous graph (HG) is defined as $G = \{V, E\}$. V represents the node set and E the set of edges. Each node and edge belong to a node (N) or edge type (R). In our implementation, node types represent different resources, each being monitored by one or multiple sensors, and edge types represent semantic or physical connectivity between them, where:

$$N = \{\textit{Host}, \textit{CPU}, \textit{Memory}, \textit{Disk}, \textit{Network}, \textit{Pressure}, \textit{GPU}, \textit{Fan}, \textit{Temperature}, \textit{FPGA}, \textit{SmartNIC}\} \quad (1)$$

$$R = \{\textit{Host-CPU}, \textit{Host-Memory}, \textit{Host-Disk}, \textit{Host-Network}, \textit{Host-GPU}, \textit{Host-Pressure}, \textit{Host-Fan}, \textit{Host-Temperature}, \textit{Host-Pressure}, \textit{Temperature-Memory}, \textit{Temperature-CPU}\} \quad (2)$$

All sensors are measured on a node level, and we exclude container-level metrics as we have seen that generalization suffers from modeling each container as a node in the graph. Each node v in a graph is associated with a feature vector f^v that only contains numerical values and whose length can differ across node types. The feature vector's values can be static (e.g., GPU model) or dynamic (e.g., current GPU frequency), for which we introduced the preprocessing (i.e., $tf'_{l,h,s}$). The unweighted and undirected edges connect each resource type with the *Host* node, and *Temperature* nodes are connected with resource types they report the temperature on (e.g., *CPU*).

B. Application Signature

The graph model also allows us to model the resource usage of applications, which builds the base of our *what-if* estimations. However, only a subset of sensors can be isolated from the host's resource usage. For example, we can measure the CPU usage per application, not the temperature or fan increase it causes. To this end, we use sensors that capture the resource usage at the container level for the following node types: *CPU*, *Memory*, *Disk* and *Network* using cAdvisor, a popular monitoring tool for container-level metrics. We introduce the application signature $as_{a,h} \in \textit{AppSig}$, ($a$ being the application), that represents the isolated resource usage of an application a on a host h, in the form of graphs. Next, we introduce a function *merge* : $(G, G) \rightarrow G$ that combines a graph of an application signature and a graph representing a host to one graph. The function adds the values of the counter type measurements together and calculates the arithmetic mean for gauge measurements. The application signatures allow us to perform lightweight *what-if* simulations to estimate the power consumption of a host based on its current system resource usage and an arbitrary application. Because as contains a set of graphs, representing the resource usage of an application over time, our model is aware of different resource phases during an application's execution.

C. Graph Neural Network Architecture

The resulting graphs are used as input for our GNN. Its architecture builds on an existing one presented by Hu et al. [32]. We use the *fast* implementation of the HGT Convolutional layer, which is stacked two times, and then use three global pooling methods (i.e., mean, max, and add) to create a 1-dimensional vector that is passed through four linear layers, which output the final vector containing the power

consumption predictions (min, max, avg) for a given graph (time frame).

D. Opportunistic Energy-aware Scheduling Algorithms

We propose three scheduler variants that differ in their decision-making, but all use the GNN for power-consumption prediction. Specifically, we take an opportunistic approach by viewing each container to schedule individually and its impact on energy consumption, in contrast to others that find an optimal placement given a set of containers.

1) Base Algorithm (GNN): The base energy-aware scheduling algorithm fetches each host's current state and adds the signature of the container to schedule. To accomplish this, we build graphs for each host and combine them with the container's application signature using the $merge$ function. The output of this is a sequence of graphs that represent the estimated resource usage of the container running on the host. Next, we use the GNN to estimate average power consumption in Watts per graph, which we use to calculate the total energy consumption. The last step finds the host with the minimum estimated total energy consumption.

2) Application-Aware (GNN-Aware): We also propose an application-aware algorithm that works similarly to the one just described but considers the future resource usage of applications concurrently running. For example, the *base* algorithm fetches the current resource usage of each host and then adds on top the application to schedule. However, applications already running on the host might end soon or run for a long time—the system resource usage varies. It requires the assumption that we have the application signature for each application running, which allows us to estimate the remaining run time and dynamically change the resource usage of each prediction depending on which point of processing they are at.

3) Packing (GNN-Packing): The last scheduling algorithm integrates with both of the previously introduced algorithms. The *Packing* strategy modifies the selection algorithm by considering the host with the second lowest estimated total energy consumption. We introduce a threshold with which we determine to take this instead of the lowest. We select it if the second one's energy consumption is within the range of the first one by adding the threshold on top. We set the threshold to 5% for all experiments. We do this to prefer nodes with more applications running to further dampen the power consumption increase.

E. Implementation

After explaining the envisioned prediction and scheduler theoretically, we describe key aspects of our implementation.

1) Monitoring: Our monitoring system is based on Prometheus, a time-series database that monitors the infrastructure by following a pull-based monitoring approach. This means that on each host, multiple exporter applications expose a range of metrics via HTTP. Prometheus scrapes each exporter every second and stores it in its database. We deploy cAdvisor[3] to monitor containers, NodeExporter[4] to monitor host metrics and the Nvidia DCGM[5] exporter to monitor GPU usage, as well as two custom exporters for HPE iLO [6] and Xilinc's xbutil [7] tool to monitor FPGAs. We observe 16 dynamic features from cAdvisor, 37 dynamic and static metrics from NodeExporter, and fan and temperature from iLO. While we did not run workload using hardware accelerators, we are still monitoring it and integrate it into our graph during preprocessing. Namely, 22 metrics for GPUs, six for FPGAs and three for SmartNICs. We must include them because it allows the model to learn the individual power consumption of very similar hosts (i.e., same CPU and memory) but have a different idle power due to equipped hardware accelerators.

Stressor	Resource	*stres-ng* Arguments
cpu	CPU	cpu (1- 88)
memrate	Memory	memrate (1-4), vm-bytes (256M)
vm	Memory	vm (1-10), vm-bytes (5%-50%)
iomix	Disk I/O	iomix (1-4), iomix-bytes (256M)

TABLE I: Stressors and parameters for profiling

2) Scheduler: The scheduler is implemented in Python and uses a Kubernetes client library to communicate with the API server. We use Python's Multiprocessing library to spawn multiple independent processes that cache monitoring data in memory to instantaneously access telemetry data required for inference and a scheduler loop that repeatedly takes new applications from the scheduler queue. Upon decision-making, the scheduler loop tells Kubernetes which host to start the application. The GNN is implemented in Python using PyTorch v2.0.1 [33] and uses the library PyTorch Geometric v2.3.1 [34] , which offers an implementation of the Heterogeneous Graph Transformer [32] network.

IV. Evaluation

Our evaluation consists of two parts: training and validating our GNN for power consumption and evaluating the energy-consumption reduction when employing our energy-aware scheduler. The cluster we used consists of three nodes, all equipped with dual-socket AMD 7443 CPUs. Two nodes were configured with lower TDP (Thermal Design Power), while one was configured for full TDP. Each node was also equipped with 256 GB of memory, and one node was also equipped with an Nvidia A100X GPU.

A. Training Dataset

To generate training data, we use different *stress-ng* stressors to stress certain parts of the system. This approach allows us to generate applications with different resource usage phases that mimic the behavior of real-world applications. Benchmark suites like FunctionBench [35] offer similar applications (e.g., matrix multiplication, disk I/O), but we want

[3]https://github.com/google/cadvisor

[4]https://github.com/prometheus/node_exporter

[5]https://github.com/NVIDIA/DCGM

[6]https://www.hpe.com/us/en/hpe-integrated-lights-out-ilo.html

[7]https://xilinx.github.io/XRT/master/html/xbutil.html

Benchmark	CPU	RAM	Parameters	Duration
LavaMD	4	4 GB	-cores 4 -boxes1d 50	∼100s
LavaMD	8	8 GB	-cores 8 -boxes1d 50	∼50s
Leukocyte	4	4 GB	40 40 testfile.avi	∼30s
Leukocyte	8	8 GB	50 40 testfile.avi	∼18s
srad_v1	4	4 GB	1000 0.5 5020 458 4	∼12s
srad_v1	8	8 GB	0.5 5020 458 4	∼10s

TABLE II: Rodinia applications for scheduler evaluation

to create a set of applications that focus on different resources so that a wide range of different resource usage is exposed. Table I shows each hardware resource's stressors, parameters, and value ranges. Each stressor runs for 100 seconds, and in total, there are 72 stress tests for each host. We split the dataset using a 5-fold approach and show the Root Mean Squared Error (RMSE) in the results for each run. The k-fold split is commonly used in machine learning [36]. This means we run 5 training-test splits, where one training dataset contains 4/5 of the complete dataset. The profiling data of different hosts is included in training and test sets. In the future, other resources, such as FPGA, should be stressed to train the model on a wider resource usage spectrum.

To train the GNN, we use *Adam Optimizer* with a *learning rate* of *1e-5*, 125 *epochs*, and *batch size* of 5.

B. Scheduler evaluation

Our scheduler evaluation consists of five different approaches: *GNN*, *GNN-Aware*, *GNN-Packing*, *Spreading* and *Packing*. *GNN*, *GNN-Aware*, and *GNN-Packing* are using the algorithms we presented using the power consumption predictions. The *Spreading* scheduling strategy balances containers, i.e., the CPU cores and memory used, as the default Kubernetes scheduler does. The *Packing* strategy always chooses the node with the highest number of CPU cores used. The number of CPU cores requested and used are pre-determined, and the applications are using the Rodinia Benchmark Suite [37]. All applications immediately run the computation after starting and report the results (i.e., execution duration) back for post-experiment analysis.

Table II shows all configurations, the requests, and limits for the number of cores and memory for Kubernetes resource allocation and the average execution duration. We define two scenarios based on a sinusoidal pattern. The short-running one submits 72 containers in 209 seconds, while the longer-running one submits 128 containers in 234 seconds.

V. Results

In this section, we present the results of our evaluation. We start by showing the results of the GNN training and, afterward, the scheduler evaluation results.

A. Graph Neural Network

The mean RMSE of our AI model from the 5-fold evaluation is around 7.5%, and the standard deviation is around 3%. These encouraging results can be improved further by considering a more complex network architecture and the inclusion of additional sensors. The selection of sensors was mainly motivated by the use case of implementing a scheduler based on the ability to perform *what-if* estimations to predict the impact of an application on power consumption.

We also see that the model can differentiate between similar systems observing different static power consumption. For example, one host in the cluster is equipped with an Nvidia A100X that increases the static power consumption by roughly 50 Watts (∼20% increase). The model can differentiate between these because we model the equipped GPU in the input.

In addition, we performed experiments with different GNN architectures, using LSTM-based approaches and different combinations of the presented model architecture (i.e., by varying the number of HGT layers). Our results indicate that there is a performance-accuracy trade-off to be made. Specifically, with an increasingly complex model, the accuracy increases but the performance suffers (i.e., inference speed decreases). We saw 2x changes in terms of time to predict but a marginal increase in accuracy (i.e., a 2% decrease of RMSE). Based on that, we chose the non-LSTM approach.

B. Energy-Aware Scheduler

We discuss the schedulers by showing the summarized results, looking at the power consumption over time, and investigating the overhead our scheduler imposes.

Table III and Table IV show the results for the short-running and long-running, respectively. The *Packing* scheduler achieved the lowest total energy consumption (Wh) and lowest EtS over all others in the short-running experiment. The *GNN-Packing* algorithm is close and only has slightly higher energy consumption, *i.e.,* by 1.6%. The Energy-to-Solution (EtS) metric shows the workload's total energy consumption [38]. Whereas the *GNN-Packing* was able to reduce total energy consumption and EtS in the long-running one. In all cases, the *Spreading* scheduler performed worse and has, in comparison to the *GNN-Packing* scheduler, an increase of 6.2% Wh and consumes 5.27% W per second more on average in the long-running one. In terms of makespan, the *Packing* scheduler performed worse, making our GNN-based approaches a viable solution to balance between the two goals. Interestingly, the performance, indicated by the *Mean Performance factor* that shows the increase to the average duration, is similar across the different approaches, even though the *Spreading* spread the workload across nodes. The reason for this is that the workload submits enough jobs to saturate all hosts, which makes the sharp difference in Watts (i.e., a decrease of ∼5%) very good.

1) Power Draw over time: Figure 2 shows the power over time for the long-running experiment setup. It is clearly visible that the *Packing* approach maintained a low power consumption. We also see a sharp increase of power consumption for the *Spreading* because it chooses all nodes equally and, therefore, will also pick nodes with higher power consumption which results in higher energy consumption. Our model can differentiate between nodes and pick the low-energy ones. While *Packing* did not select this node due to the implementation, our *GNN* could predict that the other nodes

Scheduler	Wh	Makespan (s)	EtS (J)	Mean Performance (factor)	Std. Performance (%)	Mean Power Consumption (W)
GNN	89.27±0.3	260±0.71	105206±331.63	**1.28±0.021**	22.2±1.03	403±1.55
GNN-Aware	90.05±0.11	259±0.71	104967±915.7	1.29±0.022	20.13±0.12	403±2.95
GNN-Packing	**88.84±0.39**	260±0.71	104250±782.06	1.31±0.01	20.72±0.45	**398±3.95**
Packing	87.44±0.39	261±2.83	102039±221.32	**1.32±0.004**	18.44±0.43	390±2.82
Spreading	**90.69±0.35**	251±2.12	106854±806.81	1.27±0.033	23.26±1.89	**424±0.69**

TABLE III: Results of individual schedulers for the short experiment

Scheduler	Wh	Makespan (s)	EtS (J)	Mean Performance (factor)	Std. Performance (%)	Mean Power Consumption (W)
GNN	141.69±1.07	336±1.41	169078±341.53	1.37±0.004	22.35±0.47	**502±2.83**
GNN-Aware	140.75±0.09	**332±2.83**	168130±1623.52	**1.34±0.008**	20.98±0.92	506±0.78
GNN-Packing	**139.17±0.79**	332±0.71	168039±439.82	1.36±0.016	23.78±0.61	504±3.24
Packing	141.14±1.29	**339±0.71**	168273±178.19	**1.39±0.012**	21.88±0.84	495±0.7
Spreading	**148.38±2.28**	335±2.12	178960±87.68	1.20±0.003	18.54±1.56	**532±3.25**

TABLE IV: Results of individual schedulers for long experiment

are more energy-efficient and select those on purpose. This circumstance, including the fact that *Packing* will have a very high performance degradation, due to the strategy of filling up nodes, proves that our algorithms can strike a good balance between *Packing* and *Spreading*.

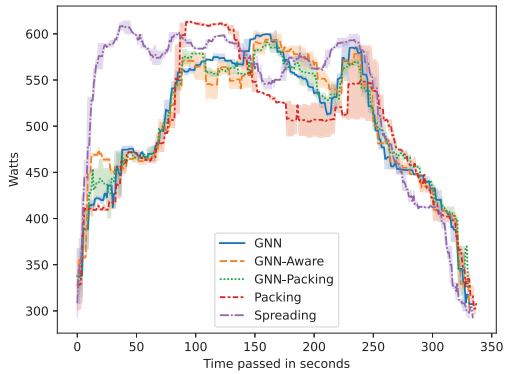

Fig. 2: Power over Time (long-running experiment)

2) Scheduling Overhead: The last aspect of our evaluation is the scheduling overhead caused by using the *GNN*. For this, we measured the time from preprocessing to postprocessing the selection algorithm. The mean overhead is around 250ms which can be improved upon by using GPUs for inference. It also shows the potential of our approach because we were still able to reduce makespan and energy consumption.

VI. Future Work and Limitations

An important task to alleviate this proof of concept to a production-ready solution is considering more edge cases than we covered. Specifically, what should the scheduler do in case of the arrival of new applications? If the application has never been run on any host, we must resort to a default scheduling strategy (i.e., *Packing*). Once the application has been run on a node, we can use its isolated monitoring from cAdvisor. This entails implementing continuous learning using MLOps [39] to fine-tune the system during run time. While our model learns the power consumption of the host instead of on a per-application base, we still need to incorporate newly available data. Lastly, to become a viable production system scheduler, scalability, and energy efficiency experiments have to be made to guarantee energy savings concerning the scheduling overhead. We also think the GNN can facilitate the creation of a clustering method to identify similar workloads [40]. This way, the GNN-based scheduler can take historical data of similar workloads for its decision. Moreover, resource interference-aware scheduling methods can be used by taking advantage of this clustering approach by avoiding placing applications from the same cluster together. Reducing energy consumption is not an exclusive goal of cloud data centers but also plays a vital role in the edge-cloud continuum where devices might be battery-powered [41]. The ability to support heterogeneous devices is, in this context, even more important. Our initial graph model can be extended with domain-expert knowledge and models of complex relationships between components to increase its accuracy. While in this work, we only model hardware accelerators but did not run profiling experiments, we plan to run more Rodinia benchmarks [42], [43] on AMD FPGAs and GPUs to extend the applicability of our model.

VII. Conclusion

This work introduces a novel model to represent hosts and host components in a data center as graphs. Atop the graph, we implement and train a Graph Neural Network to predict the power consumption over a specific time frame. The model's mean RMSE is $\approx 7.5\%$ and can facilitate the development of energy-aware data center management strategies differently. We showcase the feasibility of using our GNN in scheduling containers, where we can reduce energy consumption by an average of 6.2% while also completing the workload faster than a scheduler working like the default Kubernetes scheduler. However, improving energy efficiency is only a partial solution as it has become clear that we should focus on reducing carbon emissions as well. Therefore, deploying only energy-aware strategies might not be enough, and carbon-aware ones are required. Our approach to predicting energy consumption can facilitate building new carbon-aware strategies.

REFERENCES

[1] A. S. Andrae and T. Edler, "On global electricity usage of communication technology: trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

[2] M. Avgerinou, P. Bertoldi, and L. Castellazzi, "Trends in data centre energy consumption under the european code of conduct for data centre energy efficiency," *Energies*, vol. 10, no. 10, 2017. [Online]. Available: https://www.mdpi.com/1996-1073/10/10/1470

[3] P. Patros, J. Spillner, A. V. Papadopoulos, B. Varghese, O. Rana, and S. Dustdar, "Toward sustainable serverless computing," *IEEE Internet Computing*, vol. 25, no. 6, pp. 42–50, 2021.

[4] P. Raith, S. Nastic, and S. Dustdar, "Serverless edge computing—where we are and what lies ahead," *IEEE Internet Computing*, vol. 27, no. 3, pp. 50–64, 2023.

[5] G. Rattihalli, M. Govindaraju, H. Lu, and D. Tiwari, "Exploring potential for non-disruptive vertical auto scaling and resource estimation in kubernetes," *IEEE International Conference on Cloud Computing, CLOUD*, vol. 2019-July, pp. 33–40, 7 2019.

[6] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari, "Mashup: making serverless computing useful for hpc workflows via hybrid execution," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2022, pp. 46–60.

[7] H. Wang, D. Niu, and B. Li, "Distributed machine learning with a serverless architecture," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1288–1296.

[8] S. Nastic, P. Raith, A. Furutanpey, T. Pusztai, and S. Dustdar, "A serverless computing fabric for edge & cloud," in *2022 IEEE 4th International Conference on Cognitive Machine Intelligence (CogMI)*. IEEE, 2022, pp. 1–12.

[9] P. Czarnul, J. Proficz, A. Krzywaniak *et al.*, "Energy-aware high-performance computing: survey of state-of-the-art tools, techniques, and environments," *Scientific Programming*, vol. 2019, 2019.

[10] G. Rattihalli, N. Hogade, A. Dhakal, E. Frachtenberg, R. P. Hong Enriquex, P. Bruel, A. Mishra, and D. Milojicic, "Fine-grained heterogeneous execution framework with energy aware scheduling," *IEEE International Conference on Cloud Computing*, 2023.

[11] M. S. Aslanpour, A. N. Toosi, M. A. Cheema, and R. Gaire, "Energy-aware resource scheduling for serverless edge computing," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 190–199.

[12] R. Caspart, S. Ziegler, A. Weyrauch, H. Obermaier, S. Raffeiner, L. P. Schuhmacher, J. Scholtyssek, D. Trofimova, M. Nolden, I. Reinartz *et al.*, "Precise energy consumption measurements of heterogeneous artificial intelligence workloads," in *International Conference on High Performance Computing*. Springer, 2022, pp. 108–121.

[13] S. H. Rastegar, H. Shafiei, and A. Khonsari, "Enex: An energy-aware execution scheduler for serverless computing," *IEEE Transactions on Industrial Informatics*, 2023.

[14] S. MirhoseiniNejad, H. Moazamigoodarzi, G. Badawy, and D. G. Down, "Joint data center cooling and workload management: A thermal-aware approach," *Future Generation Computer Systems*, vol. 104, pp. 174–186, 2020.

[15] S. Bharany, S. Sharma, O. I. Khalaf, G. M. Abdulsahib, A. S. Al Humaimeedy, T. H. H. Aldhyani, M. Maashi, and H. Alkahtani, "A systematic survey on energy-efficient techniques in sustainable cloud computing," *Sustainability*, vol. 14, no. 10, 2022. [Online]. Available: https://www.mdpi.com/2071-1050/14/10/6256

[16] Y. Fan, Z. Lan, T. Childers, P. Rich, W. Allcock, and M. E. Papka, "Deep reinforcement agent for scheduling in hpc," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 807–816.

[17] Y. Fan, Z. Lan, P. Rich, W. Allcock, and M. E. Papka, "Hybrid workload scheduling on hpc systems," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 470–480.

[18] D. Nichols, A. Marathe, K. Shoga, T. Gamblin, and A. Bhatele, "Resource utilization aware job scheduling to mitigate performance variability," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 335–345.

[19] A. Tzenetopoulos, C. Marantos, G. Gavrielides, S. Xydis, and D. Soudris, "Fade: Faas-inspired application decomposition and energy-aware function placement on the edge," in *Proceedings of the 24th International Workshop on Software and Compilers for Embedded Systems*, 2021, pp. 7–10.

[20] P. Sharma, "Challenges and opportunities in sustainable serverless computing," 2022.

[21] D. C. Wilson, S. Jana, A. Marathe, S. Brink, C. M. Cantalupo, D. R. Guttman, B. Geltz, L. H. Lawson, A. H. Al-rawi, A. Mohammad, F. Keceli, F. Ardanaz, J. M. Eastep, and A. K. Coskun, "Introducing application awareness into a unified power management stack," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 320–329.

[22] W. A. Hanafy, Q. Liang, N. Bashir, D. Irwin, and P. Shenoy, "Carbonscaler: Leveraging cloud workload elasticity for optimizing carbon-efficiency," *arXiv preprint arXiv:2302.08681*, 2023.

[23] D. Ou, C. Jiang, M. Zheng, and Y. Ren, "Container power consumption prediction based on gbrt-pl for edge servers in smart city," *IEEE Internet of Things Journal*, 2023.

[24] X. Zhang, J.-J. Lu, X. Qin, and X.-N. Zhao, "A high-level energy consumption model for heterogeneous data centers," *Simulation Modelling Practice and Theory*, vol. 39, pp. 41–55, 2013.

[25] K. M. U. Ahmed, M. H. Bollen, and M. Alvarez, "A review of data centers energy consumption and reliability modeling," *IEEE Access*, vol. 9, pp. 152 536–152 563, 2021.

[26] 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.

[27] Z. Zhou, J. H. Abawajy, F. Li, Z. Hu, M. U. Chowdhury, A. Alelaiwi, and K. Li, "Fine-grained energy consumption model of servers based on task characteristics in cloud data center," *IEEE access*, vol. 6, pp. 27 080–27 090, 2017.

[28] T. Khan, W. Tian, S. Ilager, and R. Buyya, "Workload forecasting and energy state estimation in cloud data centres: Ml-centric approach," *Future Generation Computer Systems*, vol. 128, pp. 320–332, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X21004155

[29] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[30] S. Tuli, S. S. Gill, M. Xu, P. Garraghan, R. Bahsoon, S. Dustdar, R. Sakellariou, O. Rana, R. Buyya, G. Casale *et al.*, "Hunter: Ai based holistic resource management for sustainable cloud computing," *Journal of Systems and Software*, vol. 184, p. 111124, 2022.

[31] D. S. H. Tam, Y. Liu, H. Xu, S. Xie, and W. C. Lau, "Pert-gnn: Latency prediction for microservice-based cloud-native applications via graph neural networks," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2155–2165. [Online]. Available: https://doi.org/10.1145/3580305.3599465

[32] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *Proceedings of The Web Conference 2020*, ser. WWW '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2704–2710. [Online]. Available: https://doi.org/10.1145/3366423.3380027

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[34] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[35] J. Kim and K. Lee, "Functionbench: A suite of workloads for serverless cloud function service," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 502–504.

[36] T.-T. Wong and P.-Y. Yeh, "Reliable accuracy estimates from k-fold cross validation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1586–1594, 2020.

[37] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[38] T. Wilde, A. Auweter, M. K. Patterson, H. Shoukourian, H. Huber, A. Bode, D. Labrenz, and C. Cavazzoni, "Dwpe, a new data center

energy-efficiency metric bridging the gap between infrastructure and workload," in *2014 International Conference on High Performance Computing & Simulation (HPCS)*, 2014, pp. 893–901.

[39] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (mlops): Overview, definition, and architecture," *IEEE Access*, vol. 11, pp. 31 866–31 879, 2023.

[40] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller, "Graph clustering with graph neural networks," *Journal of Machine Learning Research*, vol. 24, no. 127, pp. 1–21, 2023. [Online]. Available: http://jmlr.org/papers/v24/20-998.html

[41] G. R. Russo, V. Cardellini, and F. L. Presti, "Serverless functions in the cloud-edge continuum: Challenges and opportunities," in *2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2023, pp. 321–328.

[42] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, ser. IISWC '09. USA: IEEE Computer Society, 2009, p. 44–54. [Online]. Available: https://doi.org/10.1109/IISWC.2009.5306797

[43] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang, "Understanding performance differences of fpgas and gpus," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 93–96.