EELAS: Energy Efficient and Latency Aware Scheduling of Cloud-Native ML Workloads

Ilias Syrigos*, Dimitris Kefalas*, Nikos Makris*† and Thanasis Korakis*†
*Dept. of Electrical and Computer Engineering, University of Thessaly, Greece
†Centre for Research and Technology Hellas, CERTH, Greece
Email: ilsirigo@uth.gr, dkefalas@uth.gr, nimakris@uth.gr, korakis@uth.gr

2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS) | 978-1-6654-7706-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/COMSNETS56262.2023.10041344

***Abstract*—The widespread use of microservices and the use of cloud-native methodologies for the deployment of services have increased the service providers' flexibility and management efficiency. As the available resources for scheduling such workloads have extended the boundaries of traditional datacenters to the fog, edge, and beyond-edge, the scheduling of challenging workloads must also account for energy efficiency, as these devices are typically battery-powered and resource-constrained, while maintaining acceptable performance. Specifically for ML inference workloads, provisioning and access latency plays a crucial role in their successful operation. Towards combating these issues, we design, develop, and evaluate our platform for Energy Efficient Latency-Aware Scheduling (EELAS) of workloads. First, we formulate the scheduling problem as an ILP problem, and then we develop a less complex heuristic method that allows the efficient allocation of resources within the continuum. Our EELAS prototype integrates with Kubernetes and is capable of reducing the overall energy consumption of cloud-to-things resources while accounting for latency of ML workloads. Our evaluation in real-world settings reveals significant energy gains for scheduling ML inference tasks, also reachable with the minimum possible latency from far-edge devices.**

***Index Terms*—cloud-to-things continuum, energy efficiency, latency aware, workload scheduling, testbed**

I. Introduction

Network Functions Virtualization (NFV) has emerged as a key solution for facilitating the deployment of workloads on any kind of underlying hardware. By decoupling the functions of the workloads from the hardware resources, and executing them only in software by making extended use of Virtual Machines or the emerging microservices (e.g. dockers, containers, Unikernels, etc.), the network edges including far edge and fog/mist computing resources can be leveraged to their full potential. Nevertheless, the complexity and resource consumption of the deployed workloads is constantly rising as they incorporate additional technological capabilities, such as the use of Machine Learning (ML). This, in turn, provides a complex environment for optimal deployment, as devices closer to the network's edge are resource-constrained and reliant on finite batteries to operate. Therefore, scheduling such resource-intensive workloads across the entire cloud-to-things continuum becomes a significant challenge.

The research leading to these results has received funding from the European Horizon 2020 Programme for research, technological development and demonstration under Grant Agreement Number No 101008468 (H2020 SLICES-SC). The European Union and its agencies are not liable or otherwise responsible for the contents of this document; its content reflects the view of its authors only.

As cloud-native deployment of services and workloads becomes the norm, several orchestrators that manage the underlying networking fabric and deploy the workloads on top have emerged. Such solutions include the widely adopted Kubernetes (K8s) framework for the core cloud and the edge, as well as the Rancher K3s platform for resource-constrained edge and beyond edge devices [1]. As a result, none of the existing solutions are able to simultaneously cover deployments over the entire cloud-to-things continuum, taking into consideration aspects such as energy efficiency of the edge/far-edge devices, communication latency, task execution delay, as well as requested resources from the different devices.

In this paper, we tackle this problem, by appropriately developing extensions to the scheduler of the core Kubernetes framework, towards supporting edge and far-edge devices [2]. Through our extensions, we are able to maximize the overall continuum energy efficiency for deployments of demanding applications, by taking into consideration aspects such as the latency for the deployed services, their location, and their execution time - subject to the underlying host resources that are different as we move from the far-edge to the core cloud. The framework is able to deliver *Energy Efficient and Latency-Aware Scheduling (EELAS)* within the resource continuum. Our solution is tailored especially for the demanding ML inference workloads, and involves continuous monitoring of the available continuum resources, before concluding on the allocation of resources. The solution is provided as a fully-fledged framework, integrated in the K8s environment, and extending existing APIs for orchestration. The main contributions of our work are the following:

- To formulate the allocation problem and provide a heuristic solution.
- To minimize the energy consumption of the cloud-to-things continuum when deploying workloads.
- To maximize the efficiency of ML inference tasks, subject to their latency from the decision making process.
- To develop and evaluate our solution in a fully-fledged framework over a real testbed setup.

We bundle our solution in a real prototype, and evaluate its efficiency over NITOS, the Greek Node of the SLICES-RI [3].

The rest of the paper is organized as follows: Section II provides an overview of the related literature. Section III provides our overall system model, the scheduling problem formulation, as well as our proposed heuristic algorithm. In Section IV we evaluate our solution, and in Section V we conclude our work and present some future directions.

II. Related Work

Service orchestration and scheduling have received a lot of attention since the wide application of NFV architectures. Given the different constraints that the deployed applications and services need to meet, scheduling is not a trivial task, as it must simultaneously fulfill the needs of the infrastructure provider while not violating contracted Service Level Agreements (SLAs) with the end-users. Based on the fact that the CO2 footprint of datacenters that host such workloads is not negligible, the community has steered its efforts towards energy efficient scheduling. In [4], a classification of the different scheduling algorithms is provided, with a large portion of them being energy efficient methods. Authors in [5] consider solar-powered devices and schedule workloads with the ultimate goal of minimizing the usage of brown energy (the energy of cloud servers) as much as possible. In [6], authors design and evaluate workload schedulers for finding efficient mappings of workflows into resources in order to maximize the quality of service while reducing the energy required to compute them.

As IoT devices are constantly spreading, this creates fertile space for the deployment of services and workloads across a wider resource continuum, as devices are present at the fog and edge levels. In [7], the authors consider a continuum of resources that spans the fog and core cloud. They propose a tree based model for scheduling workloads to the fog nodes and the core cloud, in order to reduce the overall energy consumption. The authors in [8] propose an energy-efficient resource allocation algorithm for the fog computing environment. They use ordered lists to list available resources and schedule the workloads to the least congested, achieving higher efficiency and balancing the load. In [9] authors consider the edge as an environment complementary to the fog and cloud and in real-time enhance the scheduling of data and workloads across the different locations.

As in other fields [10], ML application can prove highly beneficial for the scheduling decisions. Therefore, several works either integrate ML in the scheduling process, or develop energy efficient schedulers for ML tasks, or employ both approaches concurrently. In [11], authors develop their scheme for scheduling ML workloads to different servers with the goal of enhancing the accuracy achieved by the ML tasks. In order to accomplish this, they employ a ML scheme for the scheduler that uses reinforcement learning for updating the scheduler model during operation, and be able to achieve lower execution times and greater accuracy for their workloads. Authors in [12] use a similar approach for making their ML-based scheduler aware of ML workloads that need to be deployed and be able to be retrained during run-time with a reinforcement learning approach. Finally, in [13], authors develop a scheduler aware of possible bottlenecks in the execution of ML workloads, eventually able to schedule/migrate tasks in order to overcome limitations during training.

In this work, we progress beyond existing literature by developing a scheduling algorithm for ML inference tasks that is energy-efficient and latency-aware. This is one of the first attempts in literature to jointly address the scheduling problem, while considering that the inference tasks need to usually be executed in the most efficient (quickly and energy efficient) manner and as closer to the edge as possible. In the next section, we formulate our problem and the relevant heuristics for reaching the optimal solution.

III. System Model

In this section, we define an optimization problem for the energy-efficient scheduling of workloads on heterogeneous cloud, edge, and fog devices in a cloud-to-things continuum cluster, and then formulate it as an Integer Linear Program (ILP). We also present a scalable, energy-efficient scheduling algorithm based on a heuristic approach that reduces complexity and scheduling time.

A. Problem Definition

We describe the problem of workload scheduling in cloud-to-things continuum resources as follows: Given is a set of resource nodes scattered across the continuum. We assume that the primary factor influencing the power consumption of such a node is the CPU power consumption, which is directly proportional to the load of tasks that are actually executing on the node. As such, we do not consider the power consumption of other components, e.g., memory, storage, and fans. We are provided with the idle and maximum power consumption in Watts per CPU core for each node's CPU module. We are also provided with the Million Instructions Per Second (MIPS) of a CPU core that can be derived through standard benchmarking techniques. Furthermore, we are provided a set of workloads to schedule on the continuum resources, with each workload requiring a minimum QoS in terms of latency. This latency refers to the time needed for the deployed workload, which we consider to be a Machine Learning model, to respond with an inference to a request from a user on a specific location. This time is specified as the execution time of the ML model for a given input, plus the Round Trip Time (RTT) for the request or response to be transferred from or to the user. We assume that the ML workload can be fully parallelized and utilize several CPU cores, and that we are provided with the total number of instructions involved in the execution of the ML workload, which, for simplicity, is independent of the CPU architecture and only depends on processing power.

The objective of the problem is to minimize the total energy consumed by the nodes in the cluster. Because many fog devices are battery-powered, this translates to longer battery life and thus increased resource availability. However, we are constrained by satisfying the user's requirement for a fast inference response with a minimum acceptable latency. So, scheduling should also take latency into account, not just in terms of network latency between the user and the node where the workload is deployed, but also in terms of how long it takes an ML model to run on a certain node.

B. Problem Formulation

The definition of the proposed ILP formulation is presented as follows:

Fig. 1: The overall EELAS architecture

Sets:

N	Set of nodes in the cloud-to-things continuum
W	Set of workloads to be deployed
C_n	Set of CPU cores of node $n \in N$
L	Set of locations

Parameters:

P_n^{idle}	Idle power consumption of a CPU core of a node $n \in N$
P_n^{max}	Maximum power consumption of a CPU core of a node $n \in N$
U_{cn}	Utilization of a CPU core $c \in C_n$ of a node $n \in N$ in the range of 0 to 1
IPS_n	Million Instructions Per Second of a CPU core of a node $n \in N$
I_w	Total number of Instructions involved in the execution of a workload $w \in W$
Lat_w	Maximum user-required inference response latency for a workload $w \in W$
$NetLat_{nl}$	Network latency (RTT) between node $n \in N$ and location $l \in L$

Variables:

x_{wn}	Binary variable that equals one if workload $w \in W$ is deployed in node $n \in N$, zero otherwise

The objective is to minimize the total energy consumption in the cluster, given a set of nodes N and workloads to be deployed W.

Objective: Minimize

$$\min \sum_{\mathrm{n} \in \mathrm{N}} E_n \tag{1}$$

The above objective function is subject to the following constraints:

$$D_{wn,t} \leq Lat_w, \tag{2}$$

where the inference response delay $D_{wn,t}$ of the workload w deployed on a node n at any time slot t must be less or equal to a minimum user-defined QoS latency value. The inference response delay is given

$$D_{wn,t} = T_{wn,t} + NetLat_{nl} \quad \forall \mathrm{w} \in \mathrm{W}, \mathrm{n} \in \mathrm{N}, \mathrm{l} \in \mathrm{L}, \tag{3}$$

that equals to the delay from the execution of the ML model w on node n at time slot t, $T_{wn,}$, and the network latency (RTT) between n and the location l of the user, $NetLat_{nl}$.

$$\sum_{\mathrm{n} \in \mathrm{N}} x_{wn} \leq 1 \quad \forall \mathrm{w} \in \mathrm{W} \tag{4}$$

$$\sum_{\mathrm{w} \in \mathrm{W}} x_{wn} \leq W_{max} \quad \forall \mathrm{n} \in \mathrm{N} \tag{5}$$

Constraint (4) limits the amount of nodes on which a workload can be deployed to just one, as we do not consider replication or slicing of the workload. Inequality in the constraint denotes the case where the workload is rejected as it cannot satisfy the constraints, and the sum of x_{wn} equals zero. Additionally, constraint (5) limits the amount of workloads that can be deployed on a single node to the maximum value W_{max}. In the case of a K8s system, the parameter W_{max} is equal to 110 workloads (pods) per node.

The execution time of a workload w on node n at the time slot t is provided by

$$T_{wn,t} = x_{wn} \frac{I_w}{Sp_{n,t}} \quad \forall \mathrm{w} \in \mathrm{W}, \mathrm{n} \in \mathrm{N}, \tag{6}$$

where $Sp_{n,t}$ is the speed of the CPU on the node n at the same time slot, and which is defined as

$$Sp_{n,t} = |C_n|IPS_n - \quad (7)$$
$$(\sum_{\mathrm{c} \in \mathrm{C}_n} U_{cn,t} IPS_n + \sum_{\mathrm{w} \in \mathrm{W}} x_{wn} \frac{I_w}{|C_n|IPS_n}) \quad \forall \mathrm{n} \in \mathrm{N},$$

where the first term refers to the total computing capacity of the CPU on the node n given the number of cores $|C_n|$ and the second to the utilization of cores by already-deployed workloads plus the estimation for the utilization of the ones to be deployed.

The energy consumption of a node n after the scheduling and execution of the workloads W, can be derived by

$$E_n = \sum_{\mathrm{w} \in \mathrm{W}} x_{wn} \sum_{t=0}^{T_{wn}} P_{wn,t}^{cpu} \quad \forall \mathrm{n} \in \mathrm{N}, \quad (8)$$

that corresponds to the sum of the node's CPU power consumption for the execution of each workload w deployed on that node. The consumption of the CPU of n for executing workload w at the time slot t is given by

$$P_{wn,t}^{cpu} = \sum_{\mathrm{c} \in \mathrm{C}_n} P_{wcn,t}^{core} \quad \forall \mathrm{w} \in \mathrm{W}, \mathrm{n} \in \mathrm{N}, \quad (9)$$

and corresponds to the total power consumption of the CPU cores for the given workload w on node n. This, in turn, is defined as

$$P_{wcn,t}^{core} = (P_n^{max} - P_n^{idle})(U_{cn,t} + \frac{I_w}{IPS_n}) + P_n^{idle} \quad (10)$$
$$\forall \mathrm{w} \in \mathrm{W}, \mathrm{c} \in \mathrm{C}_n, \mathrm{n} \in \mathrm{N},$$

and equals to the amount of power consumed by a CPU core c given its utilization by already-deployed workloads at time slot t plus the estimated utilization by workload w. Even when a CPU core is not in use, it consumes power equal to P_n^{idle}.

As it is evident, this problem is not solved in deterministic time in large systems, due to its high-complexity and a heuristic approach is necessary for providing a close-to-optimal solution in a reasonable time.

C. Heuristic Approach

In this section, we present our algorithm for energy-efficient scheduling with latency constraints, which is based on a greedy heuristic approach [14]. This algorithm was developed in order to achieve better scheduling times in large-scale clusters of nodes, as it is quicker and has less complexity as compared with the ILP. Each iteration begins with a random allocation, and then attempts to improve the energy efficiency by adding and removing nodes from the allocation until a local minimum value is reached for that iteration. The iterations end when there is no longer an improvement to the global minimum value, which is the minimum of all iterations. Before proceeding with the description of the algorithm, we define a metric for the approximate estimation of the energy efficiency of each node, $EnEffIndex_n$, expressed in units of MIPS/Watt.

$$EnEffIndex_n = \frac{IPS_n}{P_n^{max}} \quad \forall \mathrm{n} \in \mathrm{N} \quad (11)$$

Algorithm 1 is the pseudocode of the proposed heuristic scheduler, Energy Efficient and Latency Aware Scheduler (EELAS), that allocates resources for a batch of workloads W across a set of cloud-to-things continuum nodes N. It takes as inputs W and N and sorts the set of workloads W in descending order of the number of instructions required for the execution of each workload, I_w. It is worth noting here that we are assuming that these values are derived from the prior application of benchmarking techniques to each workload. Our rationale behind the sorting is that the workloads with longest execution times, and thus more complexity should be scheduled first, as the shortest ones have a smaller contribution towards the energy efficiency of the system and are therefore easier to schedule. The outer **while** loop in Line 2 executes until there is no improvement in the quality of the solution after k_{max} iterations. An initial random set of nodes S consisting of as many nodes as the number of workloads to be scheduled or the maximum number of nodes in the cluster is generated. In Line 4, we assign workloads to the set of nodes S and obtain a value of energy consumption for this assignment by calling the function *EnergyAssign*.

This function's pseudocode is described by the Algorithm 2. It takes as inputs the sets of workloads W and the set of nodes S, with the latter sorted in descending order of $EnEffIndex$. Then, for each workload w, a node s is assigned, beginning from the most energy efficient, if the inference response delay is less or equal to Lw. Then, the energy consumed value for this node s is updated by adding the estimated energy consumption caused by the execution of the workload w. If w cannot be assigned to any of the nodes, a flag A_w indicating the failure in allocation is returned to the calling function. When all workloads have been assigned to a node, the total energy consumption is then determined and returned.

The EELAS algorithm receives the energy consumption value and keeps it as the minimum current value. It also receives a vector A representing the success or failure of workloads' assignment. Then, within the **repeat-until** loop it updates S by adding and removing nodes from the set of cluster nodes N until it reaches a local energy minimum (i.e. until E_{min} can no longer be decreased) that is stored in E'_{min}. If E_{min} is less than the global energy minimum $Gbal_{min}$, then $Gbal_{min}$ is set to E_{min} and x_{wn} stores the assignment of workloads to nodes, that achieves this minimum. Then a new random set S is generated and the outer **while** loop repeats.

IV. Evaluation

For the purpose of evaluating EELAS, we have implemented a prototype of our framework and deployed it on a realistic testbed comprising nodes distributed along: 1) the edge, 2) the fog, and 3) the cloud. EELAS is the orchestrator that schedules workloads i.e., ML models for providing inference to users.

In the following part, we describe our experimental setup and prototype's technical information, and in subsection IV-B, we show our experimental findings.

A. Experimental Setup

To evaluate our approach, we employ the NITOS testbed [15], part of the SLICES-RI [3]. As illustrated in Figure 1, we

Algorithm 1 Energy Efficient and Latency Aware Scheduler

procedure EELAS(W, N)
  while $k < k_{max}$ **do**
    Generate Random S
    $E_{sol}, A_W \leftarrow EnergyAssign(W, S)$
    $E_{min} \leftarrow E_{sol}$
    repeat
      $E'_{min} \leftarrow E_{min}$
      for $n \in (N - S)$ **do**
        Add n to S
        $E_{sol}, A_W \leftarrow EnergyAssign(W, S)$
        if $E_{sol} < E_{min}$ **and** $A_W = 1$ **then**
          $E_{min} \leftarrow E_{sol}$
        else
          Remove n from S
        end if
      end for
      for $s \in S$ **do**
        Remove s from S
        $E_{sol}, A_W \leftarrow EnergyAssign(W, S)$
        if $E_{sol} < E_{min}$ **and** $A_W = 1$ **then**
          $E_{min} \leftarrow E_{sol}$
        else
          Add s to S
        end if
      end for
      if $E_{min} \geq Gbal_{min}$ **then**
        $k \leftarrow k + 1$
      else
        $k \leftarrow 0$
        $Gbal_{min} \leftarrow E_{min}$
      end if
    until $E_{min} \geq E'_{min}$
  end while
end procedure

Algorithm 2 Energy Consumption Aware Assignment

procedure ENERGYASSIGN(W, S)
  Sort S by $EnEffIndex$
  for $w \in W$ **do** ▷ Outter
    for $s \in S$ **do** ▷ Inner
      if $D_{ws} \leq Lw$ **then**
        Assign w to s
        $E_s \leftarrow E_s + \sum_{t=0}^{T_{ws}} P_{ws,t}^{cpu}$
        break ▷ Both loops
      end if
    end for
    if w not assigned **then**
      $A_w \leftarrow 0$
    end if
  end for
  for $s \in S$ **do**
    $E_{sol} \leftarrow \sum_{\mathrm{n} \in \mathrm{N}} E_n$
  end for
end procedure

employ 3 separate nodes to represent the various nodes that exist along the continuum: a resource-constrained, edge-based device, a compute node for the fog network, and a piece of the cloud infrastructure for the cloud configuration. We setup the nodes in a K8s (version 1.18.18) cluster, which is extended and configured to also include the edge device (Raspberry Pi 4b) and adjust the delays between the nodes to mimic a real network configuration (the RTTs of user devices to edge are $\leq 5ms$, to fog $\leq 20ms$, to cloud $\geq 20ms$). Our framework was implemented by extending the default K8s scheduler with a priority endpoint so that our heuristic approach could be executed during workload scheduling. EELAS retrieves the values of latency requirements, location, and current workload instructions from the K8s YAML configuration files using labels as described in [16].

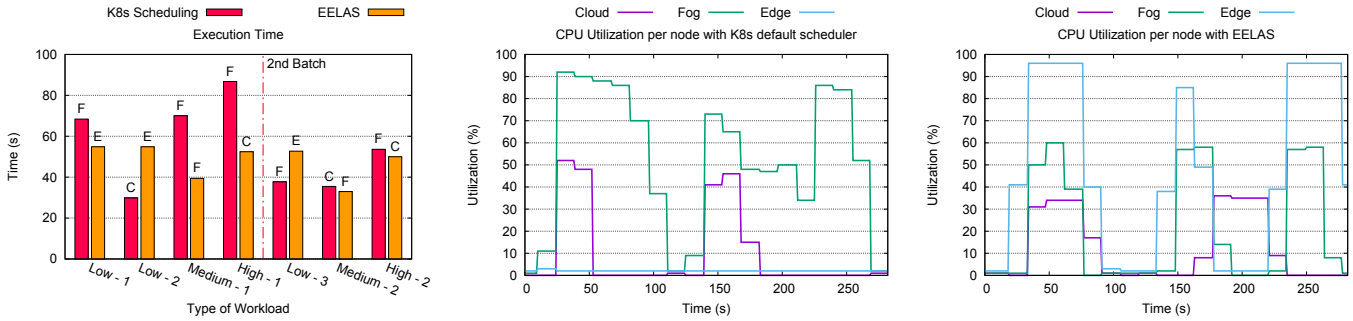
We use an inference task of object detection from a video file, utilizing OpenVino [17] (version 2020.3) and OpenCV [18] (version 4.6.0) as the ML workload, and we differentiate between the workloads by altering the resolution, and thus the size of the video that we feed as input to the ML model. The execution times on different sorts of nodes for the three video resolutions low (480p), medium (720p), and high (1080p) are listed in the Table II.

TABLE I: Hardware specification

	CPU Model	Cores	RAM	P^{max}	P^{idle}
Cloud	Intel Xeon E-2176g	32	64 GB	114 W	33 W
Fog	Intel Xeon Silver 4215	8	32 GB	85 W	27 W
Edge	Raspberry Pi 4b	4	2 GB	6 W	4 W

B. Experiment Evaluation

In order to evaluate the scheduling decisions and consequently the performance of EELAS, we compare our implementation to K8s's default scheduler. We schedule two batches of workloads in the order shown in Figure 2a. The letters above the bars denote the node on which each workload was scheduled: C for cloud, F for fog, and E for edge. K8s scheduler performs best effort scheduling and places the workloads on the fog and cloud nodes. As a result, workload "High-1" of the first batch takes longer to execute due to the increased utilization on that node. This is also evident by the execution time of the similar "High-2" workload of the second batch, deployed on the same node, which is executed in 40% less time. EELAS scheduler prefers to schedule low resolution workloads on the edge node, medium on the fog node, and high on the cloud node. As a result, we observe that execution times in the first batch are lower on average compared to K8s scheduler as workloads are spread across the continuum cluster. Figures 2b, 2c depict the utilization of the nodes' CPUs during the experiment for the K8s and EELAS schedulers, respectively. We can notice that the resource-constrained Raspberry Pi reaches high utilization values even for low-resolution workloads. Obviously, the fog node also exhibits high utilization values with the default K8s scheduling as most of the workloads are deployed there. Using these



(a) Execution time and allocations (b) Default K8s scheduling (c) EELAS scheduling

Fig. 2: CPU Utilization per node (Edge/Fog/Cloud) and allocation for the inference workloads

utilization values and the power consumption of each node's CPU from Table I, we can obtain the total energy consumed across the cluster using Equation (8). Results show that the deployment of workloads using the EELAS scheduler achieves an impressive 41.8% reduction in energy consumption compared to the default K8s scheduler. Moreover, the allocation of workloads to different nodes allows them to finish their execution in less time, thus improving the throughput for inference requests from the users, as the network on the host nodes is usually shared between all the different workloads.

TABLE II: Execution time for each workload on different nodes

	Edge	**Fog**	**Cloud**
480p	29.3s	23.2s	19.6s
720p	40.3s	33.7s	30.2s
1080p	59.7s	52.1s	47.7s

V. Conclusion

In this work, we designed, developed, and evaluated our platform for Energy Efficient Latency-Aware Scheduling (EELAS) of ML inference workloads. The problem of allocating the workloads in the cloud-to-things resource continuum was formulated as an ILP, and our developed, less complex heuristic algorithm allows us to find a near-optimal solution in less time. The designed framework was materialized in a testbed environment and was able to achieve an overall energy efficiency of over 41.8% compared to the off-the-shelf K8s scheduler. In the future, we foresee extending this work to also cover the training tasks for the cases of federated learning within the resource continuum, as well as deciding on possible migrations of workloads in order to achieve higher efficiency, as well as scaling of the deployed workloads. Moreover, we foresee extending this work with the use of ML learning methods for addressing dynamic profiling of the workloads that are imminent to be deployed when submitted to the EELAS scheduler.

References

[1] A. Valantasis, N. Makris, and T. Korakis, "Orchestration Software for Resource Constrained Datacenters: an Experimental Evaluation," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, 2022, pp. 121–126.

[2] Y. Wu, "Cloud-Edge Orchestration for the Internet of Things: Architecture and AI-Powered Data Processing," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12 792–12 805, 2021.

[3] S. Fdida *et al.*, "SLICES, a scientific instrument for the networking community," *Computer Communications*, vol. 193, pp. 189–203, 2022.

[4] G. L. Stavrinides and H. D. Karatza, "Scheduling data-intensive workloads in large-scale distributed systems: trends and challenges," *Modeling and simulation in HPC and cloud systems*, pp. 19–43, 2018.

[5] I. De Courchelle *et al.*, "Green energy efficient scheduling management," *Simulation Modelling Practice and Theory*, vol. 93, pp. 208–232, 2019, modeling and Simulation of Cloud Computing and Big Data.

[6] B. Dorronsoro *et al.*, "A hierarchical approach for energy-efficient scheduling of large workloads in multicore distributed systems," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 4, pp. 252–261, 2014, special Issue on Energy Aware Resource Management and Scheduling (EARMS).

[7] "An energy-efficient model for fog computing in the Internet of Things (IoT)," *Internet of Things*, vol. 1-2, pp. 14–26, 2018.

[8] A. U. Rehman *et al.*, "Dynamic Energy Efficient Resource Allocation Strategy for Load Balancing in Fog Environment," *IEEE Access*, vol. 8, 2020.

[9] H. A. Alharbi and M. Aldossary, "Energy-Efficient Edge-Fog-Cloud Architecture for IoT-Based Smart Agriculture Environment," *IEEE Access*, vol. 9, 2021.

[10] I. Syrigos, N. Sakellariou, S. Keranidis, and T. Korakis, "On the Employment of Machine Learning Techniques for Troubleshooting WiFi Networks," in *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2019, pp. 1–6.

[11] H. Wang, Z. Liu, and H. Shen, "Job Scheduling for Large-Scale Machine Learning Clusters," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. Association for Computing Machinery, 2020, p. 108–120.

[12] Y. Peng *et al.*, "DL2: A Deep Learning-Driven Scheduler for Deep Learning Clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 1947–1960, 2021.

[13] T. Barreto Goes Perez *et al.*, "Bottleneck-Aware Task Scheduling Based on Per-Stage and Multi-ML Profiling," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019, pp. 510–518.

[14] P. Lindberg *et al.*, "Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems," *The Journal of Supercomputing*, vol. 59, no. 1, pp. 323–360, 2012.

[15] N. Makris *et al.*, "Enabling Open Access to LTE Network Components; the NITOS testbed paradigm," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015, pp. 1–6.

[16] I. Syrigos, N. Angelopoulos, and T. Korakis, "Optimization of Execution for Machine Learning Applications in the Computing Continuum," in *2022 IEEE Conference on Standards for Communications and Networking (CSCN)*.

[17] Y. Gorbachev *et al.*, "Openvino deep learning workbench: Comprehensive analysis and tuning of neural networks inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

[18] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.