EneX: An Energy-Aware Execution Scheduler for Serverless Computing

Seyed Hamed Rastegar, Hossein Shafiei, and Ahmad Khonsari

***Abstract*—The emerging serverless computing paradigm has recently attracted huge attention from both academia and industry. It brings benefits, such as less operational complexity, high scalability and availability, and lower costs. Serverless applications are usually partitioned into several chains of functions. The serverless provider should schedule functions for execution per customers' requests considering their chained nature. Also, the existing scheduling mechanisms for serverless platforms pay little attention to the reduction of energy consumption during functions' execution. To fill this gap, we present an energy-aware execution scheduler for serverless service providers named EneX. To do so, we formulate the minimization of energy consumption for executing the incoming chains of functions with specified computational loads and deadlines. Due to the intractability of the problem, we introduce a linear programming reformulation based on which, we propose an online scheduler. Finally, our experiments demonstrate the significant improvement of EneX in terms of energy efficiency.**

***Index Terms*—Chain of functions, cloud services, energy consumption, function-as-a-service (FaaS), scheduling, serverless computing.**

I. Introduction

THE complexity of infrastructure management for cloud users and the emergence of function-as-a-service resulted in introducing the concept of serverless computing [1]. Many novel applications including cyber-physical systems [2], 6G [3], and the Internet of Things (IoT) [4], can be implemented more efficiently using the serverless approach. The importance of the concept is such that it has been praised to become the face of cloud computing in the near future [5]. Especially, serverless computing is considered as an enabling technology for Industry 4.0 [6], [7].

The aim of the serverless services is threefold which are given as follows. 1) Relieve the users of cloud services from dealing with the infrastructures or the platforms. 2) Convert the billing model to the pay-as-you-go model. 3) Autoscale the service per customers' demand [8].

The biggest difference between the serverless paradigm with other cloud services is that the customer is only concerned about the code being run. The cloud takes care of how the code is run, the necessary external libraries, and any performance and scalability requirements automatically. The customer does not need to manage the OS or the middle-ware nor do they need to install and configure any execution runtime or software framework. The customer is charged based on their actual resource usage and not a time period, which is common in other cloud services. It provides an autoscaling feature in which, when the traffic changes over time, the number of resources allocated to that service will change to adapt to handle surges. This latter feature is essential for many IoT applications, especially that of Industrial IoT where the system often must handle unpredictable very rapid upheavals in resource-demanding surveillance or control tasks [9]. It also has great impact on the scheduling schemes, which we discuss later in this article. Instead of focusing on the resource limitations, serverless schedulers focus on optimizing other parameters that impact the performance of scheduling.

The basic entities in serverless computing are functions. The customers register their functions in the service provider. Then, those functions can be invoked either by an event or per the customer's request. The execution results are sent back to the customer. The invocation of the functions is delegated to one of the available computation nodes inside the service provider. Usually, these nodes are cloud containers, such as Docker [10] or an isolated runtime environment [11]. Serverless functions usually have two common features: 1) rather than single function invocation they are typically part of a chain of invocations and 2) apart from the computational load, each invocation has a firm deadline beyond, which the returned results are ignored by the application.

There are widely-used serverless applications in which functions are invoked in chains and each of those functions has its own deadline. Serverless machine learning pipelines, which have gained traction during the past few years [12] can be considered as one of those applications. For example, consider a video surveillance system in which a deep neural network inference model is utilized to identify moving objects. Serverless

Manuscript received 27 May 2022; revised 8 January 2023 and 3 June 2023; accepted 25 June 2023. Date of publication 30 June 2023; date of current version 19 January 2024. Paper no. TII-22-2267. *(Corresponding author: Seyed Hamed Rastegar.)*

Seyed Hamed Rastegar is with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran 19395-5746, Iran (e-mail: hrastegar@ipm.ir).

Hossein Shafiei is with the Faculty of Computer Engineering, K. N. Toosi University of Technology, Tehran 1969764449, Iran (e-mail: shafiei@kntu.ac.ir).

Ahmad Khonsari is with the School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran 1439957131, Iran, and also with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran 19395-5746, Iran (e-mail: a_khonsari@ut.ac.ir).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2023.3290985.

Digital Object Identifier 10.1109/TII.2023.3290985

machine learning services are good candidates for such systems as surveillance systems may experience burst of activity during short periods and become idle otherwise [13]. The autoscaling feature of serverless services can accommodate such scenarios. It releases developers from deployment, resource management, and scaling overheads [14], [15]. In a layered distributed inference model of a serverless pipeline, the execution time of each function impacts the timeliness of the whole system as the functions in each layer should synchronize with each other [16]. Thus, the execution time of functions should be constrained by deadlines. Another example of serverless applications with the abovementioned criteria is multimedia stream processing. In these applications, multimedia contents must be processed before streaming to the viewer. A multimedia content is divided into multiple subsequent parts that each one must be processed within a specified deadline [17]. So, the design of any scheduler for a serverless platform should take into account the chain of functions with their corresponding deadlines, which have not been well studied in the literature.

Many big technology companies now offer serverless computing services with different specifications and prices. As the popularity of serverless services increases, the number of service providers and their options for pricing will grow. Several factors affect the price offered by each provider. These factors range from the company's revenue strategy to the platform it uses, maintenance costs and the energy prices. It has been shown that the latter have the most impact on the service costs in cloud computing environments [18]. Also, cloud infrastructures are reported to consume a significant part of the total electricity worldwide [19]. Therefore, the reduction of energy consumption not only is important from the sustainable computing point of view but also reduces the overall costs and, thus, increases the pricing competitiveness of the service providers. Consequently, various notable cloud service providers have considered the energy efficiency as an important criteria in design and evolution of their infrastructures. For example, Amazon has made a considerable focus on energy efficiency by using different approaches, such as chip design, scheduling methods, and renewable energy [20]. An important focus of Google has been on energy efficiency and it has reported significant energy reduction by its data centers in years [21]. Google has done it by carefully justifying the temperature, employing sophisticated cooling systems, and efficient design and management of its servers. Also, Microsoft has employed four drivers of IT operational efficiency, IT equipment efficiency, data center infrastructure efficiency, purchasing of renewable energy, and reported considerable energy savings resulted in its cloud computing infrastructure [22].

Various approaches have been proposed in the literature to reduce the energy consumption of serverless services. Their main idea is to put inactive containers or virtual machines in a hibernate low-energy mode (called cold-state mode). The transition from cold-state to active mode incurs delays to the execution of functions [5]. The delay may go beyond the deadlines defined by the customer. Thus, in these approaches, the executions are scheduled so that the number of such transitions minimizes [23], [24]. While the current focus on minimizing the transitions from cold to active mode (called warm state) is important to reduce the total energy consumption and, thus, lower costs, those approaches do not consider energy minimization during the active mode.

None of the previous researches on serverless scheduling that we review in Section II, considers energy-aware serverless execution scheduling inside providers. Therefore, in this article, we design a scheduler for a serverless provider with the aim of executing the incoming function invocations with predefined deadlines while minimizing the energy consumption of the system. It is worth emphasizing as a data center is composed of different components, including cooling, lighting, servers, and energy conversion, its energy consumption is due to the energy used by these components [25]. However, as reported by different studies around 15% of total energy consumption of a data center is related to the processing part [26], from which almost its half is consumed in the active mode. This part is the focus of this research. To summarize, the main contributions of this article are given as follows.

1) We consider the realistic scenario of minimizing energy consumption in a serverless computing environment where function invocations are chained together each having its own execution deadline and computational requirement. To the best of authors knowledge, this work is the first that addresses such an issue.
2) We mathematically model the abovementioned problem and formulate it as an optimization problem.
3) To handle the problem, which is not tractable in its initial form, we introduce a procedure to reformulate it as a linear programming (LP) problem.
4) The LP-based reformulation however works in offline mode and, thus, needs all the information of the incoming tasks to the system in time. To overcome this challenge, we design an online scheduling procedure that can very well serve the incoming tasks in time and hence is more practical for real-world scenarios.
5) We investigate the properties of the online solution including time complexity and competitive ratio (CR) and show that it is scalable and has attractive performance.
6) Through extensive evaluations and real-world experiments, we show that our approach can achieve satisfactory performance compared to other possible schemes.

The rest of this article is organized as follows. Section II reviews background and related work. We explain the system model and problem formulation in Section III. Section IV presents the proposed solution. We evaluate the performance of the proposed schemes in Section V. Finally, Section VI concludes this article.

II. Background and Related Work

In this section, we provide an overview of related research works. We first give a review on applications of serverless computing. Then, we enumerate and explain several notable works for scheduling in cloud and serverless computing. Finally, we review the works considering energy efficiency in serverless computing.

Applications: The serverless computing paradigm has attracted attention from both academia and industry. It brings benefits, such as a pay-as-you-go pricing model, lower operational complexity, and the ability to autoscale. Although the concept is relatively new, several urban and industrial management applications already started to migrate to serverless services. These applications ranges from IoT [4] to smart waste management [27], oil and gas field management [28], and smart grids [29]. Other applications include connected vehicles [2], farm management [30], optical networks [31], and diagnosing the COVID-19 disease [32].

Cloud Scheduling: Scheduling tasks in a cloud environment has been vastly investigated from different viewpoints [33], such as deadline-constrained [34], energy-awareness [35], [36], and QoS-awareness [37]. The proposed approaches are either heuristic or analytical. In contrast to the cloud computing paradigm that mainly offers infrastructure as a service (IaaS), serverless computing attempts to deliver lightweight function as a service. Also, we note that the IaaS concept in cloud computing introduces considerable overhead when running a service by a customer. In what follows, we discuss the works that mainly focus on serverless computing.

Serverless Scheduling: In real-world scenarios, functions usually depend on some external data sources. The shipping of the data from those sources to a distant computation node incurs delay. It also increases the traffic inside the provider. As such, some schedulers prioritize data locality in the scheduling. Cloudburst [38] is an example of such schedulers. Resource-aware schedulers consider available resources of computation nodes in their function assignments. For example, FnSched [39] proposes a scheduling scheme that takes into account the resource usage of functions. Their idea is to place functions alongside each other in a way that the resource contention between colocated functions minimizes. Available software packages and libraries inside computation nodes are also important for reducing execution delays in some of the existing serverless platforms, such as OpenLambda [40]. This is mainly because the preparation and installation of those packages and libraries incur minute-scale delays. The scheduler proposed in [41], handles such a scenario. Xie et al. [42] considered offering service in wireless edge networks using serverless computing. They formulate a multiobjective optimization problem and propose a deep reinforcement learning framework to solve it. Banaei and Sharifi [43] introduced ETAS, a predictive scheduling scheme to reduce response times of invocations besides increasing workers' throughputs and resource utilization. However, it does not provide any analysis of the scheme. Furthermore, despite the mentioned contributions, none of these works focus on the important aspect of energy efficiency.

Energy-Aware Serverless Scheduling: As we discussed in Section I, energy efficiency is a fundamental issue in cloud and serverless computing. Thus, it is important for a serverless scheduling scheme to be energy aware. The transition from cold to warm state inside serverless providers incur order of multisecond to minutes delays [5]. As such, the main focus of previous energy-aware schedulers is to minimize such transitions. Suresh et al. [23] proposed ENSURE, which prevents cold starts by proactively reserving a few additional containers in a warm state to smoothly handle workload variations. Fifer [24] uses a similar approach i.e., it proactively spawns containers to avoid cold starts. Vahidinia et al. [44] also tried to mitigate the cold start phase using tools from machine learning. We note that these works particularly study handling the cold and warm state transitions and hardly go into many details of scheduling the system in the active mode (i.e., warm state). Different from these works, Aslanpour et al. [45] considered a more general scenario of serverless edge computing in an IoT network, and introduces energy-aware scheduling for it. However, it considers no function chaining and also deadlines for functions.

III. System Model and Architecture

In this section, we first explain the system architecture. Then, we introduce the scheduling problem in detail. Finally, the mathematical problem formulation is presented.

A. Architecture

We consider the system architecture for the serverless provider depicted in Fig. 1. EneX[1] (our proposed scheduler) can be implemented as a standalone framework. It may act as a proxy to existing services or it can directly be integrated into available serverless platforms, such as OpenWhisk[2]. EneX depends on two main modules to be able to schedule the execution of invoked functions: a *function profiler* and a *workflow analyzer*. The function profiler derives the deadlines associated with each function to be executed. These deadlines either are declared by the customer or can be stemmed by the profiler. The workflow analyzer extracts function chains. The extraction can be performed via declaring by the user or derived by the workflow analyzer. For example, AWS Step Functions, Alibaba Serverless Workflow, and Google Cloud Composer enable their users to declare the function chains using DAG-structured[3] serverless workflows. The workflow analyzer can derive chains based on analyzing past execution logs. Finally, EneX which is the essential part and is at the heart of this architecture determines how and when to execute functions. Both the profiler and workflow analyzer interface with EneX to make the scheduling decisions. It uses the data gathered from the profiler and analyzer modules and schedules the execution of functions inside each machine using the proposed algorithm. In this regard, EneX interacts with the invoker. The invoker executes the functions in turns for specified time durations and with allocated resources according to commands received from the EneX. Due to the critical role of the scheduler, this article focuses on proposing energy-aware scheduling for function executions.

[1]Energy-aware execution scheduler for serverless

[2]https://openwhisk.apache.org/

[3]Directed Acyclic Graphs (DAG) is usually used to show dependency between jobs in scheduling algorithms.

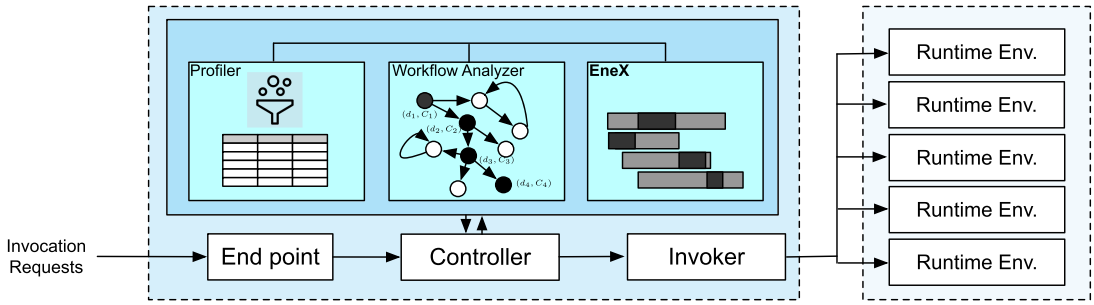


Fig. 1. Proposed energy-aware architecture for serverless providers.

B. Scheduling Problem

We consider a scenario in which a serverless service provider is responsible for providing computing resources to the incoming function invocations of its customers. In this article, we assume that the customers have previously registered their functions in the system. Thus, the provider has granted them the ability to invoke functions. The customers' chained invocation requests[4] arrive at the system continuously in time. In this regard, upon arrival of each request, the scheduler allocates the computational resources. The scheduler aims at minimizing the amount of consumed energy for the computation of tasks while conforming with the computational loads and deadlines.

Our proposed scheduler is based on two observations: 1) in real-world implementations, serverless functions are almost always chained together to form broader workflows, and 2) serverless functions usually have deadlines that are lengthier than their execution time. Putting these two together, we introduce a scheme that schedules the function execution within chains such that the total CPU utilization minimizes.

We assume that at the beginning there are N_u tasks submitted by different customers in the system. We note that a customer might submit and execute multiple tasks to the system; thus, more than one task out of the mentioned N_u tasks might correspond to a single customer. Thus, the scheduler first allocates the computational resources to these tasks. Then, it schedules the incoming tasks to the system over time. We denote the set of functions corresponding to task u by $\mathcal{T}_u = \{f_1^u, f_2^u, \ldots, f_{|\mathcal{T}_u|}^u\}$ where the subscripts indicate the order of the functions in the task. Each function has a predefined deadline for execution. We show these deadlines with $\{d_1^u, d_2^u, \ldots, d_{|\mathcal{T}_u|}^u\}$, respectively. Also, the computational loads of the functions are presented by $\{C_1^u, C_2^u, \ldots, C_{|\mathcal{T}_u|}^u\}$. We note that the deadline for each function is imposed by the application related to the task, which involves the function. In our model, we consider each task be a linear chain of subsequent functions, which should be executed sequentially in the order of their indices. Thus, executing ith function of a task is the prerequisite for starting the execution of function $(i+1)$. Table I summarizes the key notations used in the paper.

We note that various application scenarios match our above-mentioned assumption for each function in the chain to have its own individual deadline. We explained the video surveillance and multimedia stream processing applications as examples of such scenarios in the introduction. However, our formulation can be employed to consider many other simpler cases including a task just with a single deadline requirement.

[4]We refer to chained invocation requests as customers' tasks for simplicity.

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
N_u	Total number of tasks
$\mathcal{T}^u$	Set of functions of task u
f_i^u	ith function of task u
C_i^u	Computational requirement of f_i^u
d_i^u	Deadline of f_i^u
$c^u(t)$	Processing frequency devoted to task u at time instant t
$c(t)$	Total processing frequency at time instant t
σ_j	jth smallest member of set $\{d_i^u\}_{\forall i,u}$, $(j \geq 1)$, and $\sigma_0 = 0$
N_d	Total number of elements in set $\{d_i^u\}_{\forall i,u}$
$\mathcal{N}_i$	Set of all $j \in \{1, ..., N_d\}$ such that $d_{i-1}^u < \sigma_j \leq d_i^u$
y_j^u	Processing frequency allocated to task u in $(\sigma_{j-1}, \sigma_j]$
Y_j	Total processing frequency in interval $(\sigma_{j-1}, \sigma_j]$
$\lvert . \rvert$	Total number of elements in a set, i.e., its cardinality

In the following, we formulate the scheduling problem to handle the initially available N_u tasks at the system. Then, in Section IV, we design an optimal solution for this problem and introduce an online procedure to schedule the incoming tasks to the system.

C. Mathematical Problem Formulation

We define the processing frequency as $c^u(t)$, which is the amount of momentary processing unit in cycle per second for executing functions that belongs to task u at time instant t. Therefore, the total processing frequency of the system at time instant t is equal to

$$c(t) = \sum_{u=1}^{N_u} c^u(t). \tag{1}$$

Note that solving the scheduling problem is equivalent to determining the $c^u(t) \forall u, t$, and, thus, they could be expressed as the problem variables.

1) Constraints: As mentioned earlier, the function f_i^u, which its computation can be started just after it is finished for f_{i-1}^u (i.e., at d_{i-1}^u), should be computed by time instant d_i^u and its computational requirement is C_i^u cycles. Thus, the following

constraints must hold:

$$\int_{t=d_{i-1}^u}^{d_i^u} c^u(t)dt = C_i^u \quad \forall i,u \tag{2}$$

in which we set $d_0^u = 0$. Therefore, by this constraint the chain of functions of a task is taken into account.

2) Objective Function: Our objective is to minimize the energy consumption of the system. As pointed out earlier in Section I, the energy consumption here is just due to the computing components. The serverless processors are considered to be only CPU-based and the formulations, analysis, and results are accordingly. In this regard, we assume the serverless processors employ the popular and widespread power management technique of dynamic voltage and frequency scaling (DVFS). In DVFS, power consumption of a processor can be expressed in terms of its frequency as κf^3, where κ is the average switched capacitance per clock cycle [46]. We note that this relation is widely employed to model the energy consumption of tasks, executed using virtual machines [47], and applications [48], such as containers as is the case in serverless computing. Therefore, for our scenario where the frequency of the processor is $c(t)$ at time instant t as stated in (1), the energy consumption in a very small time interval of dt is equal to $\kappa c^3(t)dt$. Thus, we write the objective function in the entire considered interval, as follows:

$$\int_0^{d_{\max}} \kappa c^3(t)dt \tag{3}$$

where $d_{\max} = \max_{\forall i,u}\{d_i^u\}$. Therefore, the problem is formulated as

$$\begin{aligned} \underset{c(t),\{c^u(t)\}}{\text{minimize}} \quad & \int_0^{d_{\max}} \kappa c^3(t)dt \\ \text{subject to} \quad & \sum_{u=1}^{N_u} c^u(t) = c(t) \\ & \int_{t=d_{i-1}^u}^{d_i^u} c^u(t)dt = C_i^u \quad \forall i,u \\ & 0 \le t \le d_{\max} \\ & c^u(t), c(t) \ge 0 \quad \forall u,t. \end{aligned} \tag{4}$$

Discussion: In (4), we presented the optimization problem for scheduling the customers' tasks while minimizing the energy cost of the system. The variables of the problem, i.e., $c^u(t), c(t)\forall u$, and t are continuous function in time. Also, the objective function includes a nonlinear expression. These properties make the resulting optimization problem intractable in its current form. Therefore, we should investigate suitable ways to handle it. In the next section, we introduce a reformulation that makes the problem efficiently solvable.

IV. Proposed Solution

In this section, we reformulate the problem in (4) as a LP that can be handled much more efficiently.

To this aim, we first present useful Lemma 1. It should be noted that in this lemma we use a new set of $\{\sigma_0, \sigma_1, \ldots, \sigma_{N_d}\}$,

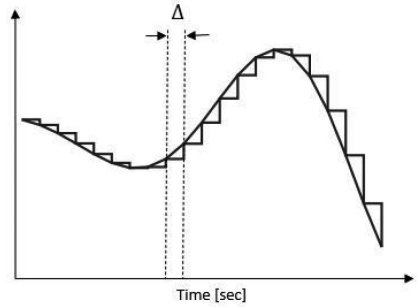


Fig. 2. Continuous function and its piece-wise linear approximation.

in which $N_d = \sum_{u=1}^{N_u} |\mathcal{T}_u|$ and $\sigma_0 = 0$, and $\{\sigma_1, \ldots, \sigma_{N_d}\}$ is obtained by sorting all the deadlines of the functions of the tasks (i.e., $\{d_i^u\}_{\forall i,\forall u}$) in the ascending order.

Lemma 1: In an interval between any two adjacent deadlines (i.e., $\sigma_j < t \le \sigma_{j+1}$), the optimum values of $c^u(t)\forall u$ for the problem in (4) are constant.

Proof: We prove the lemma by contradiction. Assume that the optimal solution of the problem in (4) is not constant between the two adjacent deadlines. Therefore, it would be in general similar to the continuous function in Fig. 2. We show that there is a constant function in this interval with less or equal energy consumption. However, the continuous function can be approximated using the staircase one in the figure. Thus, we use the staircase approximation instead in the following. We note that the staircase function will be equal to the continuous one by letting the time steps go to zero (i.e., $\Delta \to 0$), and since our discussion is valid for any value of Δ, all the conclusions obtained by considering the staircase function works for the continuous one as well. We consider the values of the function in two different time instants, namely A_1 and A_2 such that $A_1 \ne A_2$. If we assume the time steps be equal to Δ, the total computations done in these two time steps will be $(A_1 + A_2) \times \Delta$. We introduce another function in which these two values are both equal to $(A_1 + A_2)/2$ (i.e., constant in these two time steps) and the total computations is the same as the previous case.

We define $g(x) \triangleq x^3, \quad x \ge 0$, and since it is convex, we can write $g(\alpha x_1 + (1-\alpha)x_2) \le \alpha g(x_1) + (1-\alpha)g(x_2)$, $0 \le \alpha \le 1$. By setting $\alpha = 0.5, x_1 = A_1, x_2 = A_2$, and multiplying both sides by $2 \times \Delta$, we will have

$$2 \times \Delta \times \left(\frac{A_1 + A_2}{2}\right)^3 \le \Delta \times A_1^3 + \Delta \times A_2^3 \tag{5}$$

where the left- and right-hand sides of the abovementioned inequality are equal to the energy consumption of the newly introduced function and the optimal solution, respectively. Therefore, we introduced a new function that has equal or less energy consumption than the considered nonconstant optimal solution. Similar to the abovementioned approach, we proceed and show that the optimal solution of the problem in (4) is constant between two adjacent deadlines. Finally, without loss of generality, the optimal values for $c^u(t)\forall u$ could be considered to be constant between any two adjacent deadlines. ■

A. LP Reformulation

Using the abovementioned findings, we now rewrite the problem formulation and the involved variables. As the optimum value of function $c_i^u(t)$ can be considered constant in an interval of $\sigma_{j-1} < t \leq \sigma_j$, we state the allocated processing frequency in this interval for task u by new variable y_j^u. Also, by the abovementioned lemma, function $c(t)$ would be constant in each interval of the form $\sigma_{j-1} < t \leq \sigma_j$. Thus, we introduce new variables $Y_j, \quad j = 1, \ldots, N_d$ instead of $c(t)$ in the new formulation.

The total computational requirement of the ith function of task u is equal to C_i^u and its corresponding constraint was stated in (2) for continuous variables. For the new set of variables, this constraint would be as follows:

$$\sum_{j \in \mathcal{N}_i} (\sigma_j - \sigma_{j-1}) \times y_j^u = C_i^u \qquad \forall i, u \tag{6}$$

where $\mathcal{N}_i = \{j \in \{1, \ldots, N_d\} | d_{i-1}^u < \sigma_j \leq d_i^u\}$. We should note that as (6) is the representation of constraint (2) in the reformulation, the summation should be done in the interval of $(d_{i-1}^u, d_i^u]$ that is equivalently stated by set $\mathcal{N}_i$ here.

According to the abovementioned discussions, the objective function in (4), that we are aiming to minimize it, is now written as

$$\kappa \sum_{j=1}^{N_d} (\sigma_j - \sigma_{j-1}) Y_j^3. \tag{7}$$

This objective function is convex but due to its nonlinearity, the related optimization problem will be too complex and can not be solved efficiently in time. To tackle this issue, we state the following lemma.

Lemma 2: The objective function in (7) can be equivalently stated as

$$\max_j Y_j. \tag{8}$$

Proof: See Appendix A. ■

To write the new objective function in (8) in a suitable linear form, we introduce the nonnegative auxiliary variable θ as the objective function subject to new constraints of $Y_j \leq \theta \forall j$. It can be easily verified that this is equivalent to using objective function in (8). The resulting optimization problem is formulated as

$$\begin{aligned} \underset{\theta, \{y_{i,j}^u\}, \{Y_j\}}{\text{minimize}} \quad & \theta \\ \text{subject to} \quad & Y_j \leq \theta \qquad \forall j \\ & \sum_{u=1}^{N_u} y_j^u - Y_j = 0 \quad \forall j \\ & \sum_{j \in \mathcal{N}_i} (\sigma_j - \sigma_{j-1}) \times y_j^u = C_i^u \qquad \forall i, u, \\ & \theta \geq 0, \quad y_j^u, Y_j \geq 0 \quad \forall j, u. \end{aligned} \tag{9}$$

We note that y_j^us, which correspond to the allocated processing frequencies to task u in the intervals of $(\sigma_{j-1}, \sigma_j]$ are nonnegative real numbers. Also, Y_j is a summation over y_j^us for all the tasks and θ is an upper-bound on Y_js; Thus, both of them are nonnegative real numbers. On the other hand, all the relations in (9) are linear. Therefore, the reformulated optimization problem obtained in (9) is an LP and in comparison with (4), it can be solved more efficiently with less complexity.

Discussion 1: Our proposed scheduler i.e., EneX advisedly spreads computational loads over time and, thus, reduces overall energy consumption. Fig. 3 shows an illustrative example of the idea. There are four functions in the example. The gray area shows the deadline duration for each function and the black area shows the time it takes for the execution of the function given the available processing frequency at that moment. The left figure depicts our proposed approach and the right shows a strategy that starts execution just upon arrival. The latter takes higher more momentary processing frequency usage and, thus, more energy consumption.

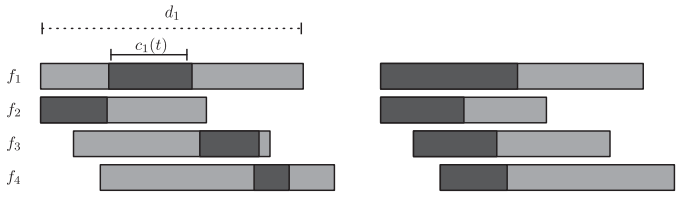


Fig. 3. Example of scheduling for two scenarios: our approach (left), upon arrival execution (right).

B. Online Scheduling

The framework discussed in the previous section solves the problem when there is just a set of initially available tasks in the system. Using the results of the previous section, we design a scheme to schedule the customers' tasks entering the system in time. In this regard, the provider first schedules the initially available tasks using (9), and then schedules each incoming task upon its arrival. A new incoming task u has a set of functions characterized by the their computational requirements, $C_i^u, i \in \{1, 2, \ldots, |\mathcal{T}^u|\}$, and deadlines, $d_i^u, i \in \{1, 2, \ldots, |\mathcal{T}^u|\}$. Here, we use a similar approach to the one in (9). The only difference is that we should consider the previously scheduled processing frequencies represented by Y_js. To this aim, we first construct set of deadlines. We assume that the new task execution request indexed by u is submitted to the system at time t_u, and j_u is such that $\sigma_{j_u-1} < t_u \leq \sigma_{j_u}$. Therefore, the new set of ordered deadlines, that we present it by $\{\hat{\sigma}_0, \hat{\sigma}_1, \ldots, \hat{\sigma}_J\}$, is resulted via concatenating t_u, $\{\sigma_j\}_{j \geq j_u}$, and $\{d_i^u\}_{i \in \mathcal{T}^u}$, and sorting them in ascending order. The previously assigned values of Y_js, represent the amount of processing frequencies in the intervals between σ_js. To match with the newly introduced deadlines, we restate the values of these processing frequencies in the intervals between $\hat{\sigma}_j$s as $\hat{Y}_j$s, which are related to the Y_js as follows:

$$\hat{Y}_j = Y_l, \quad if \quad \sigma_{l-1} < \hat{\sigma}_j \leq \sigma_l. \tag{10}$$

Also, we define $\hat{\mathcal{N}}_i \triangleq \{j \in \{1, \ldots, J\} | d_{i-1}^u < \hat{\sigma}_j \leq d_i^u\}$.

Thus, upon arrival of the uth task, the scheduler allocates the resources as follows:

$$\begin{aligned} \underset{\theta,\{\hat{y}_j^u\},\{\hat{Y}_j\}}{\text{minimize}} \quad & \theta \\ \text{subject to} \quad & \hat{y}_j^u + \hat{Y}_j \le \theta \quad \forall j \\ & \sum_{j\in\hat{\mathcal{N}}_i} (\hat{\sigma}_j - \hat{\sigma}_{j-1}) \times \hat{y}_j^u = C_i^u \quad \forall i \\ & \theta \ge 0, \quad \hat{y}_j^u \ge 0 \quad \forall j. \end{aligned} \tag{11}$$

After some manipulations and using typical constrained optimization methods, such as Karush–Kuhn–Tucker conditions [49], for problem in (11), the obtained solution is stated as

$$\hat{y}_j^u = \left(\lambda_i^u - \hat{Y}_j\right)^+ \quad \forall j \in \hat{\mathcal{N}}_i \tag{12}$$

where

$$\begin{aligned} & \sum_{j\in\hat{\mathcal{N}}_i} \left(\lambda_i^u - \hat{Y}_j\right)^+ = C_i^u \quad \forall i \\ & \lambda_i^u \ge 0 \quad \forall i \end{aligned} \tag{13}$$

and $(x)^+$ is equal to x for $x \ge 0$, and zero, otherwise. Therefore, the solution of (11) is obtained by first solving (13) for λ_i^us and then setting $\hat{y}_j^u = (\lambda_i^u - \hat{Y}_j)^+$. The overall procedure for determining $\hat{y}_j^u$s could be done as presented in Algorithm 1. In this algorithm, using the values of parameters obtained from scheduling the previous requests, we first construct $\hat{\sigma}_j$s, and $\hat{Y}_j$s in Lines 1 and 2, respectively. For each i starting from $i = 1$ to $i = |\mathcal{T}_u|$ the following procedure inside the for loop is done. After setting initial values for $\hat{\mathcal{N}}_0$ and $\hat{\mathcal{N}}_i$, in each iteration of the while loop, we assume that $\lambda_i^u \ge \hat{Y}_j$, and calculate the value of λ_i^u using (13) that is stated according to the expression in Line 9 of the algorithm. Using the obtained value of λ_i^u, we then determine $\hat{y}_j^u = \lambda_i^u - \hat{Y}_j$. If the resulting $\hat{y}_j^u$ for all $j \in \hat{\mathcal{N}}_i$ be nonnegative, the final solutions for the current i are obtained and we can start the procedure for $i + 1$. Otherwise, we update $\hat{\mathcal{N}}_0$ according to Line 15, and set $\hat{y}_j^u = 0$ for $j \in \hat{\mathcal{N}}_0$. We solve the equation for the remaining $\hat{y}_j^u$s. We continue the same procedure in each iteration of the while loop until all the $\hat{y}_j^u$s become nonnegative according to Line 11 of the algorithm. In this case, we set the parameter Ind_i to zero so that the while loop does not continue for index i and the next iteration of upper level for loop will be started or if $i = |\mathcal{T}_u|$ the algorithm is finished.

Note that the abovementioned formulation and the algorithm are for the case that after initial scheduling, a task is submitted to the system. However, for the next arriving tasks the same procedure should be done by setting the algorithm input values to $Y_j = \sum_i \hat{y}_j^u + \hat{Y}_j$ where $\hat{y}_j^u$s are the previous outputs of Algorithm 1, and replacing $\{\sigma_j\}$ by $\{\hat{\sigma}_j\}$ obtained above.

C. Discussion on Online Scheduling

In our problem, tasks arrive at the system in time, and the serverless provider must schedule the resources to meet the computation demands and deadlines of these incoming tasks.

Algorithm 1: Online Scheduling.

Input: Y_js, σ_js,d_i^us, C_i^us
Output: $\hat{y}_j^u$s: Optimal values of processing frequencies

1 Construct set of $\hat{\sigma}_j$s by concatenating σ_js and d_i^us, and sorting them in ascending order
2 Construct the new set $\hat{Y}_j$s, where $\hat{Y}_j = Y_l$, if $\sigma_{l-1} < \hat{\sigma}_j \le \sigma_l$.
3 Initialize $Ind_i = 1, \quad \forall i \in \{1, ..., |\mathcal{T}_u|\}$
4 **for** i = *1:*$|\mathcal{T}_u|$ **do**
5 Set $\hat{\mathcal{N}}_0 = \emptyset$
6 Set $\hat{\mathcal{N}}_i$ = Indices of $\hat{\sigma}_j$s such that $d_{i-1}^u < \hat{\sigma}_j \le d_i^u$
7 **while** $Ind_i \neq 0$ **do**
8 Set $\hat{\mathcal{N}}_i = \hat{\mathcal{N}}_i - \hat{\mathcal{N}}_0$
9 Set $\lambda_i^u = \dfrac{C_i^u + \sum_{j\in\hat{\mathcal{N}}_i} \hat{Y}_j}{|\hat{\mathcal{N}}_i|}$
10 Set $\hat{y}_j^u = \lambda_i^u - \hat{Y}_j, \quad \forall j \in \hat{\mathcal{N}}_i$
11 **if** $\hat{y}_j^u \ge 0, \forall j \in \hat{\mathcal{N}}_i$ **then**
12 Set $Ind_i = 0$
13 **end**
14 **else**
15 Define $\hat{\mathcal{N}}_0$ = set of all $j \in \hat{\mathcal{N}}_i$ such that $\hat{y}_j^u < 0$
16 Set $\hat{y}_j^u = 0, \forall i \in \hat{\mathcal{N}}_0$
17 **end**
18 **end**
19 **end**

To this aim, we first formulate the problem for the scenario, in which it is assumed that the information of all tasks is available at the system and the scheduler solves the problem using this information. For this scenario, we present the optimal solution as LP in Section IV-A, and name it the *offline* solution of the problem. We then design an *online* solution in Section IV-B based on the offline one. In this online solution presented in Algorithm 1, we assume that initially a portion of tasks have been submitted to the system and the resources are scheduled for them using the offline solution. Then, upon arrival of each request, we use the proposed online algorithm to allocate the resources to it.

There are two important points when considering online and offline solutions. First, the offline solution cannot be used for a large number of tasks submitted to the system as the required time and also memory drastically increases by increasing the number of tasks. However, the online solution does not encounter such an issue. Second, in a practical scenario the required information for the offline solution to work, including the deadline and computational requirements of the future tasks already not submitted to the system, is not available. Thus, the practical solution for such a system is to schedule the request upon arrival as is done in the online solution. We note that when evaluating the performance of the proposed schemes in Section V, we discuss the main features of the online solution compared with that of offline by conducting several experiments.

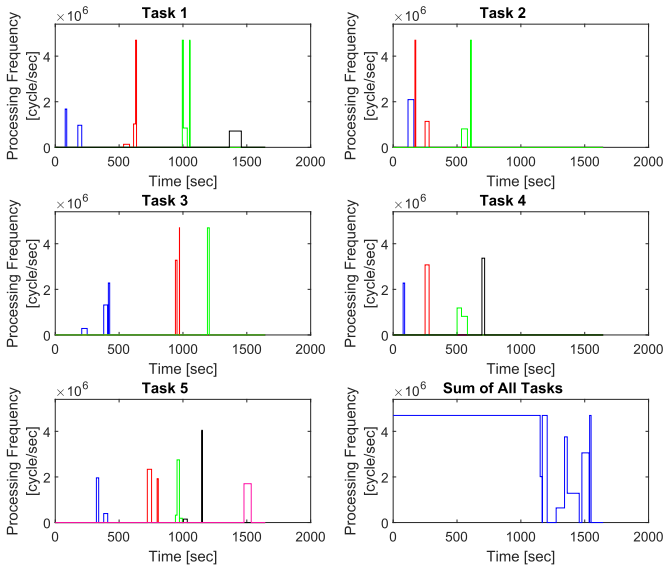


Fig. 4. Example of using the proposed method for allocation of computational resources. The first to the fifth functions of a task are shown, respectively, with blue, red, green, black, and pink colors. The scheme has scheduled them correctly in order.

TABLE II
PARAMETERS OF THE FIVE CONSIDERED TASKS

u	$\lvert\mathcal{T}_u\rvert$	$C_i^u(\times 10^6)$	d_i^u
1	4	$49, 57, 73, 67$	$207, 679, 1055, 1478$
2	3	$96, 65, 68$	$162, 347, 613$
3	3	$77, 45, 67$	$474, 973, 1211$
4	4	$27, 97, 79, 71$	$93, 451, 629, 784$
5	5	$47, 95, 60, 17, 97$	$492, 853, 995, 1276, 1539$

V. PERFORMANCE EVALUATION

In this section, we present the numerical results for the proposed scheme. To this aim, we assume that the number of functions of a task is an integer randomly distributed with equal probability in [1,5]. We generate the computational requirements and the deadlines of the functions of the tasks according to the uniform distribution in the intervals of $[1, 100] \times 10^6$ cycles, and [50, 500] s, respectively. Also, we set $\kappa = 10^{-24}$.

A. Illustrative Example

In order to show how our method works, we consider a scenario, where $N_u = 30$ tasks are initially in the system, and solve the scheduling problem for them using (9). The allocation of computational resources in terms of processing frequency in time for the first five tasks is plotted in Fig. 4. Also, the parameters for these five tasks in the experiment are reported in Table II, which indicates the number of functions for each task and the corresponding computational and deadline requirements. In the figure and for a better representation, the computational resources allocated to different functions of each task are shown in distinct colors. It is worth mentioning that the maximum value of time in these plots depends on the maximum deadline of the functions of each task. The results verify that our proposed method schedules the functions precisely in order and preserves their deadline constraints. Note that the total computational resources used by the system to execute the tasks of all the N_u tasks in time is plotted in this figure, and as expected our method tries to maintain a constant processing frequency in time in order to have the minimum energy usage. To investigate how our method works for greater values of N_u, plots of the total processing frequencies for $N_u = 50$ and $N_u = 100$ are presented in Fig. 5. According to these plots, by increasing the number of tasks in the system the total allocated processing frequency is increased. Another observation from these plots is that the time duration in which the total processing frequency is constant decreases by increasing N_u.

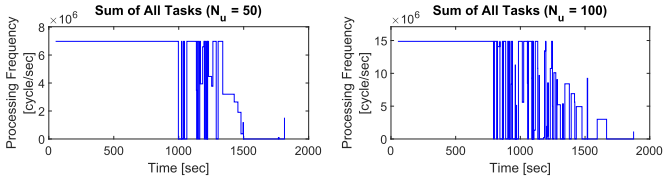


Fig. 5. Total processing frequencies for two scenarios with $N_u = 50$ and $N_u = 100$ tasks initially at the system.

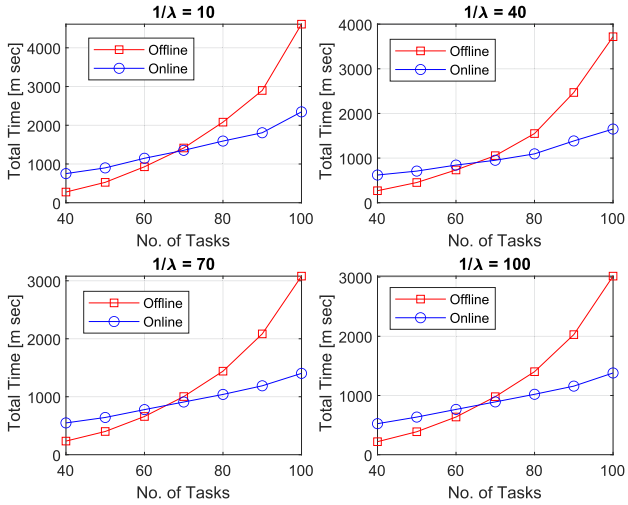


Fig. 6. Time required to schedule the tasks submitted to the system for online and offline schemes for different values of λ.

B. Online versus Offline

In this section, we sketch important features of our proposed online scheduling scheme by conducting several experiments.

1) Scalability: We have plotted the time required for both the online and offline schemes to schedule the submitted task to the system in Fig. 6. According to this figure, the time required for the offline scheme increases drastically by increasing the number of tasks while the time for the online method has an almost linear behavior in this scenario. It is an interesting feature for an algorithm to have a linear time complexity so that it can scale well by increasing problem dimensions. It is also worth emphasizing that according to Fig. 6, the time required by the

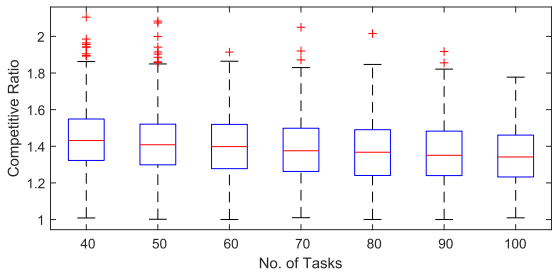


Fig. 7. CR of the online scheme.

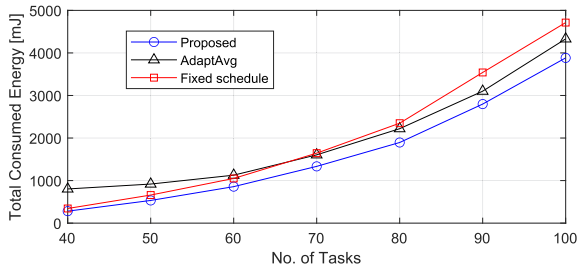


Fig. 8. Total energy consumption versus number of tasks.

online scheme for scheduling a single task is of minor value which is on average less than 0.025 s.

2) Performance: To evaluate the performance of an online scheme, the widespread method is to compare its cost with the cost of the offline solution. The ratio between these costs (called CR) is usually considered an important factor in the performance of online schemes [50]. However, finding this ratio theoretically is not possible for many problems. An alternative and acceptable approach is to empirically study this CR by conducting experiments. In this regard, we calculated the ratio of these costs for numerous runs of the experiment and statistically reported the CR for the online method proposed in this work in Fig. 7. The statistics of CR for different number of tasks shows a mean and maximum value of around 1.5 and 2, respectively.

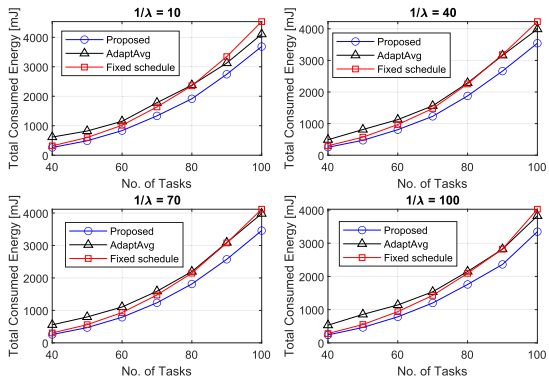


Fig. 9. Sum of energy consumption versus number of tasks in online mode for different values of λ.

C. Comparison

In this section, we evaluate the performance of our proposed scheme. In particular, the following schemes for comparison are used.

1) *Proposed:* It refers to our proposed methods in this work.
2) *Fixed Schedule:* In this method, at a given time the scheduler schedules a function with the earliest deadline from each task and allocates a fixed processing frequency to each of them. The fixed frequency can be simply determined by dividing the residual computational requirement of that function by the remaining time to its deadline. The total processing frequency of the system at each time instant, thus, would be the sum of these fixed frequencies. Note that the idea behind using this method for comparison is that according to Lemma 1, the optimal processing frequency for scheduling is constant between any two consecutive deadlines.
3) *AdaptAvg:* This scheme uses the computational and deadline requirements of the submitted tasks. In particular, it first determines an average quantity by dividing the sum of required computation loads by the maximum value of the deadlines. Then, it performs the scheduling task-by-task and tries to adapt itself to a constant processing frequency as close as possible to the obtained average value above.

For the comparison, we perform the following experiments.

1) Experiment 1: In this experiment, we consider a scenario in which the customers have submitted their tasks to the system and the provider schedules them according to (9). Therefore, in this scenario there is no incoming task in time and the orders of the tasks have been justified initially. We plotted the total consumed energy versus the number of tasks in Fig. 8. The results demonstrate the better performance of the proposed scheme in terms of energy consumption compared with the two other approaches

2) Experiment 2: In this experiment, we investigate the scenario discussed in Section IV-B. In particular, we assume that there are several tasks initially submitted at the system, and the new task execution requests arrive at the system according to the Poisson distribution with parameter λ. The provider schedules the tasks according to Algorithm 1. In Fig. 9, we plotted the energy consumption of our proposed scheme and the two other methods introduced previously versus the total number of tasks (N_u). It is worth noting that we assume that $\lfloor N_u/2 \rfloor$ tasks initially have been submitted to the system and the remaining tasks (i.e., $N_u - \lfloor N_u/2 \rfloor$) enter one-by-one according to the Poisson distribution. We conducted the experiment for different values of mean interarrival time (i.e., equal to $1/\lambda$) in Fig. 9. According to the figure, the performance of the proposed scheme is considerably better than the two other schemes. Also, in plots of Fig. 9, the energy consumption decreases by increasing $1/\lambda$. This observation is due to less computation to be performed in the same time duration by increasing $1/\lambda$, and, thus, less energy would be consumed by the abovementioned scheduling schemes.

We note that in this comparison there is a tradeoff between the complexity of a method and its resulting energy consumption. In particular, our proposed method might need more computation

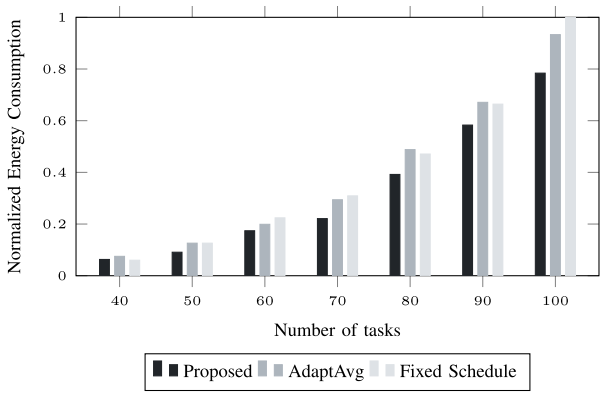


Fig. 10. Normalized energy consumption for real-world OpenWhisk implementation of three various scheduling schemes.

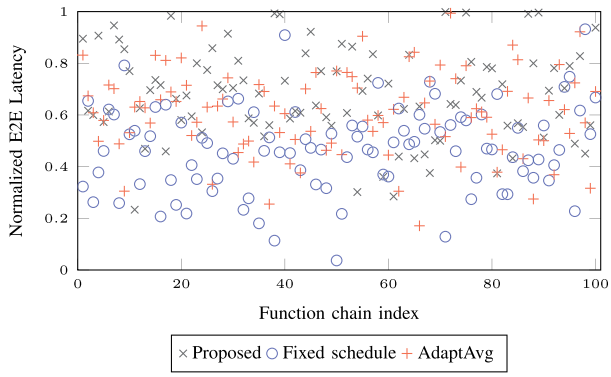


Fig. 11. Normalized E2E latencies of 100 function chains for three different scenarios. Chains are indexed from 1 to 100.

than the Fixed Schedule and AdaptAvg approaches but at the same time, it presents lower energy consumption. However, as we pointed out when discussing the online and offline methods in Section IV-C the complexity of the online solution is acceptable in practice and is not a concern.

D. Real-World Implementation

We implemented our proposed scheduler on top of OpenWhisk's `core.scheduler` module. For the sake of comparison we also implemented two other previously described scheduling strategies i.e., Fixed Schedule and AdaptAvg approaches. We evaluated our implementation on a 16-node OpenWhisk cluster. We used a dedicated node for the OpenWhisk's `controller` and `scheduler`. Nodes were HP Proliant DL360 G9 servers connected over Gigabit Ethernet switches. We utilized an open-source version of Intel Power Gadget[5] to measure the energy consumed by each node. Using this tool to measure energy consumption in serverless environments is a common approach [24], [51], [52]. We used a synthetic workload similar to that of numerical experiments described before.

Fig. 10 depicts the evaluation results. We measure the energy consumption as total energy consumed. We observe that in our real-world OpenWhisk implementation the proposed scheme can achieve lower energy consumption compared with that of Fixed Schedule and AdaptAvg approaches. Also, the results from this real-world implementation matches to the ones from simulations and, thus, it validates the analysis done in this work.

Fig. 11 shows the scatter plot for normalized end-to-end latencies of 100 different function chains. Here, the function chains are indexed from 1 to 100. We calculated the time between a chain being scheduled until the execution is finished. As shown in the figure, on average the latencies are increased marginally using our approach. This is mainly due to the nature of our approach, which tries to delay the execution of functions to reduce energy consumption while conforming with the deadlines. Fixed Schedule exhibits the lowest latency, however, as discussed previously in this section, its energy consumption is significantly higher.

[5]https://github.com/sosy-lab/cpu-energy-meter.

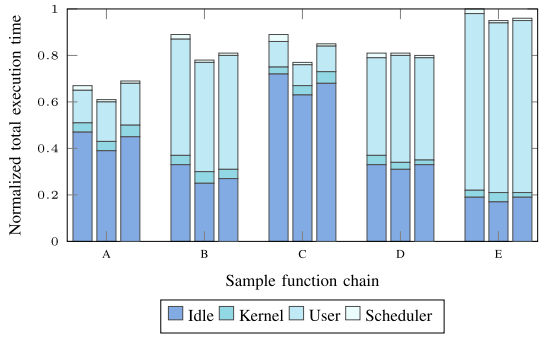


Fig. 12. Normalized average CPU profiles of five sample function chains for three different scenarios; 1) our proposed approach (right columns), 2) Fixed schedule (middle columns), and 3) AdaptAvg (left columns).

To evaluate the overheads imposed by our approach, we calculated and compared the total execution times for the three different scenarios. Fig. 12 depicts the comparison for five sample chains. It shows the fraction of CPU time spent in the idle, kernel, user modes, and scheduling, respectively. The overhead of EneX's scheduler is negligible compared with the total execution time and slightly higher than that of Fixed Schedule.

VI. Conclusion

Serverless computing has recently attracted many research studies in the scientific community and more importantly appealed considerable interest from the industry. Current approaches for scheduling in serverless providers do not well address the energy awareness issue and also do not properly consider the chained nature of functions invocations. This article proposed EneX, which is an energy-aware scheduler for serverless function chains that conforms with the deadlines defined for each function. We presented a mathematical formulation for the problem. In order to handle the complexity of the formulated problem, we introduced a tractable LP reformulation. Based on that, we designed an online scheduling scheme for the problem. Our evaluations confirmed that EneX makes significant improvements in terms of energy efficiency. We note that in

this article, we considered the tasks with linear chain structure, in which completely executing a function is the prerequisite for starting the next one, in the formulation and solution. Furthermore, more general structures of tasks, e.g., with parallel executable functions, can also be supported by converting them into multiple tasks with linear topology and use them as the inputs to our introduced framework. However, determining the optimal procedure for these scenarios may require further studies and is an interesting topic of future work.

Appendix A
Proof of Lemma 2

The lemma argues that the optimum solution of problem in (9) has the minimum value of $\max_j Y_j$. We prove the lemma by contradiction. Suppose that the optimum solution of the problem with the minimum value of objective in (7) does not have the minimum value of $\max_j Y_j$ in the feasible region of the problem. We show this optimum solution by $\bar{Y}_1, \bar{Y}_2, \ldots, \bar{Y}_{N_d}$. Assume that $\bar{Y}_k = \max_j \bar{Y}_j$. On the other hand, we introduce an other set of points in the feasible region of the problem by $Y'_1, Y'_2, \ldots, Y'_{N_d}$ such that $Y'_k = \bar{Y}_k - \epsilon_k$, $Y'_l = \bar{Y}_l + \epsilon_l$, and $Y'_j = \bar{Y}_j, j \neq k, l$. l is any arbitrary index in $\{1, \ldots, k-1, k+1, \ldots, N_d\}$. Also, $\epsilon_k, \epsilon_l \geq 0$, and we assume

$$Y'_l + \epsilon_l \leq \bar{Y}_k \tag{14}$$

so that $\max_j Y'_j$ would not be greater than $\max_j \bar{Y}_j$. Also, we set $D_j \triangleq \sigma_j - \sigma_{j-1}$. First, to preserve the problem constraints, we should have

$$\begin{aligned} D_k \times \bar{Y}_k + D_l \times \bar{Y}_l &= D_k \times Y'_k + D_l \times Y'_l \\ &= D_k \times (\bar{Y}_k - \epsilon_k) + D_l \times (\bar{Y}_l + \epsilon_l) \end{aligned} \tag{15}$$

which results that $\epsilon_k D_k = \epsilon_l D_l$. The difference of energy consumption for the two schemes is equal to

$$\begin{aligned} &\kappa \sum_{j=1}^{N_d} D_j \times Y'^3_j - \kappa \sum_{j=1}^{N_d} D_j \times \bar{Y}^3_j \\ &= \kappa \left(D_k (\bar{Y}_k - \epsilon_k)^3 + D_l \left(\bar{Y}_l + \epsilon_l\right)^3 - D_k \bar{Y}^3_k + D_l \bar{Y}^3_l \right) \\ &= \kappa D_k \left(3\bar{Y}_k \epsilon_k^2 - 3\bar{Y}_k^2 \epsilon_k - \epsilon_k^3 \right) + D_l \left(3\bar{Y}_l \epsilon_l^2 + 3\bar{Y}_l^2 \epsilon_l + \epsilon_l^3 \right). \end{aligned} \tag{16}$$

However, since $\epsilon_k D_k = \epsilon_l D_l$, we rewrite the abovementioned relation as

$$\kappa D_k \epsilon_k \left(3\bar{Y}_k \epsilon_k - 3\bar{Y}_k^2 - \epsilon_k^2 + 3\bar{Y}_l \epsilon_l + 3\bar{Y}_l^2 + \epsilon_l^2 \right). \tag{17}$$

As $\kappa D_k \epsilon_k \geq 0$, we investigate the expression in parenthesis in (17) hereafter. Considering (14), we write $3\bar{Y}_l^2 + 3\bar{Y}_l \epsilon_l = 3\bar{Y}_l(\bar{Y}_l + \epsilon_l) \leq 3\bar{Y}_l \bar{Y}_k$. Combining this result with the first two terms in parenthesis in (17) and again employing (14), we form another inequality as $-3\bar{Y}_k^2 + 3\bar{Y}_k \epsilon_k + 3\bar{Y}_l \bar{Y}_k = 3\bar{Y}_k(\epsilon_k + \bar{Y}_l - \bar{Y}_k) \leq 3\bar{Y}_k(\epsilon_k - \epsilon_l)$. Therefore, the expression in (17) is written as

$$\begin{aligned} 3\bar{Y}_k(\epsilon_k - \epsilon_l) + \epsilon_l^2 - \epsilon_k^2 &= (\epsilon_l - \epsilon_k)\left(\epsilon_l + \epsilon_k - 3\bar{Y}_k\right) \\ &\leq \left(\bar{Y}_k - \bar{Y}_l - \epsilon_k\right)\left(-2\bar{Y}_k + \epsilon_k - \bar{Y}_l\right) \end{aligned} \tag{18}$$

in which we have used $\epsilon_l \leq \bar{Y}_k - \bar{Y}_l$ from (14). If we set $\epsilon_k = \gamma(\bar{Y}_k - \bar{Y}_l)$, $0 \leq \gamma \leq 1$ (18) would become as follows:

$$\begin{aligned} &\left(\bar{Y}_k - \bar{Y}_l - \epsilon_k\right)\left(-2\bar{Y}_k + \epsilon_k - \bar{Y}_l\right) \\ &= (1-\gamma)\left(\bar{Y}_k - \bar{Y}_l\right)\left((\gamma - 2)\bar{Y}_k - (1+\gamma)\bar{Y}_l\right). \end{aligned} \tag{19}$$

However, as we have supposed that $\bar{Y}_k \geq \bar{Y}_l$ and $0 \leq \gamma \leq 1$, the above expression is less than or equal to zero. Therefore, we introduce feasible points $Y'_1, Y'_2, \ldots, Y'_{N_d}$ for which $\sum_{j=1}^{N_d} D_j \times Y'^3_j \leq \sum_{j=1}^{N_d} D_j \times \bar{Y}^3_j$, which is in contrast with $\{\bar{Y}_j\}_{j=1}^{N_d}$ to be the optimum solution of the problem. Thus, the lemma is proved.

References

[1] R. Buyya et al., "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 1–38, 2018.

[2] H.-W. Deng, M. Rahman, M. Chowdhury, M. S. Salek, and M. Shue, "Commercial cloud computing for connected vehicle applications in transportation cyberphysical systems: A case study," *IEEE Intell. Transp. Syst. Mag.*, vol. 13, no. 1, pp. 6–19, Spring 2021.

[3] H. Tataria et al., "6G wireless systems: Vision, requirements, challenges, insights, and opportunities," *Proc. IEEE*, vol. 109, no. 7, pp. 1166–1199, Jul. 2021.

[4] C. Cicconetti, M. Conti, and A. Passarella, "A decentralized framework for serverless edge computing in the Internet of Things," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 2166–2180, Jun. 2020.

[5] E. Jonas et al., "Cloud programming simplified: A berkeley view on serverless computing," 2019, *arXiv:1902.03383*.

[6] G. Aceto, V. Persico, and A. Pescape, "A survey on information and communication technologies for industry 4.0: State-of-the-art, taxonomies, perspectives, and challenges," *IEEE Commun. Surv. Tut.*, vol. 21, no. 4, pp. 3467–3501, Oct.–Dec. 2019.

[7] P. Dziurzanski, J. Swan, and L. S. Indrusiak, "Value-based manufacturing optimisation in serverless clouds for industry 4.0," in *Proc. Genet. Evol. Comput. Conf.*, 2018, pp. 1222–1229.

[8] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: A survey of opportunities, challenges, and applications," *ACM Comput. Surv.*, vol. 54, no. 11s, 2022, Art. no. 239.

[9] M. S. Aslanpour et al., "Serverless edge computing: Vision and challenges," in *Proc. Australas. Comput. Sci. Week Multiconf.*, 2021, pp. 1–10.

[10] A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, "Serverless computing for container-based architectures," *Future Gener. Comput. Syst.*, vol. 83, pp. 50–59, 2018.

[11] S. Shillaker and P. Pietzuch, "Faasm: Lightweight isolation for efficient stateful serverless computing," in *Proc. USENIX Annu. Tech. Conf. USENIX Assoc.*, 2020, pp. 419–433.

[12] A. Barrak, F. Petrillo, and F. Jaafar, "Serverless on machine learning: A systematic mapping study," *IEEE Access*, vol. 10, pp. 99337–99352, 2022.

[13] M. Zhang, F. Wang, Y. Zhu, J. Liu, and Z. Wang, "Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines," in *Proc. 12th ACM Multimedia Syst. Conf.*, 2021, pp. 80–93.

[14] S. Park, J. Choi, and K. Lee, "All-you-can-inference: Serverless DNN model inference suite," in *Proc. 8th Int. Workshop Serverless Comput.*, 2022, pp. 1–6.

[15] F. Xu, Y. Qin, L. Chen, Z. Zhou, and F. Liu, "λDNN: Achieving predictable distributed DNN training with serverless architectures," *IEEE Trans. Comput.*, vol. 71, no. 2, pp. 450–463, Feb. 2022.

[16] R. Stahl, Z. Zhao, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Fully distributed deep learning inference on resource-constrained edge devices," in *Proc. 19th Int. Conf. Embedded Comput. Syst.: Architectures, Model. Simul.*, 2019, pp. 77–90.

[17] C. Denninnart and M. A. Salehi, "Harnessing the potential of function-reuse in multimedia cloud systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 617–629, Mar. 2022.

[18] C. Wu, R. Buyya, and K. Ramamohanarao, "Cloud pricing models: Taxonomy, survey, and interdisciplinary challenges," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–36, 2019.

[19] Z. Wen et al., "Running industrial workflow applications in a software-defined multicloud environment using green energy aware scheduling algorithm," *IEEE Trans. Ind. Inform.*, vol. 17, no. 8, pp. 5645–5656, Aug. 2021.

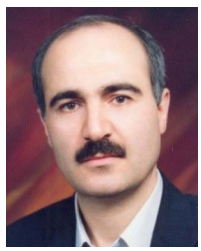
[20] Amazon, "Delivering progress every day: Amazon's 2021 sustainability report," Accessed: May 25, 2023. [Online]. Available: https://sustainability.aboutamazon.com/2021-sustainability-report.pdf
[21] Google, "Efficiency of Google data centers," Accessed: May 25, 2023. [Online]. Available: https://www.google.com/about/datacenters/efficiency
[22] Microsoft, "The carbon benefits of cloud computing: A study of the Microsoft cloud," Accessed: May 25, 2023. [Online]. Available: https://www.microsoft.com/en-us/download/confirmation.aspx?id=56950
[23] A. Suresh, G. Somashekar, A. Varadarajan, V. R. Kakarla, H. Upadhyay, and A. Gandhi, "ENSURE: Efficient scheduling and autonomous resource management in serverless environments," in *Proc. IEEE Int. Conf. Autonomic Comput. Self-Organizing Syst.*, 2020, pp. 1–10.
[24] J. R. Gunasekaran et al., "Fifer: Tackling resource underutilization in the serverless era," in *Proc. 21st Int. Middleware Conf.*, 2020, pp. 280–295.
[25] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Commun. Surv. Tut.*, vol. 18, no. 1, pp. 732–794, Jan.–Mar. 2015.
[26] C. Jin, X. Bai, C. Yang, W. Mao, and X. Xu, "A review of power consumption models of servers in data centers," *Appl. Energy*, vol. 265, pp. 114806, 2020.
[27] E. Al-Masri, I. Diabate, R. Jain, M. H. L. Lam, and S. R. Nathala, "A serverless IoT architecture for smart waste management systems," in *Proc. IEEE Int. Conf. Ind. Internet*, 2018, pp. 179–180.
[28] R. F. Hussain, M. A. Salehi, and O. Semiari, "Serverless edge computing for green oil and gas industry," in *Proc. IEEE Green Technol. Conf.*, 2019, pp. 1–4.
[29] P. Singh, M. Masud, M. S. Hossain, A. Kaur, G. Muhammad, and A. Ghoneim, "Privacy-preserving serverless computing using federated learning for smart grids," *IEEE Trans. Ind. Inform.*, vol. 18, no. 11, pp. 7843–7852, Nov. 2022.
[30] S. Palacios, D. Zabrocki, B. Bhargava, and V. Aggarwal, "Auditable serverless computing for farm management," in *Proc. Int. Workshop Big Data Emergent Distrib. Environ.*, 2021, pp. 1–6.
[31] I. Pelle, F. Paolucci, B. Sonkoly, and F. Cugini, "Latency-sensitive edge/cloud serverless dynamic deployment over telemetry-based packet-optical network," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 9, pp. 2849–2863, Sep. 2021.
[32] M. Golec, R. Ozturac, Z. Pooranian, S. S. Gill, and R. Buyya, "IFaaSBus: A security-and privacy-based lightweight framework for serverless computing using IoT and machine learning," *IEEE Trans. Ind. Inform.*, vol. 18, no. 5, pp. 3522–3529, May 2022.
[33] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Gener. Comput. Syst.*, vol. 91, pp. 407–415, 2019.
[34] Z. Sun, B. Zhang, C. Gu, R. Xie, B. Qian, and H. Huang, "ET2FA: A hybrid heuristic algorithm for deadline-constrained workflow scheduling in cloud," *IEEE Trans. Serv. Comput.*, vol. 16, no. 3, pp. 1807–1821, May/Jun. 2023.
[35] X. Li, W. Yu, R. Ruiz, and J. Zhu, "Energy-aware cloud workflow applications scheduling with geo-distributed data," *IEEE Trans. Serv. Comput.*, vol. 15, no. 2, pp. 891–903, Mar./Apr. 2022.
[36] A. Marahatta, S. Pirbhulal, F. Zhang, R. M. Parizi, K.-K. R. Choo, and Z. Liu, "Classification-based and energy-efficient dynamic task scheduling scheme for virtualized cloud data center," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1376–1390, Oct.–Dec. 2021.
[37] Y. Huang et al., "Deep adversarial imitation reinforcement learning for QoS-aware cloud job scheduling," *IEEE Syst. J.*, vol. 16, no. 3, pp. 4232–4242, Sep. 2022.
[38] V. Sreekanti et al., "Cloudburst: Stateful functions-as-a-service," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2438–2452, Aug. 2020.
[39] A. Suresh and A. Gandhi, "FnSched: An efficient scheduler for serverless functions," in *Proc. 5th Int. Workshop Serverless Comput.*, 2019, pp. 19–24.
[40] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol. 62, no. 12, pp. 44–54, 2019.
[41] G. Aumala, E. Boza, L. Ortiz-Aviles, G. Totoy, and C. Abad, "Beyond load balancing: Package-aware scheduling for serverless platforms," in *Proc. IEEE/ACM 19th Int. Symp. Cluster Cloud Grid Comput.*, 2019, pp. 282–291.
[42] R. Xie, D. Gu, Q. Tang, T. Huang, and F. R. Yu, "Workflow scheduling in serverless edge computing for the industrial Internet of Things: A learning approach," *IEEE Trans. Ind. Inform.*, vol. 19, no. 7, pp. 8242–8252, Jul. 2023.
[43] A. Banaei and M. Sharifi, "ETAS: Predictive scheduling of functions on worker nodes of apache OpenWhisk platform," *J. Supercomput.*, vol. 78, no. 4, pp. 5358–5393, 2022.
[44] P. Vahidinia, B. Farahani, and F. S. Aliee, "Mitigating cold start problem in serverless computing: A reinforcement learning approach," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 3917–3927, Mar. 2023.
[45] M. S. Aslanpour, A. N. Toosi, M. A. Cheema, and R. Gaire, "Energy-aware resource scheduling for serverless edge computing," in *Proc. IEEE 22nd Int. Symp. Cluster Cloud Internet Comput.*, 2022, pp. 190–199.
[46] H. Liu et al., "Thermal-aware and DVFS-enabled big data task scheduling for data centers," *IEEE Trans. Big Data*, vol. 4, no. 2, pp. 177–190, Jun. 2018.
[47] Z. Zhou et al., "An adaptive energy-aware stochastic task execution algorithm in virtualized networked datacenters," *IEEE Trans. Sustain. Comput.*, vol. 7, no. 2, pp. 371–385, Apr.–Jun. 2022.
[48] C. Tang, C. Zhu, H. Wu, Q. Li, and J. J. P. C. Rodrigues, "Toward response time minimization considering energy consumption in caching-assisted vehicular edge computing," *IEEE Internet Things J.*, vol. 9, no. 7, pp. 5051–5064, Apr. 2022.
[49] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge Univ. Press, 2004.
[50] M. Chen and S. C.-K. Chau, *Preliminaries of Online Algorithms and Competitive Analysis*. Cham: Berlin, Germany: Springer, 2022, pp. 11–22.
[51] S. Chen, A. Jin, C. Delimitrou, and J. F. Martınez, "Retail: Opting for learning simplicity to enable QoS-aware power management in the cloud," in *Proc. IEEE 28th Int. Symp. High- Perform. Comput. Architecture*, 2022, pp. 155–168.
[52] V. M. Bhasi, J. R. Gunasekaran, P. Thinakaran, C. S. Mishra, M. T. Kandemir, and C. Das, "Kraken: Adaptive container provisioning for deploying dynamic DAGs in serverless platforms," in *Proc. ACM Symp. Cloud Comput.*, 2021, pp. 153–167.

Seyed Hamed Rastegar received the B.Sc. degree in electrical engineering from the University of Tehran, Tehran, Iran, in 2009, the M.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, in 2011, and the Ph.D. degree in electrical engineering from the University of Tehran in 2018.

He is currently a Senior Postdoctoral Researcher with the School of Computer Science, Institute for Research in Fundamental Sciences, Tehran. His research interests include computer and communication networks, edge/cloud computing, and data science.

Hossein Shafiei received the B.Sc. degree in electrical and computer engineering from Shahid-Beheshti University, Tehran, Iran, in 2006, the M.Sc. degree in computer engineering from the Iran University of Science and Technology, Tehran, in 2009, and the Ph.D. degree in computer science from the University of Tehran, Tehran, in 2014.

He is currently an Assistant Professor of computer engineering with the K. N. Toosi University of Technology, Tehran. His research interests include wireless networks, cloud computing, distributed systems, and the security and privacy of networked systems.



Ahmad Khonsari received the B.Sc. degree in electrical and computer engineering from Shahid Beheshti University, Tehran, Iran, in 1991, the M.Sc. degree in computer engineering from the Iran University of Science and Technology, Tehran, in 1996, and the Ph.D. degree in computer science from the University of Glasgow, Glasgow, U.K., in 2003.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Tehran, Tehran, and a Researcher with the School of Computer Science, Institute for Research in Fundamental Sciences, Tehran. His research interests include simulation and data analysis, performance modeling/evaluation, wired/wireless networks, cloud and distributed systems, quantum information processing, and high-performance computer architectures.