Security, Reliability, Cost, and Energy-Aware Scheduling of Real-Time Workflows in Compute-Continuum Environments

Ahmad Taghinezhad-Niar and Javid Taheri, *Senior Member, IEEE*

***Abstract*—Emerging computing paradigms like mist, edge, and fog computing address challenges in the real-time processing of vast Internet of Things (IoT) applications. Alongside, cloud computing offers a suitable platform for executing services. Together, they form a multi-tier computing environment known as compute-continuum to efficiently enhance data management and task execution of real-time tasks. The primary considerations for compute-continuum include variations in resource configuration and network architecture, rental cost model, application security needs, energy consumption, transmission latency, and system reliability. To address these problems, we propose two scheduling algorithms (RCSECH and RSECH) for real-time multi-workflow scheduling frameworks. Both algorithms optimize for rental cost, energy consumption, and task reliability when scheduling real-time workflows while considering deadlines and security requirements as constraints. RCSECH also factors in reliability alongside these constraints. The environment under investigation consists of a compute-continuum architecture consisting of mist, edge, fog, and cloud layers, each potentially composed of heterogeneous resources. The framework undergoes evaluation via simulation experiments, revealing promising results. Specifically, the framework exhibits the capability to enhance reliability by up to 7%, reduce energy consumption by 8%, surpass reliability constraints by more than 25%, and generate cost savings by at least 15%.**

***Index Terms*—Compute-continuum, real-time, reliability, security, workflow scheduling.**

I. Introduction

THE need to use computing resources as a utility has driven the growth of large-scale distributed computing systems, generating vast amounts of diverse data. Cloud computing has become the primary solution in response to this trend [1]. The rise of IoT devices has led to extensive data processing and Big Data emergence. By 2025, it is expected that about 75% of the 55.7 billion global appliances will be connected to IoT systems [2].

Manuscript received 31 January 2024; revised 30 May 2024; accepted 8 July 2024. Date of publication 10 July 2024; date of current version 6 September 2024. Recommended for acceptance by J. Catalao. *(Corresponding author: Ahmad Taghinezhad-Niar.)*

Ahmad Taghinezhad-Niar is with the Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz 5166616471, Iran (e-mail: a.taghinezhad@tabrizu.ac.ir).

Javid Taheri is with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, BT7 1NN Belfast, U.K., and also with the Department of Mathematics and Computer Science, Karlstad University, 651 88 Karlstad, Sweden (e-mail: javid.taheri@kau.se).

Digital Object Identifier 10.1109/TCC.2024.3426282

Fog computing, reducing the need for distant cloud data processing, creates a layer between IoT and cloud. It brings the cloud closer to the network edge, where IoT sensors and devices generate data [3]. Mist/edge computing adds another layer between IoT and fog to further reduce data transmission time [4]. The concept of mist/edge is to extend fog to the furthest point possible, right close to the IoT endpoint devices. This differentiation is supported by the National Institute of Standards and Technology of the USA [4].

The compute continuum integrates mist, edge, fog, and cloud computing to create a versatile computing environment accommodating diverse applications [4], [5]. Fog computing serves as an intermediary or extension of cloud computing, addressing its latency and local processing limitations while leveraging its infrastructure. Mist computing operates at the edge, while fog computing provides a broader range of functions [5]. The multi-tier environment hierarchy follows IoT, mist/edge, fog, and cloud layers, with the mist/edge layer referring to the closest layer to the edge. This architecture optimizes resource utilization and minimizes latency, integrating edge-to-cloud computing paradigms in workflow management which we are addressing workflow scheduling problems in this compute-continuum environment. This structure allows efficient data management and processing from IoT devices. For example, in a smart city, sensor data could be processed at the mist/edge layer, aggregated at the fog layer, and analyzed at the cloud layer for acquiring city-wide insights.

The expansion of technology has given rise to various workflow applications, characterized by intricate task/service interdependence modeled as Directed Acyclic Graphs (DAGs). This spectrum extends beyond scientific workflows to include applications within the Internet of Things.

Scheduling services are essential for running workflow applications with the desired Quality of Service (QoS). Real-time workflows often have time-critical aspects that demand strict reliability standards, or they may have security requirements that cannot be executed on unreliable resources. Additionally, factors such as resource leasing costs add to the scheduling complexity. Therefore, workflow scheduling is pivotal in meeting incoming application requirements and enhancing QoS. Mist/Edge and Fog resources often face limitations in terms of energy efficiency and computational power compared to cloud resources. Therefore, managing energy consumption, particularly for battery-powered IoT devices, becomes crucial [4], compounding the

complexity of the scheduling problem due to these conflicting prerequisites [4].

To this end, prior studies have addressed a limited subset of the objectives and constraints [6], [7] considered in this study. For instance, several approaches focus on scheduling single workflows or individual tasks [8] within a single computing [9], [10] environment. In contrast, our paper considers a broader set of objectives and constraints for multiple real-time workflows for compute-continuum.

A. Motivating Example

Let us consider a novel healthcare paradigm that leverages the distributed processing capabilities of a mist/edge/fog/cloud computing continuum for real-time patient monitoring. This system caters to chronic respiratory conditions like asthma or COPD, employing wearable sensors and smart home devices to gather continuous physiological data (vital signs), medication adherence information, and environmental factors (air quality, pollen count). The compute continuum enables effective scheduling:

- *Real-time edge processing* (mist/edge/fog) minimizes latency for critical data analysis (i.e., wearable sensor data can be pre-processed and filtered at the edge to reduce transmission volume and minimize processing delays)
- *Cloud-based advanced analytics* tackles computationally intensive tasks like advanced data analytics or complex pattern recognition when needed.

The objectives and challenges for scheduling considerations for this system include: 1) Strict deadlines for data analysis to enable timely interventions, 2) Reliability to prevent life-threatening system failures, 3) Data security for sensitive patient health information, 4) Energy efficiency by prioritizing the utilization of energy-efficient resource, and 5) Cost-effectiveness by leveraging the mist/edge/fog for routine tasks.

B. Contribution

This work tackles the challenge of workflow scheduling in compute-continuum environments by considering a comprehensive set of objectives and constraints. The main contributions of this paper are:

- *Compute-Continuum Environment:* Balancing between allocation of Computationally intensive tasks to powerful cloud resources, while scheduling latency-sensitive tasks to Mist/Edge and Fog resources.
- *Energy Efficiency:* Minimize energy consumption by using energy-efficient resources while meeting task requirements (reliability, security, deadlines).
- *Rental Cost Minimization:* Prioritize edge resources to reduce costs, while efficiently using cost-efficient cloud resources considering other objectives.
- *Reliability Enhancement:* Introducing a Reliability Guard and a duplication method considering energy and security constraints to ensure reliability.
- *Deadline Adherence:* Propose a method for efficiently distributing workflow deadlines among tasks to guarantee adherence to overall deadlines.
- *Task Security:* For tasks with specific security requirements defined by the user, we calculate the security needs of their intermediate tasks and execute them on reliable resources
- *Multi-Workflow Scheduling:* Use logarithmic Min-Max normalization to handle data skewness of optimization metrics to propose a balanced scheduler and utilize resource gaps for efficient task execution.

To the best of our knowledge, no prior research has designed a scheduling method that systematically incorporates all the aforementioned objectives and requirements in a compute-continuum environment.

II. Related Work

The allocation of tasks with different computation times involving data dependencies on heterogeneous resources poses a computationally challenging problem known as NP-hard [11]. This section identifies existing heuristic approaches for workflow scheduling in a compute-continuum environment, assesses their strengths and limitations, and explains research gaps that our study aims to address.

Chakravarthi et al. [11] proposed a cloud single workflow scheduling heuristic using min-max normalization during resource allocation to improve reliability within budget constraints. However, it lacks consideration for security and energy consumption, and its applicability is limited to single-workflow scenarios. Li et al. [12] proposed a scheduling model for cloud-edge environments, optimizing makespan, load balance, and energy consumption. However, they did not address reliability, security, or rental costs.

In our previous work [13], we focused on minimizing energy consumption and rental costs in cloud workflow scheduling. Subsequently, we investigated uncertainty in task execution time [6], yet we did not consider reliability, compute-continuum, and security. Expanding our research [1], we addressed reliability-aware multi-workflow scheduling in multi-cloud systems. However, we did not explore reliability-constrained workflows, compute-continuum architectures, and security awareness, which are the focus of this study.

There are also studies that focused on scheduling for compute-continuum environments. Attiya et al. [14] aimed to optimize cost, energy consumption, and makespan in the cloud for individual task scheduling, overlooking workflow applications, security, reliability, and the compute-continuum environment. Khaleel [15] proposed reliability- and energy-aware scheduling for fog-cloud environments but overlooked rental cost and security concerns. Their subsequent work [3] focused on scheduling reliability and workflow makespan but did not consider rental cost, security, and edge environments. Shukla et al. [16] proposed a meta-heuristic workflow scheduling for Fog-Cloud focusing on makespan, cost, and energy. However, security, reliability, and real-world challenges are not addressed.

While security-sensitive scheduling has gained recent attention, existing approaches often prioritize only one or two objectives (e.g., [4], [8], [10], [17]) or are limited to single-workflow or independent task scheduling or single computing environments [8], [17]. Security challenges of workflow

TABLE I
OVERVIEW OF THE EXISTING LITERATURE AND OUR PAPER

Objectives	Researches	Our study
Time-factor	[4], [7], [8], [9], [10], [14], [16], [20]	Yes
Reliability	[3], [10], [13]	Yes
Lease-cost	[4], [6], [7], [9], [13], [16], [17]	Yes
Multi-workflow	[3], [4], [6], [10]	Yes
Energy	[4], [6], [7], [13], [16], [20], [21]	Yes
Security	[4], [7], [10], [17], [19]	Yes
Cloud tier	[3], [7], [8], [14], [16], [17]	Yes
Fog tier	[3], [4], [7], [8], [14], [16], [19], [21]	Yes
Mist/Edge tier	[4], [22]	Yes

scheduling are discussed in [18]. Chen et al. [17] propose a cost- and makespan-aware workflow scheduling method for security-sensitive tasks in clouds. Their security model assigns tasks based on a user-defined security scale (0-1), but selecting this scale remains a challenge. Javanmardi et al. [19] proposed a security-aware workflow scheduling solution, neglecting energy, reliability, and rental costs. Alam et al. [10] proposed a security-aware workflow scheduling algorithm in the cloud, considering factors such as makespan and fault tolerance. However, they did not consider compute-continuum, energy, rental cost, or reliability.

Stavrinides et al. [4] proposed a heuristic considering energy awareness and security but did not address execution reliability, which is crucial for reducing task failures. While their security model assigns the highest security level to resources closest to the network edge, it suffers from impracticality due to the inherent difficulty of knowing the real-world security state of resources. In this paper, we propose a model where security classifications are randomly assigned to resources across all tiers, following a normal distribution pattern. This approach acknowledges the inherent uncertainty in real-world security state and avoids assigning unrealistically high-security levels to edge resources by default.

Ali et al. [8] proposed a scheduler for independent tasks in fog-cloud computing but overlooked workflow inter-dependencies and reliability constraints. Their security model categorizes tasks based on different security labels for fog and cloud resources. Their model can not be used directly for workflow scheduling due to classified security dependencies of workflow tasks.

Our paper is specifically designed to schedule multiple workflows with deadline, security, and reliability constraints, operating with the objectives of rental cost, and energy consumption within a compute-continuum environment (Mist/Edge, Fog, Cloud). Table I organizes and positions the related studies currently available in relation to our proposed methodology.

III. PROBLEM FORMULATION AND MODELING

In this section, the initial focus is on illustrating the compute-continuum resource and workflow application models pertaining to the proposed scheduling frameworks, which constitute fundamental components of this study.

A. Modeling Compute-Continuum Environment

This paper explores a compute-continuum environment (also referred to as mist/edge, fog, cloud) for workflow applications. IoT devices such as sensors or smart appliances send data to the mist/edge layer for initial processing. The mist/edge layer is the first (closest) resource tier to the devices and provides localized processing power, storage, and communication bandwidth. The second tier, the fog resources, offers intermediate-level processing and storage. The third tier, the cloud resources, provides abundant but remote computing resources, leading to higher latency due to longer communication distances.

We categorize our compute-continuum environment into three tiers, denoted as M for mist/edge, F for Fog, and C for Cloud. Each tier, represented by i (where $i \in \{C, M, F\}$, encompasses a distinct set of resource types, denoted as RT^i. These resources form a pool $RP = \{r_1^i, r_2^i, \ldots, r_P^i\}$, containing P different resources, including processing power, storage capacity, and communication bandwidth. Their availability and characteristics vary between tiers. Cloud resources follow a pay-per-use model (adhering to the Amazon EC2 pricing scheme) that is time-based and calculated per hour. Even a fraction of an hour's usage incurs a charge for the entire hour [1], [7]. Each cloud resource type in r_j^C has an associated cost per hour.

Each physical host in the mist/edge, fog, and cloud layers has a multi-core processor with identical physical cores. Each resource is allocated a vCPU operating at a frequency of f_i, where f_i is the vCPU's operating frequency in each resource. Each vCPU corresponds to a physical core in the underlying physical host's processor and maintains its own task-processing queue. Mist/edge resources are limited in number and capacity compared to fog resources, and fog resources are less abundant than cloud resources. This results in a computing power hierarchy, denoted by $f_i^M < f_i^F < f_i^C$.

A workflow application is denoted as $G^z = (T^z, D^z, \Gamma^z, St^z, Dl^z)$, where tasks are represented by T^z and task dependencies by $D^z = \{d^z_{(p,s)} | (t_p^z, t_s^z \in T^z)\}$. The reliability requirement, start time, and deadline of an application z are defined by Γ^z, St^z, and Dl^z, respectively.

$T^z = \{t_1^z, t_2^z, \ldots, t_N^z\}$ represents a collection of indivisible, independent, and non-preemptive tasks. Each task $t_s^z \in T^z$ is defined as $t_s^z = (id, ti, sr, in, ot)$. Here, id is the unique task identifier, ti is the task length in terms of instruction number, sr is the task's security requirement (private (pr), semi-private (sp), or public (pb)), and in and ot specify the task's input and output data size, respectively.

A direct relationship between two tasks (t_p^z, t_s^z) in a workflow is indicated by $d^z_{(p,s)} \in D^z$. Here, $pre(t_s^z)$ denotes the immediate predecessors of task t_s^z, while $suc(t_p^z)$ represents the immediate successors of task t_p^z.

A workload refers to a collection of application DAGs submitted for scheduling. In (1), a workload is represented as W, where z symbolizes a workflow application and $|G|$ signifies the total number of workflows in a workload.

$$W = \bigcup_{z=1}^{|G|} G^z \tag{1}$$

1) Network and Communication Model: The mist/edge, fog, and cloud tiers of a virtual network are interconnected. We represent the speed of data transmission from the IoT to the mist/edge, fog, or cloud layer using the symbol $\tau_{\mathrm{I/L}}$, where L stands for the specific layer (mist/edge, fog, or cloud). This data transfer rate is uniformly distributed within a specific range. The heterogeneity of the network connecting the IoT and the corresponding tier is represented by $H_{\mathrm{I/L}}$, while $\hat{\tau}_{\mathrm{I/L}}$ denotes the average data transfer rate between the two tiers. Initially, when an application is submitted for scheduling, data is transferred from the IoT application to the corresponding tier via the intermediate tiers. This relationship is represented in (2) [4].

$$\hat{\tau}_{\mathrm{I/M}} > \hat{\tau}_{\mathrm{I/F}} > \hat{\tau}_{\mathrm{I/C}} \tag{2}$$

The rate at which data is transferred between two resources, whether they are in the same tier or different tiers, is uniformly distributed within a certain range defined in (3) [4].

$$\tau_{\mathrm{I/L}}^{k,j} \sim U\left(\hat{\tau}_{\mathrm{I/L}} \cdot \left(1 - \frac{H_{I/L}}{2}\right), \hat{\tau}_{\mathrm{I/L}} \cdot \left(1 + \frac{H_{I/L}}{2}\right)\right) \tag{3}$$

The pair I/L is a member of the set $\{I/M, I/F, I/C, M/M, M/F, M/C, F/F, F/C, C/C\}$. The parameters $\hat{\tau}_{\mathrm{I/L}}$ and $H_{\mathrm{I/L}}$ denote the average data transfer rate and the degree of network heterogeneity between the specific tiers, respectively.

A dedicated node within the mist/edge tier houses a resource manager, whose duty is to assign and coordinate incoming workflows throughout the mist/edge, fog, and cloud tiers. This manager oversees a universal waiting queue, which holds tasks from all workflows entering the system until they are prepared for resource scheduling.

2) Task Computation and Communication Time: The computation of task execution time and resource energy are directly linked. As a result, the time it takes to execute a task varies across heterogeneous resources. The execution time for each task on a resource, denoted as r_j^i, given the frequency of that resource (f_j^i), is detailed with CP in (4). In this equation, ti_s^z denotes the number of clock cycles necessary for the execution of all instructions in a task.

$$CP_{r_j^i}^{t_s^z} = \frac{ti_s^z}{f_j^i} \tag{4}$$

Eq. (5) describes the time required to transfer files from t_p^z to t_s^z. In this equation, $FS_{t_p^z}^{t_s^z}$ represents the size of the files that need to be transferred. This equation calculates the time required to transfer files by dividing the size of the files by the data transfer rate (bandwidth) between the two tiers.

$$CM_{t_p^z}^{t_s^z} = \frac{FS_{t_p^z}^{t_s^z}}{\tau_{\mathrm{I/L}}^{k,j}} \tag{5}$$

If two tasks are executed on the same tier within the same type of resource, the data transmission time between them is considered negligible. For convenience, this time is set to zero.

The earliest start time of task t_s^z on a resource r_j^i is detailed in (6). In this equation, $RT(r_j^i)$ represents the ready time of the resource to begin executing a task. The term EFT in (7) is defined to calculate the anticipated completion time of a task within a compute-continuum environment.

$$EST_{r_j^i}^{t_s^z} = \max\left\{RT(r_j^i), \max_{t_p \in pre(t_s^z)}\left\{EFT_{r_z^q}^{t_p^z} + CM_{t_p^z}^{t_s^z}\right\}\right\} \tag{6}$$

$$EFT_{r_j^i}^{t_s^z} = EST_{r_j^i}^{t_s^z} + CP_{r_j^i}^{t_s^z} \tag{7}$$

The term LFT in (8) is defined to compute the latest time at which a task can be completed in the provided environment. Here, $\hat{r}$ represents the average amount of resource (computation and communication) that could be made available to execute tasks. Hence, the value of $LFT^{t_{end}^z}$ reflects the deadline for the workflow application.

$$LFT_{\hat{r}}^{t_i^x} = \min_{t_s \in suc(t_i)}\left\{LFT_{\hat{r}}^{t_s^x} - CM_{t_p^z}^{t_s^z} - CP_{\hat{r}}^{t_j^z}\right\} \tag{8}$$

3) Resource Rental Cost of Task Execution: Each cloud service provider is linked to a distinct organization and follows its specific billing procedures (e.g., per minute or hour). It is reasonable to assume that utilizing resources with greater computational capabilities will result in higher costs. To capture this assumption in our model, we can apply different billing procedures. The cost associated with renting a cloud resource, denoted as r_j^C, for task execution is determined by the number of 'Charge Time (CT)' required and the rental price of the resource in use. The definitions of CT and rental cost of tasks are provided in (9) and (10), respectively. In these equations, $CI(r_j^C)$ represents the charge interval for different cloud resources and the term $LC(r_j^C)$ signifies the lease cost of cloud resource per CI.

$$CT_{r_j^i}^{t_s^z} = \left\lceil \frac{\max(EFT_{r_j^i}^{t_i^z}, RT(r_j^i))}{CI(r_j^i)} \right\rceil - \left\lceil \frac{RT(r_j^i)}{CI(r_j^i)} \right\rceil \tag{9}$$

$$C_{r_j^i}^{t_s^z} = CT_{r_j^i}^{t_s^z} \times LC(r_j^i) \tag{10}$$

The workflow's makespan, represented as S, for the workflow application z is outlined in (11). Because we update/replace the EFT of each task with its actual finish time after its execution, the difference between the EFT of the last task of G^z with its start time (St^z) will show the actual execution time of G^z.

$$S^z = \max_{t_s^z \in G^z}\{EFT_{r_j^i}^{t_s^z}\} - St^z \tag{11}$$

4) Energy Consumption Model: The energy consumption of processors, which are identified as the primary energy-consuming components in this paper [4], can be managed through software-based strategies and are intrinsically linked to the operation of applications.

The energy usage of a multi-core processor under full load is commonly denoted by its Thermal Design Power (TDP), provided by the processor's manufacturer, and is measured in watts (W) [4]. For processors with uniform physical cores, the maximum energy consumption of a single core, and consequently of the associated resource, can be estimated using (12). In this equation, σ_j represents the number of physical cores of

the processor (P_j).

$$P_j^{\max} = \frac{TDP_j}{\sigma_j} \tag{12}$$

Assuming each resource represents a processor core, its power consumption in an idle state drops significantly compared to its maximum load. This is a common characteristic of CMOS circuits. As a result, the power consumption of a processor's physical core when idle can be approximated using (13) [4], where GL represents the power consumption when the processor is idle.

$$P_j^{\min} = P_j^{\max} \cdot GL \tag{13}$$

As a consequence, the cumulative power usage of a resource (i.e., physical core) over a time span t, can be determined using (14), where UT_j denotes the percentage utilization of the resource at a specific time. Therefore, the energy usage of the core over a period of time can be computed using (15) [4], [13].

$$P_j(t) = P_j^{\min}(t) + \left(P_j^{\max} - P_j^{\min} \cdot (UT_j)\right)(t) \tag{14}$$

$$P_m^i = \int_{t_0}^{t_1} P_m^i(t) \cdot dt \tag{15}$$

Hence, we are able to calculate the energy usage of a task executed on a resource denoted as r_j^i on Processor P_m using (16).

$$E_{r_j^i}^{t_s^z} = \int_{t_0}^{t_1} P_m(t) \cdot dt \tag{16}$$

5) System Reliability Model: Reliability is crucial in compute-continuum systems (mist/edge/cloud) for seamless workflow execution over time [13]. These environments encompass heterogeneous resources (VMs/containers with varying capabilities, software applications, and management strategies), leading to potential variations in reliability for workflow execution. As suggested in [9], we use the Weibull distribution to manage task execution reliability in compute-continuum resources, which can model variable failure rates of resources instead of the less accurate exponential distribution that is commonly used for reliable scheduling (in which resources only have fixed fault rates). It includes parameters: $R(t)$ (reliability), θ (average time to failure), and β (slope parameter indicating resource fault type). β can simulate various failure rates. Our model is based on the failure rate ($\lambda = \frac{1}{\theta}$), facilitating the transformation of (17) into (18) [9].

$$R(t) = e^{-\left(\frac{t}{\theta}\right)} \tag{17}$$

$$R(t) = e^{-(\lambda \times t)^{\beta}} \tag{18}$$

In this context, the parameters of the Weibull distribution pertaining to the compute-continuum resources are denoted by:

$$\Lambda = \{(\lambda_{r_1^i}, \beta_{r_1^i}), (\lambda_{r_2^i}, \beta_{r_2^i}), \ldots, (\lambda_{r_P^i}, \beta_{r_P^i})\}.$$

Similarly, the Weibull distribution parameters that correspond to the communication network between resources are represented as:

$$\psi = \{(\lambda_{r_1^i}, \beta_{r_1^i}), (\lambda_{r_2^i}, \beta_{r_2^i}), \ldots, (\lambda_{r_P^i}, \beta_{r_P^i})\}.$$

Therefore, relying on (17), we can define the reliability of the execution and communication of task t_s^z on resource r_j as outlined in (19).

$$R_{r_j}^{t_s^z} = RC(d_{(p,s)}^z) \cdot RX_{r_j}^{t_s^z} \tag{19}$$

In this scenario, $RC(d_{(p,s)}^z)$ denotes the reliability of the communication edge that exists between the resources of multiple tiers. This can be determined using (20), where r_p and r_s are the resources executed task t_p^z and t_s^z, respectively. In addition, $RX_{r_j}^{t_s^z}$ defined in (21) is reliability of task execution on the resource.

$$RC(d_{(p,s)}^z) = e^{\left(\frac{CM_{t_p}^{t_s}}{\lambda_{(r_p,r_s)}}\right)^{\beta_{(r_p,r_s)}}} \tag{20}$$

$$RX_{r_j}^{t_s^z} = e^{\left(\frac{CP_{r_j}^{t_s^z}}{\lambda_{r_j}}\right)^{\beta_{r_j}}} \tag{21}$$

Combining all equations leads to (22), which represents the reliability of a task for both computation and communication in a compute-continuum environment.

$$R_{r_j}^{t_s^z} = e^{\left(\frac{CM_{t_p}^{t_s}}{\lambda_{(r_p,r_j)}}\right)^{\beta_{(r_p,r_j)}}} \cdot e^{\left(\frac{CP_{r_j}^{t_s^z}}{\lambda_{r_j}}\right)^{\beta_{r_j}}} \tag{22}$$

By definition, the reliability of a task refers to the probability of completing the task without any failures. Thus, for tasks with low execution reliability, task duplication can be employed (as a strategy) to enhance their reliability. To this end, the reliability of a task that is duplicated on two specific resources, namely r_m and r_j for parallel execution, can be defined in (23) [1].

$$R_{(r_m,r_j)}^{t_s^z} = 1 - \left((1 - R_{r_m}^{t_s^z})) \times (1 - R_{r_j}^{t_s^z})\right) \tag{23}$$

Hence, the reliability of a workflow application (consists of $|T^z|$ tasks) can be described as (24), as the multiplication of the reliability of all its tasks.

$$R(G^z) = \prod_{s=1}^{|T^z|} R_{r_j}^{t_s^z} \tag{24}$$

6) Security Model: In this paper, task Security Requirement (SR) is categorized into three distinct levels, denoted as $SR: \{pu, sp, pr\}$. The first level, 'pu' for public security requirement is the most basic level, implies that data can be processed on any available resource. The second level, 'sp' for semi-private security, stipulates that data processing can only take place on resources that are labeled as semi-reliable or reliable. The third and highest level, 'pr' for private security requirement, mandates that data processing is exclusively allowed on resources that are classified as reliable.

Upon scheduling a task, it is necessary to check if a resource is considered reliable for each task based on the varying security requirements. This verification aligns with the mapping rules specified in (25). Here, each resource is assigned a security level,

and $SR(t_s^z)$ denotes the security requirement of the task t_s^z.

$$RR(t_s^z, r_j^i) = \begin{cases} 1 & \text{if } SR(t_s^z) \in pu \ \& \ r_j^i \in \{pr, sp, pu\} \\ 1 & \text{if } SR(t_s^z) \in sp \ \& \ r_j^i \in \{sp, pr\} \\ 1 & \text{if } SR(t_s^z) \in pr \ \& \ r_j^i \in \{pr\} \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

The security requirements for entry tasks in a workflow are initially specified. Subsequently, the security requirements for non-entry tasks are determined by assessing the maximum security demand of their preceding tasks. If we assume $pr = 3$, $sp = 2$, and $pu = 1$, we can define the security level of a task using (26).

$$SR(t_s^z) = \max_{t_p^z \in \text{pre}(t_s^z)} \{SR(t_p^z)\} \tag{26}$$

7) Problem Formulation: The main objectives of our paper are minimizing energy consumption, and rental cost and maximizing reliability defined in (27).

$$\begin{aligned} &\text{Minimize:} \sum_{t_i^z \in T^z} \sum_{r_j \in RP} C_{r_j}^{t_i^z} \cdot X_{ij} \\ &\text{Minimize:} \sum_{t_i^z \in T^z} \sum_{r_j \in RP} E_{r_j}^{t_i^z} \cdot X_{ij} \\ &\text{Maximize:} \sum_{t_i^z \in T^z} \sum_{r_j \in RP} R_{r_j}^{t_i^z} \cdot X_{ij} \end{aligned} \tag{27}$$

Subject to the following constraints

$$\sum_{r_j \in RP} X_{ij} \cdot RR(t_i^z, r_j) \geq 1, \quad \forall t_i^z \in T^z \tag{28}$$

$$EFT_r^{t_{exit}^z} \leq Dl^z \tag{29}$$

$$R(G^z) \geq \Gamma^z, \quad \forall G^z \in W \tag{30}$$

$$\sum_{j \in R} X_{ij} \leq 1 \quad \forall t_i \in T^z \tag{31}$$

$$X_{ij} \in \{0, 1\} \tag{32}$$

Eq. (28) guarantees the fulfillment of the security requirement for each task. (29) ensures that every workflow adheres to its designated deadline. (30) mandates the adherence to the reliability constraint for each workflow. Eq (31) ensures that each task can only be assigned to a maximum of one resource. Whenever a task is duplicated, it is treated as a distinct task. The binary decision variable X_{ij} defined in (32) is set to 1 when task t_i is scheduled on resource r_j. In the proposed algorithms, only the RCSECH algorithm adheres to the reliability constraint specified in (30).

IV. Proposed Scheduling Heuristics

This section details the proposed workflow scheduling framework for a compute-continuum (multi-tier) system. The process begins when a workflow application enters the application queue. The application analyzer processes parameters such as LFT, EST, and EFT based on task dependencies and deadline constraints. The scheduler then calculates/distributes both deadline and security requirements among workflow tasks, following by relocating the prepared task to the TaskQueue. A prepared task is defined as a task for which its parent task has been scheduled. The scheduler allocates tasks considering security, reliability, energy efficiency, rental cost, and deadline constraints.

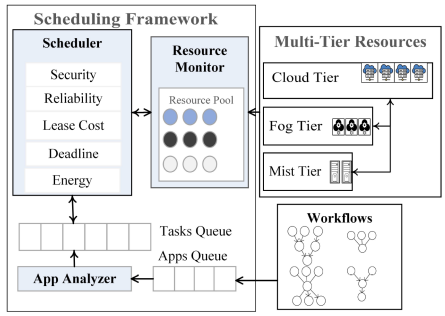

Fig. 1. Proposed workflow scheduling framework architecture.

To this end, we proposed two algorithms to schedule tasks: 'Reliability-Constrained, Security, Energy and Cost-aware Heuristic' (RCSECH) and 'Reliability, Security, Energy and Cost-aware Heuristic' (RSECH). RCSECH treats reliability as a constraint, while RSECH treats reliability only as an objective. Both algorithms are designed to optimize scheduling decisions based on their specific criteria.

A. Overview

The proposed scheduling architecture, shown in Fig. 1, connects a compute-continuum environment with workflow applications. The scheduler uses a heuristic to utilize compute-continuum resources for real-time workflow task completion, considering deadline, security, and reliability constraints.

B. Workflow Applications' Arrival

Dynamic submitted workflows with specific constraints arrive at any time. The analyzer calculates security requirements, deadlines, and reliability guards for each task. Upon task scheduling, new tasks are submitted to the queue. The scheduler, considering multiple objectives, selects resources from available types across all tiers.

C. Tasks Deadlines

We propose a deadline distribution method inspired by LFT [4], [11], and our prior work [13]. Initially, two dummy tasks, positioned at the start and the end of the workflow, are added to the workflow graph. Tasks are then partitioned into various levels; that is, organizing them into distinct levels, ensuring that tasks within the same level are independent of each other.

The graph level of each task is determined by (33). Here, we define the set as $GL = 1, 2, \ldots, V$, where the start-task (a dummy task) of the workflow is allocated to GL_V, and the end-task (also a dummy task) is assigned to GL_1. The set representing the tasks of a particular GL is delineated in (34).

Algorithm 1: Distributing Tasks Deadline of a DAG.

Input: a DAG G^z
Output: Dl of each tasks of G^z
1 Assigning tasks of G^z into their specified level according to Eq. (33);
2 Calculate Θ^z for arriving G^z based on Eq. (35);
3 Calculate $LFT_{\hat{r}}^{t_i^z}$ for all tasks of the workflow;
4 **foreach** $GL_e^z : G^z$ **do**
5 | Determine $\Theta(GL_e^z)$ based on Eq. (36);
6 **end**
7 **foreach** $GL_e^z : G^z$ **do**
8 | Calculate $Dl(GL_e^z)$ based on Eq. (37);
9 **end**
10 **foreach** $t_s^z : T^z$ **do**
11 | Calculate $Dl(t_s^z)$ based on Eq. (38);
12 **end**

The cumulative execution time for each workflow application can be computed using (35).

$$GL(t_s^z) = \begin{cases} \max\limits_{t_s^z \in suc(t_s^z)} \{GL(t_s^z)+1\}, & \text{otherwise} \\ 1, & t_s^z = t_{end}^z. \end{cases} \tag{33}$$

$$GL_e^z = \{t_s^z | GL(t_s^z) = e\} \tag{34}$$

$$\Theta^z = \sum_{i=1}^{|T^z|} CP_{\hat{r}}^{t_s^z} \tag{35}$$

Following this, (36) is utilized to ascertain the fraction of the level in the workflow's overall execution time, represented as $\Theta(GL_e^z)$. A deadline is established for each GL_e^z by (37). By progressing from GL_V^z to GL_1^z (top to bottom levels), each level deadline is computed. The initial value of GL_1^z is assigned as the arrival time of the application.

This deadline distribution method sets strict deadlines near the start-task. However, the LFT-based method [11] is more relaxed near the start-task and stricter near the end-task. To harmonize these approaches, we compute an average of the results, as articulated in (37), to set task deadlines. Algorithm 1 outlines our proposed deadline distribution method, wherein task deadlines are determined using (38).

$$\Theta(GL_e^z) = \sum_{t_s^z \in GL_e^z} CP_{\hat{r}}^{t_s^z} \tag{36}$$

$$Dl(GL_e^z) = Dl(GL_{(e+1)}^z) + \left(\Theta(GL_e^z) \cdot \frac{D^z - A^z}{\Theta^z}\right) \tag{37}$$

$$Dl(t_s^z) = \frac{Dl(GL(t_s^z)) + LFT_{\hat{r}}^{t_s^z}}{2} \tag{38}$$

Fig. 2 depicts the process of assigning GL and parameters to a workflow DAG, wherein security requirements and deadlines are distributed based on LFT and $Dl(GL)$ and proposed Dl. The illustration also showcases the application of the proposed deadline distribution method. Specifically, the figure represents communication time on the communication edges, SR, and computation time on the task nodes. Additionally, the figure includes visual representations of GL, $DL(GL)$, LFT, and the proposed Dl for each node.

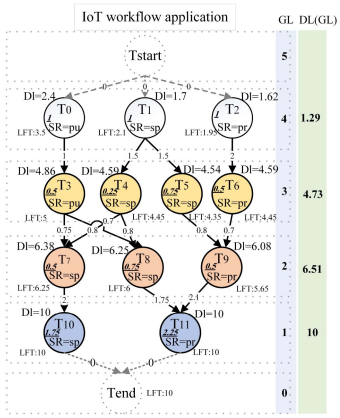

Fig. 2. Security and Deadline Distribution on a Workflow Application.

D. Reliability Guard

When reliabilities of specific tasks are calculated to be very low (i.e., w.r.t. the distributed reliability), the overall application reliability could potentially violate its reliability constraint. To alleviate this, we propose a duplication method in the following section to ensure the required reliability.

In (39), we introduce the concept of Reliability Guard (RG) for a workflow application. Here, $SCD(T^z)$ denotes the number of scheduled tasks of T^z at the scheduling time, $|T^z|$ represents the total number of tasks, and Γ^z signifies the reliability requirement of the application G^z.

$$RG^z = \begin{cases} \Gamma^{z\left(\frac{1}{|T^z|}\right)} & \text{if } SCD(G^z) = 0 \\ \dfrac{\Gamma^z}{R\left(SCD(G^z)\right)} & \text{Otherwise} \end{cases} \tag{39}$$

According to (24), the reliability of a workflow is determined by multiplying the reliabilities of its constituent tasks. In (39), we compute a Reliability Guard for each task. If a task's reliability falls below this value, it signals that the required reliability cannot be achieved, indicating a need to enhance the reliability of that specific task.

In Fig. 2 for example, assume the required reliability of 0.95 is sought; in this case, RG^z would be 0.9957. If we presume that all tasks, except t_{11}, are scheduled with a graph reliability of 0.97, then the reliability guard would be 0.979.

E. Scheduler Algorithm

Normalization is a framework (or a method) for integrating metrics into workflow scheduling algorithms. It enables efficient optimization, simplifies comparison, and aids decision-making. Challenges arise from sensitivity to parameters, metric distribution assumptions, and complexity. However, normalization can enhance scheduling algorithms in practical deployment. In our method, we tailor normalization techniques, employing logarithmic normalization for reliability and rental costs, and

min-max normalization for energy considerations to address skewed distributions, thereby ensuring fine-grained control over the optimization process.

We utilize additional methods such as deadline/security distribution and duplication to enhance reliability and refine the task/resource set, thus optimizing our normalization approach. Tasks are prioritized based on earliest deadlines (Earliest Deadline First (EDF)), and then allocated to accessible compute-continuum resources using the scheduler. Logarithmic Min-Max Normalization is applied to address skewed distributions and assign higher values to fog and mist/edge layer resources, while standard min-max normalization is utilized for energy considerations.

Eqs. (40), (41), and (42) identify locally optimal resources for cost, energy, and reliability, respectively, with a small constant added to prevent division by zero when the max and min values are the same. Before assessing normalized variables, we determine maximum and minimum values for resource cost, energy, and reliability across all available resources from all tiers. The weight parameters of cost, energy, and reliability are denoted by $\omega_{cs}, \omega_{es}, \omega_{rs}$, respectively.

$$CS_{r_j^i}^{t_s^z} = \frac{\log(C_{max}^{t_s^z} - C_{r_j^i}^{t_s^z} + 1)}{\log(C_{max}^{t_s^z} - C_{min}^{t_s^z} + 1)} \cdot \omega_{cs} \tag{40}$$

$$ES_{r_j^i}^{t_s^z} = \frac{E_{max}^{t_s^z} - E_{r_j^i}^{t_s^z} + 1}{E_{max}^{t_s^z} - E_{min}^{t_s^z} + 1} \cdot \omega_{es} \tag{41}$$

$$RS_{r_j^i}^{t_s^z} = \frac{\log\left(R_{r_j^i}^{t_s^z} - R_{\min}^{t_s^z} + 1\right)}{\log\left(R_{max}^{t_s^z} - R_{\min}^{t_s^z} + 1\right)} \cdot \omega_{rs} \tag{42}$$

Lastly, the objective metric of the schedule, which harmonizes three assessment metrics (monetary cost, reliability, and energy consumption), is shown in (43). The resource that exhibits the most favorable objective metric, according to (43), is chosen for the execution of the task.

The computation of the objective value for the allocated resources commences with the identification of an idle gap that fits the task. The term ω_s is a constant that is used to determine the impact and significance of various resources in our compute-continuum environment. Here, CI represents the impact of the cloud, FI denotes the impact of the fog, and MI signifies the impact of the mist/edge. The range of these values can be (0,1].

$$S_{r_j^i}^{t_s^z} = \omega_s \cdot \left(CS_{r_j^i}^{t_s^z} + RS_{r_j^i}^{t_s^z} + ES_{r_j^i}^{t_s^z}\right) \tag{43}$$

$$\omega_s = \begin{cases} CI & \text{if } r_j^i \in r^C \\ FI & \text{if } r_j^i \in r^F \\ MI & \text{if } r_j^i \in r^M \end{cases} \tag{44}$$

Upon scheduling a workflow application (i.e., scheduling the dummy t_{end}), the duplication algorithm for G^z is invoked to ensure the required reliability. If not met, the algorithm examines tasks in the Possible Duplication Queue (PDQ^z) with reliability below RG^z, sorted by precedence and EDF order. These tasks are assigned to the first idle gap of available resources that fits the task and meets the security demand of the task with the lowest energy consumption ($E_{r_j^i}^{t_s^z}$), continuing until G^z meets reliability requirements. The overall reliability is recalculated using (23) and updated using (45).

$$R(G^z) = R(G^z)/R_{r_j^i}^{t_s^z} \cdot R_{(r_j^i, r_j^l)}^{t_s^z} \tag{45}$$

The task scheduling portion of RCSECH is outlined in Algorithm 2. The algorithm initially arranges the tasks in the TaskQueue in ascending order based on the EDF. The steps outlined in lines 3-6 detail the process of inspecting provisioned resources for the task t_s^z. A resource is only considered for task scheduling if it can complete the task prior to its deadline and meets the task's reliability requirements, as indicated in lines 4-5. If no resource is found that can meet the task deadline, as indicated in lines 7-9, the task is then scheduled on the resources with the shortest finish time that also fulfills the task's reliability requirement based on (25). Following this, the task is assigned to the resource that exhibits the maximum value of (43). If the task is associated with a resource type and has not been provisioned yet, it is provisioned initially, and then the task is scheduled on the resource.

Algorithm 3 illustrates the duplication process when a task is sent for scheduling. When all tasks of G^z are scheduled, the duplication process starts a loop by popping tasks from (PDQ^z) and duplicating them. This process ends if the list gets empty or the total reliability of workflow G^z is satisfied. During duplication, a candidate resource for task replication is selected, as mentioned previously.

The proposed RSECH does not incorporate our duplication algorithm; and thus, it does not include lines 12 to 18 of Algorithm 2, nor does it utilize Algorithm 3.

F. Analysis of Time Complexity

The computation of task relationships, Tasks' EST, EFT, and the identification of levels (GL) necessitate a time complexity of $O(T^2)$ upon the arrival of a workflow. The distribution of deadlines demands a complexity of $O(2 \cdot T)$. For task scheduling, the TaskQueue is initially sorted, requiring a time complexity of $|T| \log |T|$. Subsequently, a heuristic is employed to examine the maximum objective metric for each resource. In the worst-case scenario, $|T|$ resources are provided. When employing the task duplication method under these conditions, the system must manage a maximum of $|T|$ tasks in the worst-case scenario. The process of finding the resource that consumes the least energy would have a time complexity of $O(|T|^2)$. Additionally, the process of identifying gaps in a resource incurs a time complexity of $O(\phi)$, where ϕ represents the quantity of gaps in the resource. The methods proposed in this study exhibit an overall time complexity on the order of $O(|T|^3)$ for each workflow. This complexity is known to be suitable for workflow scheduling heuristics [1], [9].

V. Experiments

In this section, we discuss the evaluation of our proposed algorithms (RCSECH and RSECH) using metrics and the simulation environment. We compare our solutions with two contemporary

Algorithm 2: RCSECH Heuristic.

Input: $TaskQueue$,RP
Sort the 'TaskQueue' using the EDF method.;
foreach $t_s^z \in TaskQueue$ **do**
  foreach $r_j^i \in RP$ **do**
    if $RR(t_s^z, r_j^i) < 0$ ***or*** $EFT_{r_j^i}^{t_i^z} > Dl(t_s^z)$ **then**
      continue;
    end
    Calculate $S_{r_j^i}^{t_s^z}$ based on Eq. (43);
  end
  if *no resource meets the specified conditions* **then**
    Schedule t_s^z to the fastest resource which satisfies $RR(t_s^z, r_j^i) \geq 1$;
  else
    Identify the resource with the maximum value of $S_{r_j^i}^{t_s^z}$ and schedule t_s^z to it ;
  end
  if $R_{r_j^i}^{t_s^z} < RG^z$ **then**
    Add t_s^z to PDQ^z;
  end
  Delete t_s^z from $TaskQueue$;
  if $t_s^z == t_{end}^z$ ***and*** $R(G^z) < \Gamma^z$ **then**
    Call Duplication-Algorithm(PDQ^z);
  end
end

Algorithm 3: Duplication-Algorithm.

Input: PDQ^z
foreach $t_s^z : PDQ^z$ **do**
  if $R(G^z) \geq \Gamma^z$ **then**
    return;
  end
  Duplicate t_s^z and schedule it on an available resource r_j^i which $RR(t_s^z, r_j^i) == 1$ and have the lowest energy consumption;
end

algorithms: Hybrid-MCD [3] and SCEAH [4]. While Workflow Management Systems (WMS) like Pegasus coordinate distributed workflow applications, simulation tools are essential for evaluating new algorithms. This is mostly because WMS systems often lack direct control over lower-level parameters needed for complex algorithms [4], [9], [23]. To address this, we have developed a simulator inspired by Workflowsim. Our scheduling methodologies, implemented in Java, were executed on an Intel Core i7 4712HQ processor with 16 GB of RAM.

A. Experiment Configurations

In this study, we evaluated the performance of studied scheduling heuristics using real-world workflow applications: Sipth, Ligo, Epigenomics, Montage, and Cybershake. The study conducted in [24] examines the features and attributes of these workflows. Our algorithms were tested with scientific workflows

TABLE II
SUMMARY OF FREQUENTLY USED NOTATIONS

Symbols	Description
r_j^i	Resource j in computing tier i
$SR(t_s^z)$	Security requirement of a task
$RR(t_s^z, r_j^i)$	Resources requirement satisfaction of a task
Γ^z	Reliability requirement of workflow z
$RC(d_{(p,s)}^z)$	Reliability of the task communication
RG^z	Reliability Guard of workflow z
$R_{r_j^i}^{t_s^z}$	Reliability of a task on a resource
$R_{(r_m,r_j)}^{t_s^z}$	Reliability of a duplicated task on two resources
$\tau_{I/L}$	Data transmission speed from IoT to tier L
$H_{I/L}$	Heterogeneity of the network from IoT to tier L
TDP_j	Thermal Design Power of processor j
P_m^i	Energy usage of r_j^i processor on a period of time
$E_{r_j^i}^{t_s^i}$	Energy consumption of task on a resource
$Dl(t_s^z)$	Calculated deadline of a task
$C_{r_j^i}^{t_s^z}$	Resource rental cost of a task
$CP_{r_j^i}^{t_s^i}$	Execution time of task on resource
$CM_{t_p^z}^{t_s^z}$	Time to transfer files between tasks

TABLE III
MIST/EDGE, FOG, AND CLOUD RESOURCES

Processor	f	TDP	Cores	$P^{\max}$	$P^{\min}$	LC
Atom C3436L	1.5	11	4	1.5 W	0.45 W	0
Atom C3436L	1.67	13	2	1.67 W	0.50 W	0
Atom C3436L	1.83	13	2	1.83 W	0.55 W	0
Atom C3436L	2	18	4	2.00 W	0.6 W	0
Xeon D-1531	2.2	45	6	2.2 W	0.66 W	0
Xeon D-1649N	2.3	45	6	2.3 W	0.69 W	0
Xeon S 4215	2.5	85	8	2.5 W	0.75 W	0
Xeon E-2254ME	2.6	45	4	2.6 W	0.78 W	0
Xeon D-1653N	2.8	165	8	2.8 W	0.84 W	0
Xeon Scalable	3.4	165	10	3.4 W	1.02 W	0.025
Xeon Scalable	3.8	165	10	3.8 W	1.14 W	0.051
Xeon Scalable	3.9	165	12	3.9 W	1.17 W	0.102
Xeon Scalable	4.2	165	16	4.2 W	1.26 W	0.204

to address real-time complexity, demonstrating adaptability and robustness in challenging conditions. This highlights their suitability for real-time IoT applications that usually have simpler structures.

We compared these algorithms under various security requirement probabilities. The workflows' arrival rate follows a Poisson distribution, with an anticipated arrival rate of '1' per minute. To explore a variety of scenarios, specific security probabilities (p_1, p_2, p_3) for mist/edge, fog, and cloud are used. The deadline for each application is denoted by κ, which establishes the severity of an application's deadline relative to its minimum execution time given the resources that are available. This formula is articulated in (46) and is borrowed from [1], [4], [9]. The default value of κ is empirically set to 2 for our experiments.

$$Dl^z = \kappa \times EFT_{t_{\text{end}}^z}^{\hat{r}} \tag{46}$$

The used resources, detailed in Table III, are divided into three tiers: mist/edge, Fog, and Cloud. For the Cloud tier, we utilized Amazon resource types from m1.medium to m1.2xlarge. It is assumed that a resource can be provisioned using virtualization technology from these resource types. Therefore, a resource type

with a TDP of 11 and 4 cores would have a maximum power consumption of 1.5 watts [4].

Based on [4], we consider the following default values: $\hat{\tau}_{I/L}$ for I/M, I/F, I/C, M/M, M/F, M/C, F/F, F/C, C/C are 20Mb, 25Mb, 50Mb, 50Mb, 50Mb, 33Mb, 1 Gb, 100Mb, 10 Gb; $H_{I/L}$ is 0.5 [4]; failure rates λ and β are 10^{-5} and 1, respectively [1], [3]. The default distribution of security probabilities among tasks and resources is established at (0.33, 0.33, 0.33), alongside a CCR set at 1.5.

B. Evaluation Metrics

We utilize various metrics to assess our work. The first metric, described in (47), quantifies average reliability by dividing the aggregate reliability of each workflow by the total number of workflows in a workload.

$$R(W) = \frac{\sum_{z=1}^{|G|} R(G^z)}{|G|} \tag{47}$$

Additionally, we analyze the monetary cost and energy consumption of scheduling algorithms, as specified in (27). To assess the algorithm's ability to meet constraints, we consider the Deadline Success Rate (SR), calculated according to (48), and the Reliability Success Rate (RSR) depicted in (49), which addresses the reliability constraint specified in (30). Finally, ENS evaluates the energy saving of evaluated algorithms relative to our RSECH.

$$\mathrm{SR}(W) = \frac{\sum_{z=1}^{|G|} SR(G^z)}{|G|} \tag{48}$$

$$\mathrm{RSR}(W) = \frac{\sum_{z=1}^{|G|} PSR(G^z)}{|G|} \tag{49}$$

C. Results and Discussion

We conducted a comprehensive evaluation of each algorithm's performance, employing tasks from [25, 50, 75, 100] in size and Communication to Computation ratios (CCR) from [0.5, 1.0, 1.5, 2.0], to form a workflow. All workflows are submitted according to a Poisson distribution to examine the effects of computation and communication-intensive workflows. Each experimental iteration consisted of 200 randomly selected from five distinct scientific workflows, as mentioned earlier, with varying task sizes. The final results were obtained by averaging the outcomes from twenty independent experiments.

1) Workloads With Various CCR and Security Requirement: In this section, we analyze the results of the experiments. Fig. 3 shows the rental costs for varying CCR values. Our proposed algorithms (RCSECH and RSECH) led to more cost-effective solutions by identifying initial free gaps that can fit the task in balanced Energy Cost and reliability. They prioritize mist and fog resources, which incur low or zero rental costs, provided they meet reliability, deadline, and security constraints. Consequently, tasks are primarily assigned to these resources, optimizing rental costs and avoiding reliance on more expensive cloud resources. RSECH outperforms RCSECH in rental cost due to RCSECH's need for duplication to meet reliability requirements. SCEAH has a higher cost but performs better

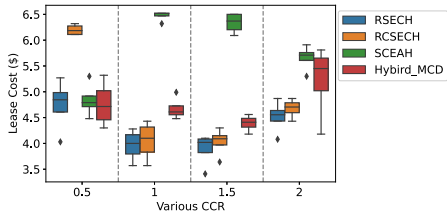

Fig. 3. Monetary Cost of the scheduling in various CCR.

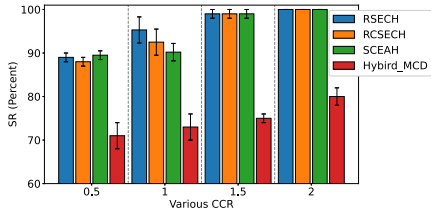

Fig. 4. Success rate of the scheduling in various CCR.

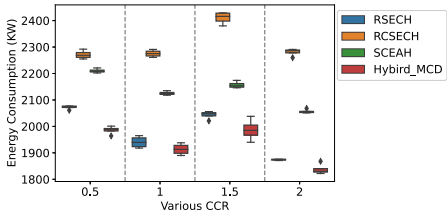

Fig. 5. Energy consumption of the scheduling in various CCR.

in other QoS metrics, such as the SR compared to Hybird-MCD. Both RCSECH and RSECH provide low-cost services, especially for workflows with higher CCR ratios, allowing for optimal task scheduling in terms of rental cost.

Fig. 4 shows the SR of algorithms in various CCRs. RCSECH and RSECH also outperform others when CCR is greater than or equal to 1. For CCR 0.5, SCEAH leads to better SR. Hybrid-MCD performs the least due to its LFT method for deadline distribution. In LFT method, it sets a flexible deadline for early tasks but a strict one for later tasks in the DAG. This allows the algorithm to postpone the execution of early tasks (until their LFT), while promptly executing later tasks to avoid missing their deadline (that is not always possible).

Fig. 5 shows energy consumption in various CCRs. RSECH outperforms SCEAH and RCSECH, with results comparable to Hybrid_MCD. RCSECH incurs slight energy overheads to satisfy its reliability constraint. Despite the overhead, it is justified as it ensures constraint satisfaction, preventing QoS violations. Hybrid_MCD uses the LFT method for deadline distribution, allowing energy-efficient resource use, but this also incurs an SR overhead.

Fig. 6 shows average reliability in various CCRs. Reliability deteriorates as the computation of the workloads increases from the communication of the workloads. RCSECH provides better average reliability. RSECH can outperform others, even without using the task duplication technique.

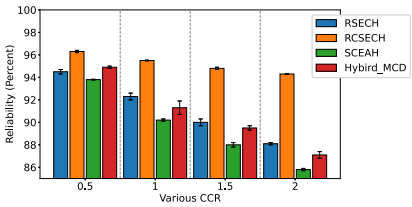

Fig. 6. Average Reliability of the scheduling in various CCR.

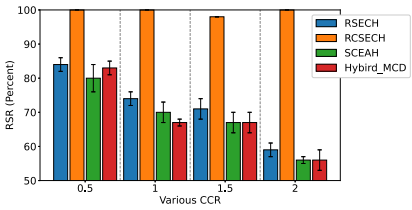

Fig. 7. RSR of the scheduling in various CCR.

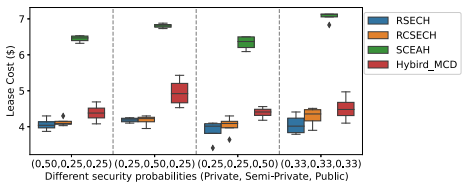

Fig. 8. Monetary Cost of the scheduling in various security probability.

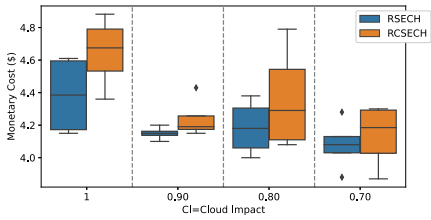

Fig. 9. Monetary Cost of the scheduling in Cloud Impact Value.

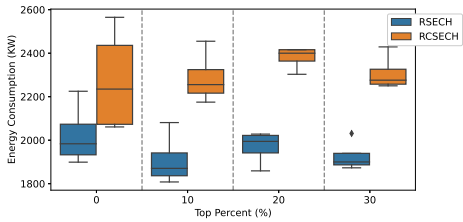

Fig. 10. Energy consumption of the scheduling in various Cloud Impact Value.

TABLE IV
ANALYSIS OF ALGORITHMS AVERAGE RESULT

Metric/Alg	RSECH	RCSECH	SCEAH	Hybird-MCD
RU(M,F,C)(%)	(29,65,6)	(31,63,6)	(21,68,11)	(29,65,7)
SR	93.32%	92.5%	93.11%	74.17%
LC	4.016$	4.21$	6.82$	4.65$
Reliability	90.96%	95.01%	88.5%	90.0%
RSR	76.9%	100%	71%	72.25%
ENS(RSECH)	0%	15.82%	-4.3%	8%

Fig. 7 shows the RSR in various CCRs. A reliability constraint of 0.95% is set for each workflow. RCSECH achieves a 100% RSR across all CCRs due to the task duplication method proposed in this paper. This method ensures that each workflow meets or exceeds the specified reliability constraint. Our approach identifies tasks with execution reliability lower than the reliability guard and uses task duplication to satisfy the overall workflow reliability.

based on the figure, other scheduling algorithms cannot ensure the reliability constraint and end up violating it, resulting in a lower RSR. Specifically, in our experiments, which included various computation- and communication-intensive tasks of different sizes and graph structures, maintaining reliability above the constraint could not be achieved without task duplication.

Fig. 8 shows rental costs for various algorithms under different security requirements, assessed across four security probabilities which shows that RSECH and RCSECH outperform others. Higher public security requirements can reduce rental costs by enabling the scheduler to search a larger domain of solutions, particularly cloud resources, to find an optimal match between tasks and resources. Conversely, stringent security requirements limit the available resource options.

2) Workloads With Various Cloud Impact (CI): We used (44) to assess each tier's influence when selecting resources for tasks, evaluating rental cost and energy consumption under different cloud impacts. Fig. 9 shows the Cloud's impact on rental costs. Equal Cloud, Fog, and Mist/edge impacts increase monetary costs due to the Cloud's superior computational capabilities. Reducing the impact on the cloud involves balancing resource scoring, increasing utilization of Fog and mist/edge, and subsequently lowering overall monetary costs.

Fig. 10 shows energy consumption outcomes for different CI. Lower CI reduces energy consumption due to Fog and Mist/edge resources' lower frequency and TDP. In our study, a CI of 0.90% led to the balanced energy and rental costs, and thus it was used as the default for our experiments.

3) Overall Comparison of the Algorithms: Our algorithm outperforms others in several metrics (Table IV). These results are derived from a comprehensive evaluation involving 200 randomly submitted workflows, each executed with 20 iterations, as previously described

The notion, RU(M,F,C)(%) means the percentage of distribution resources from different tiers (e.g., Mist/Edge, Fog, and Cloud) used to run the workload. RSECH allocates 29% of tasks to the mist/edge tier, 65% to the Fog, and 6% to the Cloud. It leads to higher SR and has the lowest rental cost. RCSECH excels in reliability, satisfying 100% of workload reliability constraints (RSR). Other algorithms show lower results in this metric due to constraint violations. RSECH also leads to better Energy Saving (ENS) compared to others. To comprehensively evaluate the scalability of our proposed framework, we conducted additional experiments using workflows ranging from 100 to 500 tasks and a corresponding increase in resource number as outlined in Table III. However, due to the absence of statistically significant variations in the results across these experiments, we

have chosen to present only the most relevant findings in the following sections for clarity and conciseness.

VI. Conclusions and Future Work

This paper presents a real-time workflow scheduling heuristic for a compute-continuum architecture, emphasizing reliability, cost, and energy efficiency under security, reliability, and deadline constraints. Our algorithms, RCSECH and RSECH, outperform Hybrid-MCD and SCEAH in various conditions, including different CCR values and security probabilities. RSECH reduces rental costs by 15% and 69%, and RCSECH by 10% and 62%, compared to Hybrid-MCD and SCEAH, respectively. RSECH also saves 8% more energy than SCEAH and has an 18% higher SR than Hybrid-MCD. Notably, RCSECH fully meets reliability constraints, unlike the other algorithms. Future work will focus on developing a distributed version to optimize load and address single points of failure, and exploring anonymization techniques like Noise Addition and Tokenization to enhance security and adapt to task-specific requirements.

References

[1] A. Taghinezhad-Niar and J. Taheri, "Reliability, rental-cost and energy-aware multi-workflow scheduling on multi-cloud systems," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 2681–2692, Third Quarter 2023.

[2] J. F. Tsai, C. H. Huang, and M. H. Lin, "An optimal task assignment strategy in cloud-fog computing environment," *Appl. Sci. (Switzerland)*, vol. 11, no. 4, pp. 1–8, 2021.

[3] M. I. Khaleel, "Hybrid cloud-fog computing workflow application placement: Joint consideration of reliability and time credibility," *Multimedia Tools Appl.*, vol. 82, no. 12, pp. 18185–18216, 2023.

[4] G. L. Stavrinides and H. D. Karatza, "Security, cost and energy aware scheduling of real-time IoT workflows in a mist computing environment," *Inf. Syst. Front.*, pp. 1–19, 2022.

[5] M. Iorga, L. Feldman, R. Barton, M. Martin, N. Goren, and C. Mahmoudi, "Fog computing conceptual model," *Special Publication (NIST SP)*, Nat. Inst. Standards Technol., Gaithersburg, MD, 2018. Accessed: Jul. 11, 2024. [Online]. Available: https://doi.org/10.6028/NIST.SP.500-325

[6] A. Taghinezhad-Niar, S. Pashazadeh, and J. Taheri, "QoS-aware online scheduling of multiple workflows under task execution time uncertainty in clouds," *Cluster Comput.*, vol. 25, pp. 3767–3784, 2022.

[7] G. L. Stavrinides and H. D. Karatza, "Workload scheduling in fog and cloud environments: Emerging concepts and research directions," *Lecture Notes Netw. Syst.*, vol. 289, pp. 3–32, 2022.

[8] H. S. Ali and R. Sridevi, "Mobility and security aware real-time task scheduling in fog-cloud computing for IoT devices: A. fuzzy-logic approach," *Comput. J.*, vol. 67, no. 2, pp. 782–805, 2023.

[9] X. Tang, "Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2909–2919, Fourth Quarter 2022.

[10] M. Alam, M. Shahid, and S. Mustajab, "Security prioritized multiple workflow allocation model under precedence constraints in cloud computing environment," *Cluster Comput.*, vol. 27, no. 1, pp. 341–376, 2024.

[11] K. K. Chakravarthi, P. Neelakantan, L. Shyamala, and V. Vaidehi, "Reliable budget aware workflow scheduling strategy on multi-cloud environment," *Cluster Comput.*, vol. 25, pp. 1189–1205, 2022.

[12] F. Li, W. J. Tan, and W. Cai, "A wholistic optimization of containerized workflow scheduling and deployment in the cloud–edge environment," *Simul. Modelling Pract. Theory*, vol. 118, 2022, Art. no. 102521.

[13] A. Taghinezhad-Niar, S. Pashazadeh, and J. Taheri, "Energy-efficient workflow scheduling with budget-deadline constraints for cloud," *Computing*, vol. 104, no. 3, pp. 601–625, Mar. 2022.

[14] I. Attiya, M. A. Elaziz, L. Abualigah, T. N. Nguyen, and A. A. El-Latif, "An improved hybrid swarm intelligence for scheduling IoT application tasks in the cloud," *IEEE Trans. Ind. Informat.*, vol. 18, no. 9, pp. 6264–6272, Sep. 2022.

[15] M. I. Khaleel, "Multi-objective optimization for scientific workflow scheduling based on Performance-to-Power Ratio in fog–cloud environments," *Simul. Modelling Pract. Theory*, vol. 119, 2022, Art. no. 102589.

[16] P. Shukla and S. Pandey, "DE- GWO: A multi-objective workflow scheduling algorithm for heterogeneous fog-cloud environment," *Arabian J. Sci. Eng.*, vol. 49, no. 3, pp. 4419–4444, 2024.

[17] H. Chen, X. Zhu, D. Qiu, L. Liu, and Z. Du, "Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2674–2688, Sep. 2017.

[18] M. Alam, M. Shahid, and S. Mustajab, "Security challenges for workflow allocation model in cloud computing environment: A comprehensive survey, framework, taxonomy, open issues, and future directions," *J. Supercomputing*, vol. 80, no. 8, pp. 1–65, 2024.

[19] S. Javanmardi, M. Shojafar, R. Mohammadi, V. Persico, and A. Pescapè, "S-FoS: A secure workflow scheduling approach for performance optimization in SDN-based IoT-Fog networks," *J. Inf. Secur. Appl.*, vol. 72, 2023, Art. no. 103404.

[20] M. Hussain, L.-F. Wei, A. Rehman, F. Abbas, A. Hussain, and M. Ali, "Deadline-constrained energy-aware workflow scheduling in geographically distributed cloud data centers," *Future Gener. Comput. Syst.*, vol. 132, pp. 211–222, 2022.

[21] H. Jiang, X. Dai, Z. Xiao, and A. K. Iyengar, "Joint task offloading and resource allocation for energy-constrained mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4000–4015, Jul. 2022.

[22] G. L. Stavrinides and H. D. Karatza, "Security and cost aware scheduling of real-time IoT workflows in a mist computing environment," in *Proc. 8th Int. Conf. Future Internet Things Cloud*, 2021, pp. 34–41.

[23] N. Garg, D. Singh, and M. S. Goraya, "Energy and resource efficient workflow scheduling in a virtualized cloud environment," *Cluster Comput.*, vol. 4, pp. 1–31, 2020.

[24] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Gener. Comput. Syst.*, vol. 29, pp. 682–692, 2013.

Ahmad Taghinezhad-Niar received the MSc and PhD degrees in computer engineering from the University of Tabriz, in 2017 and 2021, respectively. He is an assistant professor with the Department of Computer Engineering, University of Tabriz, Iran. His research interests lie in distributed systems, cloud computing, scheduling algorithms, and formal methods. Additionally, he actively contributes to the academic community by serving as a reviewer for esteemed journals.

Javid Taheri (Senior Member, IEEE) received the bachelor's and master's degrees in electrical engineering from the Sharif University of Technology, Tehran (Iran), in 1998 and 2000, respectively, and the PhD degree in mobile computing from the University of Sydney (Australia), in 2007. He is a full professor with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, U.K. and with the Department of Computer Science, Karlstad University, Sweden. He is also a visiting professor with Ericsson AB, Stockholm, Sweden. He is the recipient of many awards, including being selected as one of the top 200 young researchers in the world by the Heidelberg Forum in 2013 and the recipient of the prestigious IEEE Middle Career Researcher award from TSCS in Scalable Computing in 2019. He co-authored more than 250 scientific articles and papers, has served as an editor for more than 25 journals and is a member of the organizing team for more than 50 international conferences.