IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 7, 2021, accepted September 23, 2021, date of publication September 29, 2021,
date of current version October 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3116716

Hybrid Workflow Scheduling on Edge Cloud Computing Systems

RAED ALSURDEH, RODRIGO N. CALHEIROS, (Senior Member, IEEE), KENAN M. MATAWIE, AND BAHMAN JAVADI, (Senior Member, IEEE)
School of Computer, Data and Mathematical Sciences, Western Sydney University, Sydney, NSW 2000, Australia
Corresponding author: Raed Alsurdeh (r.alsurdeh@westernsydney.edu.au)

ABSTRACT Internet of Things applications can be represented as workflows in which stream and batch processing are combined to accomplish data analytics objectives in many application domains such as smart home, health care, bioinformatics, astronomy, and education. The main challenge of this combination is the differentiation of service quality constraints between batch and stream computations. Stream processing is highly latency-sensitive while batch processing is more likely resource-intensive. In this work, we propose an end-to-end hybrid workflow scheduling on an edge cloud system as a two-stage framework. In the first stage, we propose a resource estimation algorithm based on a linear optimization approach, gradient descent search (GDS), and in the second stage, we propose a cluster-based provisioning and scheduling technique for hybrid workflows on heterogeneous edge cloud resources. We provide a multi-objective optimization model for execution time and monetary cost under constraints of deadline and throughput. Results demonstrate the framework performance in controlling the execution of hybrid workflows by efficiently tuning several parameters including stream arrival rate, processing throughput, and workflow complexity. In comparison to a meta-heuristics technique using Particle Swarm Optimization (PSO), the proposed scheduler provides significant improvement for large-scale hybrid workflows in terms of execution time and cost with an average of 8% and 35%, respectively.

INDEX TERMS Hybrid workflow scheduling, streaming scheduling, gradient search optimization, resource estimation and provisioning.

I. INTRODUCTION

The Internet of Things (IoT) is a technology that refers to a large set of objects (machines, devices, etc.) that can connect and share data without requiring human intervention. The adoption of this technology steers innovation in the form of smart and intelligent applications to simplify human life and enables new business-oriented, user-specific, and human-centric applications. The next generation of IoT platforms will be applied in domains like 1) industrial for business continuity management and collaborative supply chain management [1], 2) public services in smart cities and social sensing [2] and 3) customer services for autonomous cars and smart homes [3]. In IoT, the large network of connected physical devices of sensors and actuators can produce a massive amount of data that need to be handled in real-time or near real-time. Moreover, IoT devices generate data with different frequencies and different timeliness requirements for data delivery. Thus, the IoT-generated data possesses properties that conform to the big data paradigm [4]. However, IoT big data is difficult to process in traditional computing systems. Data stream is not available at once, the stream rate is often of high speed and may vary with time, and timely analysis of the data stream is critical [5].

Data analytic applications are commonly managed and executed with workflow technology. Generally, a workflow is a systematic representation of a process as a set of dependent tasks accordingly to a set of rules [6]. The workflow model aims to automate and minimize the complexity of managing various types of processes, such as human activities, business processes, and scientific experiments [7]. In data analytic applications, it is common to integrate batch and stream processing models [8]. Batch processing involves storing the upcoming data before processing while streaming

The associate editor coordinating the review of this manuscript and approving it for publication was Chih-Min Yu.

processing is related to performing operations on data streams in a real-time or near-time manner. We refer to this integration as a hybrid workflow [9]. Hybrid workflows can be applied in many application domains like traffic monitoring [10], social sensing [11], and business analytic [12].

Stream and batch processing have different Quality of Service (QoS) requirements. Stream processing is latency-sensitive and subjects to constraints like stream rate and throughput [13]. Data stream arrival rate may change over time making it hard to estimate data stream collection and processing intervals. Thus, it's not trivial to determine the size of these intervals in advance. If the interval is too large, the accuracy of prediction can deteriorate when the nature of the data changes, and if the interval is too small, accuracy can deteriorate when that is rather stationary. On the other hand, batch processing has a less time sensitivity processing and more computation-intensive to conduct large data computation for data aggregation and predictive modelling functions. Overall, both processing models aim for maximizing throughput, which determines the efficiency of processing computation under certain user and application QoS requirements.

Cloud computing offers a robust and scalable computation model. However, the bottleneck of pushing continuous data stream from a large number of IoT devices cannot be neglected for latency and cost constraints [14]. Transmitting large chunks of data requires a high network bandwidth to reduce data migration latency. Moreover, much of the raw data (from IoT devices) is unnecessary or unusable and cannot be directly injected into computation cycles. As a result, reducing transmitted data to the cloud and performing more analytic closer to IoT devices is convenient. This refers to edge computing which is a general term of enabling technologies allowing computation to be performed at the edge of the network [15]. However, this may increase the complexity of managing and monitoring the execution of IoT-based workloads (such as hybrid workflows) on such a heterogeneous computing system, i.e., edge cloud. Hybrid workflow scheduling implies an understanding of the differences in processing behaviour of stream and batch applications to propose workflow schedulers that support maintainable integration between these applications with consideration of user QoS constraints.

Few research works have studied the hybrid stream-batch workflows and proposed general-purpose execution models [16]–[19] to maintain QoS constraints like latency, throughput, fault-tolerant, etc. However, the hybrid modelling is not well-formulated to reflect the data and computation dependency between stream and batch tasks. The motivation of hybrid model is to provide an efficient management and control over applications which are complex in scale and internal dependencies, involve short-term stream intervals and online batch feeding, and flexible to their parameters tuning. In this paper, we are extending our previous work [9] to propose a hybrid workflow scheduling framework on edge cloud computing that considers the integration requirements for hybrid workflows while optimizing the workflow execution time and monetary cost. The framework involves algorithms for resource estimation, provisioning, and task scheduling. This paper has the following contributions:

- Hybrid workflow resource estimation algorithm based on the gradient descent search (GDS) technique.
- Cluster-based resource provisioning and denting adjustment scheduling algorithm for hybrid workflow on edge cloud computing environment.
- Comprehensive performance analysis of estimation and scheduling algorithms, including scheduling adaptability, edge capability, and optimization time

Next, we discuss some of the state of the art algorithms and techniques for resource estimation, provisioning, and task scheduling in the cloud and edge cloud systems are presented in Section II, and then the problem formulation is described in Section III, including the edge cloud system model and a detailed description of the application model. The hybrid workflow scheduling in the edge cloud system is described in Section IV and the system performance is evaluated based on experimental findings from various perspectives in Section V. Finally, conclusions are provided in Section VI.

II. RELATED WORK

The main objective of workflow scheduling is to generate scheduling plan(s) that maximize efficiency under certain optimization objective(s) such as makespan, execution time, monetary cost, reliability and resource utilization. Directed acyclic graph (DAG) is commonly used to represent a scheduling plan; vertices denote workflow tasks, and the edges show the dependencies among them [20]. Workflow and DAG are often used interchangeably in the literature. Generally, a workflow is a systematic representation of a process as a set of dependent tasks accordingly to a set of rules [6]. The workflow model aims to automate and minimize the complexity of managing various types of processes, such as human activities, business processes and scientific experiments [7].

However, few researchers have proposed models which are efficient and relevant to hybrid workflows. This section provides a broad analysis of the research body on workflow scheduling, including resource provisioning and task allocation in different computing systems.

A. RESOURCE PROVISIONING AND TASK SCHEDULING IN CLOUD COMPUTING

In cloud computing, resource provisioning is an adaptive process of provisioning and deprovisioning resources according to workload changes and service demand at a certain point in time [21]. Resource provisioning is critical to control resource utilization and cost by avoiding resource over-provisioning. Provisioning is estimated by reactive or proactive mechanisms. Reactive mechanisms continuously track service workloads and fire scaling triggers in response to resource demands. However, the time needed to update and

apply a new provisioning plan can be detrimental, particularly for latency-sensitive applications that involve real-time or near real-time processing. Proactive mechanisms are efficient in incorporating timely-constrained provisioning by observing workloads and predicting resource demands by applying statistical or mathematical models, such as queuing theory, reinforcement learning and control theory [22].

Mao and Humphrey [23] proposed two algorithms to resolve the auto-scaling issue: scheduling-first, and scaling-first. The first algorithm applies total budget distribution to each workflow task, generates the fastest execution plan, and finally acquires cloud resources. The second algorithm estimates the required resources concerning data size and resource capabilities, and finally schedules the workflow tasks. Nikravesh *et al.* [24] proposed a predictive auto-scaling system to scale cloud resources automatically for different types of workloads and fixed observation window size. The work does not illustrate how workload features are injected into the prediction model. Wagner and Michalewicz [25] worked on time series data prediction based on an adaptive sliding window. The technique is not suitable for time-sensitive applications due to the complexity of real-time windows size identification. Deypir *et al.* [26] used an adjustment technique for stream arrival rate with variable window size. Experimental evaluations showed the proposed technique efficiency in adapting window size to improve system performance. However, the technique involves user intervention to set the window size reduction threshold. Warneke and Kao [27] considered the dependencies between application tasks, and determined the specifications of cloud resources needed to provide efficient resource allocation and scheduling framework. However, application modelling does not study how changes in workflow structure or input data are reflected in generated schedules.

For task scheduling in cloud systems, the literature has demonstrated an extensive research body which covers a wide range of task scheduling aspects for optimizing objectives like cost, execution time, energy, reliability, security, energy, etc. [28]–[30]. Topcuoglu *et al.* [31] proposed one of the best heuristics scheduling algorithms, the heterogeneous earliest finish-time (HEFT) algorithm to minimizes the overall workflow makespan through minimizing the earliest finish time for critical tasks. HEFT performs task allocation in two steps: task prioritizing and instance selection. Task prioritizing ranks tasks in a list according to the cumulative execution time on each VM instance and average communication time between VMs of dependent tasks. The unallocated task with the highest rank is selected and mapped to its best instance, which guarantees the lowest finish time. Durillo and Prodan extended the HEFT algorithm [31] by proposing the Multi-objective-HEFT (MOHEFT) algorithm [32]. MOHEFT is a generic multi-objective scheduling algorithm, which works by generating several scheduling solutions. The quality of a solution is assessed using the metric of crowding distance. However, complete coverage traversing is adopted in MOHEFT to generate new solutions for assigning tasks to instances, which consumes a large amount of time. Thus, the technique is not efficient for large-scale workflows [33].

Abrishami *et al.* [34] adapted the partial critical path (PCP) algorithm for deadline-constrain. The algorithm works by generating PCPs, and for each PCP, included tasks are allocated to the same VM instance based on heuristics. Grouping critical tasks can enhance resource utilization and reduce the total execution time while meeting the deadline. The technique can reduce computation time and cost, but the communication time and cost are not considered.

B. THE CONTRIBUTION OF EDGE COMPUTING

In cloud computing, data transfer times can maximize the overall workflows execution time and cost. IC-PCP overcomes the issue by grouping and scheduling dependent tasks on the same VM to reduce the amount of transferred data between VMs [34]. However, IC-PCP does not provide an accurate estimate of execution and transmission time. Sahni and Vidyarthi [35] applied the grouping strategy by constructing pipelines of interdependent tasks aiming to reduce the workflow execution cost while meeting the deadline. However, no priority policy is applied to select the next allocated pipeline [44]. Lin and Wu [45] proposed a technique to minimize the overall workflow execution delay considering the budget constraint. The scheduling problem assumes a one-to-one VM allocation to each workflow task. However, latency is a bottleneck for running stream applications on cloud systems. Issues like streaming from devices over long distances with cloud resources and the unpredictable quality of transmission networks need to be considered in stream workflow scheduling. However, in reality, migrating streams from the smart objects and transferring the data to the centralized cloud environment is considerably not efficient due to the increase in the latency time and response time of the real-time data items.

Computing models like edge computing and edge cloud computing are proposed to resolve such issues [46]–[50]. Edge computing is intended to overcome the challenges (limited bandwidth and network latency) of migrating large amounts of data for real-time data processing. Recently, many models have been proposed to conquer the challenges of stream workflows by adopting resource provisioning and task scheduling techniques for use in an edge cloud computing environment, to improve edge resource utilization, reduce latency by processing data-intensive tasks in nearby edges, maximize communication stability for high-performance stream processing, and provide an efficient task placement strategy to achieve reliable resource provisioning.

Madej *et al.* [37] proposed a fairness-based scheduler for edge cloud computing. The paper compares four scheduling techniques, namely, (First-in First-out) FIFO, a client fair, priority fair, and a hybrid that accounts for the fairness of both clients and job priorities. The experimental results demonstrate that the hybrid technique is the best and that the fair scheduler is feasible to implement in edge cloud systems. Naha *et al.* [38] proposed deadline-constrained resource

TABLE 1. Summary of the related work.

Author(s)	Application Type	Technique	Objective(s)	Infrastructure
Topcuoglu et al. [31]	General Workflow	Heuristics	Execution time	Cloud
Abrishami et al. [34]	General Workflow	Heuristics	Execution time and cost	Cloud
Sahni et al. [35]	General Workflow	Heuristics	Execution time and cost	Cloud
Zhang et la. [36]	IoT batch jobs	Online heuristics	Execution time	Cloud
Madej et al. [37]	General Workflow	Heuristics	Scheduling fairness	Edge Cloud
Naha et al. [38]	Delay-sensitive	Heuristics	Processing time and cost	Fog Cloud
Zhou et al. [39]	IoT data streaming	Online gradient descent (OGD)	Energy Bandwidth	Collaborative Edge
Huang et al. [40]	IoT batch jobs	Transformed linear programming (LP)	Revenue	Mobile Edge Cloud
Farhadi et al. [41]	Data-intensive	Mixed integer linear program (MILP)	Service placement Throughput	Edge Cloud
Zhang et al. [42]	Computation-intensive	Deep reinforcement learning (DRL)	Energy consumption Completion time Computation cost	Edge Cloud
Wu et al. [43]	IoT stream applications	Deep reinforcement learning (DRL)	Execution time	Edge
The proposed framework	Hybrid Workflow	Gradient descent search (GDS)	Execution time and cost Throughput	Edge Cloud

allocation and provisioning in the fog cloud environment. The algorithm addresses the issue of user QoS variation and resource limitations on fog computing. Resources are allocated based on a scoring system and considering various parameters. Zhou *et al.* [39] proposed an online gradient descent technique to estimate the rate of data streams to reduce the cost of cross-edge IoT data streaming processing. The work does not consider tuning other streaming processing parameters like aggregation processing interval and throughput. Farhadi *et al.* [41] proposed a data-intensive application scheduler using a mixed-integer linear program (MILP) on an edge cloud system. The technique aims to maximize the system utilization in terms of the number of placed scheduled services. Zhang *et al.* [42] adopted a deep reinforcement learning (DRL) framework for joint workflow scheduling optimization on a hybrid edge cloud system. Wu *et al.* [43] also applied a DRL framework to schedule IoT stream applications on an edge system while reducing the execution latency.

C. HYBRID WORKFLOW SCHEDULING

Hybrid workflow, or batch stream combination, has been sufficiently studied and many research frameworks have been proposed to address some of the requirements of such an integration. Kailasam *et al.* [16] proposed BStream which combines batch and stream processing at different cloud layers and uses a predictive model to estimate resource allocation and task distribution to meet task deadlines. Carbone *et al.* [17] developed Flink, so-called Apache Flink, a unified stream and batch computing framework which expresses various classes of data processing applications as fault-tolerant dataflows in a low-latency and high throughput. Apache Flink is designed for ''stateful streaming processing'' which handles batch tasks as a bounded stream resolving the high latency of micro-batch architecture in some streaming processing frameworks like Spark [18].

Zhang *et al.* [36] proposed an online heuristic algorithm called DyBBS for adaptive batch size and execution parallelism to reduce end-to-end processing latency and reflects the workload changes. Pishgoo *et al.* [19] proposed batch-stream processing architecture for distributed modeling of streaming events to detect anomalies in such events. They argue that there is a limitation on considering the hybrid architecture regarding how different processing units should interact with each other specifically at the algorithm level. Huang *et al.* [40] proposed a linear optimization model to schedule IoT batch tasks in a mobile edge computing (MEC) system. Meanwhile many industrial frameworks demonstrate adequate support for hybrid workflows, but the influence of workflow structure, i.e., the dependency between stream and batch tasks (in terms of data and computation), is not well formulated to reflect workflow execution optimization for cost, throughput, deadline, etc. Thus, the need arises for developing schedulers that can adopt the hybrid workflow execution behaviour.

Table 1 summarizes the most related works in terms of application type, optimization technique, objectives and computing system. The current literature does not effectively illustrate the opportunities of edge cloud systems in resolving scheduling issues of hybrid workflows like load balancing, edge capacity limitations and data communication between edge and cloud resources. In this work, we provide an end-to-end hybrid workflow scheduling framework in an edge cloud computing environment. The framework handles the requirements of integrating stream and batch processing in IoT applications. The framework is relied on proposed algorithms and techniques for resource estimation using gradient descent search (GDS) technique and resource provisioning and task scheduling using a cluster-based approach.

III. THE SYSTEM MODEL

Cloud computing is not the ideal computing system for latency-sensitive applications, such as real-time gaming, augmented reality and real-time streaming [51]. Because cloud resources are located closer to the core network,

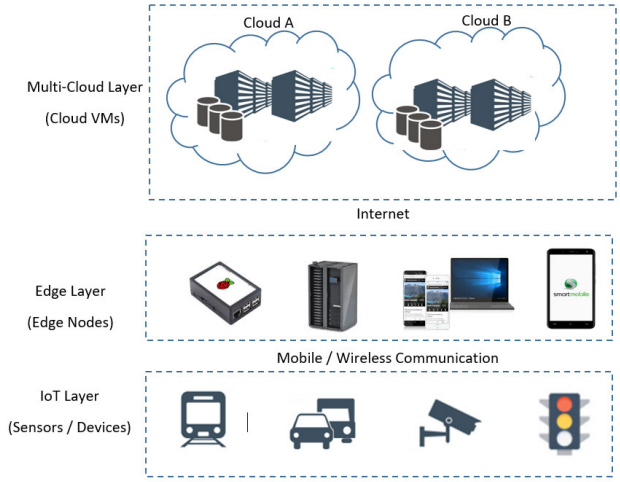

FIGURE 1. Edge cloud computing system model.

the round-trip latency of these applications will be high due to passing data through multiple gateways. Edge computing refers to technologies that permit the computation at the network edges, and act as an intermediate layer to transmit streams to cloud services on behalf of IoT devices [15]. However, a high percentage of this data is temporary and only a small amount might contain meaningful information. Edge computing has a significant role in processing massive data and enables better filtering of what data is uploaded to the cloud. For instance, an autonomous vehicle requires a huge amount of data, in the forms of images and video, to be processed in real-time to make safe driving decisions. However, the availability and low efficiency of edge servers are decisive factors in quality achievement [39]. Hybrid workflow specifications require the development of efficient resource provisioning and scheduling techniques that coordinate the execution of hybrid workflows on edge cloud systems.

A. EDGE CLOUD COMPUTING SYSTEM

The edge cloud system is convenient for hybrid workflows because the stream tasks with latency sensitivity can benefit from the availability of edge resources, whereas batch tasks with heavy workloads can be processed at powerful computation nodes in the public cloud. Figure 1 shows the adopted edge cloud system model which is composed of IoT, edge and multi-cloud resource layers. The three layers are explained as follow:

- *Layer 1 (IoT).* The layer represents the user interaction layer, in which IoT devices (sensors, smartphones, relays, etc.) generate data which can be on various formats and bounded to a particular point of time. IoT devices can send workloads to the nearest edge nodes or cloud resources.
- *Layer 2 (edge layer):* This layer represents all devices on the path between the IoT layer and cloud layer. An edge node can be a non-stationary computation device, such as mobile devices or a Raspberry Pi, or a stationary device, such as a personal desktop, company server or a cloudlet.
  We assumed that edge devices are limited in computation and storage capabilities, but able to handle some pre-processing tasks on the stream processing pipeline. Overall, the edge layer offers computation close to data sources (IoT devices and sensors) to reduce data transfer time, and performs pre-processing in a timely manner constraint [15].
- *Layer 3 (multi-cloud services):* The third layer includes cloud resources that have high computational and storage capabilities, and can handle heavyweight computations such as predictive analysis, machine learning, business intelligence, big data analytics, and complex visualization. Cloud providers offer a range of computing types including Virtual Machines (VMs), Containers and Serverless computing.

TABLE 2. Mathematical notations.

Notation	Description
λ_i	Stream arrival rate (msg/s)
ω_i	Stream processing aggregation window size
μ_i	Task service rate (msg/s)
ρ	System utilization ($0 < \rho < 1$)
W_q	Task waiting time in the queue
σ_s^2	Coefficient square of service time variance
S^2	The variance of service time
τ_i	Minimum task processing throughput
α_i	Task data production factor
d_i	Total data generated by a task
L_i	Total data received by a task
ϑ_i	Task deadline (s)
γ_k	Gradient descent learning rate at step k
ET_i	Task processing time
EST_i	Task earliest start time (s)
LFT_i	Task late finish time (s)

B. APPLICATION MODEL

A hybrid workflow $w = G(T, E)$ is modelled as a Direct Acyclic Graph (DAG) of tasks $V = \{t_1, t_2, \ldots, t_n\}$ with direct edges E and no cycles or conditional dependencies. A task t_i represents a stream ($t_i{}^S$) or batch ($t_i{}^B$) task application. The two types of application are fundamentally different in their modelling features. Table 2 presents mathematical notations used in the hybrid workflow modeling.

A stream task $t_i{}^S$ represents the execution of a stream processing pipeline and it is modeled as an $M/G/c$ queuing system [52]. The queuing modeling provides an estimation of the amount of resources to run a stream processing pipeline under constraints like system utilization, waiting time, and system throughput. Stream processing system must consider these constraints with the variation in workload to achieve a highly optimized, minimal overhead execution engine to deliver real-time response for high-volume applications [53]. The number of servers c in a queuing system refers to the level of parallelism where processing cycles are replicated in

case of a high stream arrival rate. A stream task t_i^S is modeled as:

$$t_i^S = \{\lambda_i, \mu_i, \omega_i, \tau_i, \alpha_i, d_i, c_i\} \quad (1)$$

Based on the approximation by Kleinrock [52], the number of queuing system servers c_i is estimated as follow:

$$c_i = \frac{\lambda_i}{\mu_i \rho} \quad \rho < 1 \quad (2)$$

To reduce the complexity of $M/G/c$ queues, we assume that at a given moment, all servers c will be busy. Hokstad [54] showed that we an $M/G/c$ queue which has an identical service time ($S = 1/\mu$) can be represented as a $M/G/1$ queue with service time ($S = 1/\mu c$). Based on this assumption, the approximation provided by Kleinrock [52] is adopted to estimate the waiting time on queue W_q:

$$W_q = \frac{\rho(1+S^2)}{2(1-\rho)\mu} \quad (3)$$

A batch task t_i^B has the following features:

$$t_i^B = \{\vartheta_i, \tau_i, \alpha_i, \mu_i, d_i, L_i, c_i\} \quad (4)$$

A batch task t_i^B receives data collectively from stream and batch tasks. The total amount of data L_i received by a batch task is computed based on the following formula:

$$L_i = \frac{\tau_i \sum_{j=1}^{n} d_j}{\mu_i} \quad (5)$$

where d_j is the amount of data generated from task t_j, which can be a stream or a batch task. The number of servers c_i (parallelization factor) to run a single iteration of a batch application is estimated as follow:

$$c_i = \frac{L_i}{\vartheta_i \alpha_i} \quad (6)$$

IV. HYBRID WORKFLOW SCHEDULING IN EDGE CLOUD RESOURCES

Workflow scheduling is utilized for mapping tasks to computation resources and planning their execution while fulfilling dependency between tasks and achieving certain QoS parameters and resource preservation goals. Without loss of generality, Figure 2 shows hybrid workflow scheduling framework which consists of three steps, including resource estimation, resource provisioning, and workflow scheduling. In the resource estimation stage, the amount of resources and time to run the workflow are estimated under task execution constraints, including deadline, throughput, and waiting time. Accordingly, workflow tasks are formulated in execution groups. Next, the required computation nodes at the edge cloud system are provisioned to host the execution of these groups. Two type of resources are involved, namely, public clouds (D_1 and D_2) and edge nodes (D_3). Lastly, the workflow tasks are mapped to provisioned resources considering a multi-objective scheduling optimization of execution time and monetary cost.

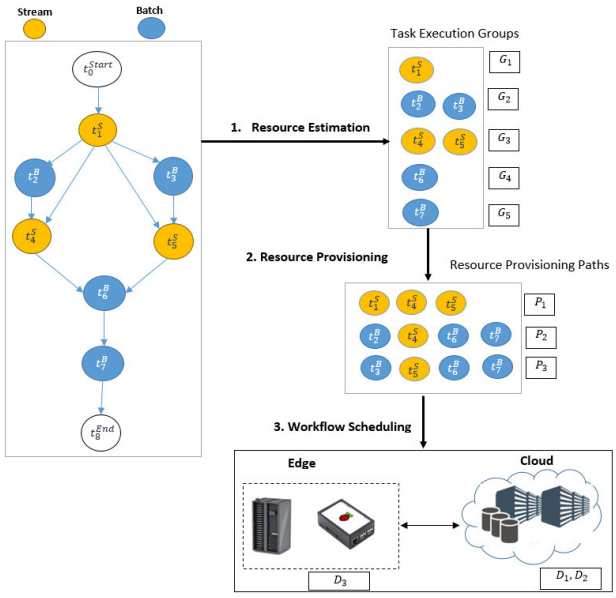

FIGURE 2. A High-level abstraction of hybrid workflow estimation and scheduling.

A. RESOURCE ESTIMATION WITH GRADIENT DESCENT SEARCH APPROACH

The optimization objective of our framework aims to minimize the total execution time E and the number of computation units (cores) R while meeting the execution constraints of stream and batch tasks. The estimation process works by grouping workflow tasks to construct an execution plan considering their execution dependencies and maximizing resource utilization at the group level. The group-based technique can simplify resource provisioning and task scheduling in an edge cloud environment, particularly for complex hybrid workflows. The optimization function for resource estimation is formulated as:

$$\min(E * R) \quad (7)$$

1) RESOURCE ESTIMATION PROBLEM FORMULATION

Gradient descent (GD) is a first-order iterative optimization technique of finding a local minimum for linear and non-linear functions [55]. Gradient descent comes in three variants: batch or vanilla gradient descent, stochastic gradient descent (SGD) and mini-batch gradient descent. Based on the minimum of data required to perform the estimation process and the low complexity of the estimation model with pre-defined workflow execution functions, the batch gradient descent variant is used [56]. The gradient descent search (GDS) process starts at some arbitrary point $x^{(0)}$ and then iteratively moving at direction $\Delta x^{(k)}$ after every step $k \geq 0$ by step size γ_k to the next point $x^{(k+1)}$ as:

$$x^{(k+1)} = x^{(k)} - \gamma_k \Delta f(x^{(k)}) \quad (8)$$

Since the objective is minimizing the function, the step size γ_i value should be selected in a way that will minimize the

function value of the new point such that the step size γ_k^* at step k satisfies:

$$\gamma_k^* = \underset{k\geq 0}{\operatorname{argmin}} f(x^{(k)}) - \gamma \Delta f(x^{(k)}) \tag{9}$$

The application of the gradient descent search (GDS) on the resource estimation problem is described as follow:

- A stationary search point $x^{(k)}$ is a $3 \times d$ matrix, where 3 refers to the number of problem parameters (λ_i, ω_i, τ_i) of a stream task ${t_i}^S$ and d refers to the number of stream tasks.

$$x^{(k)} = \begin{bmatrix} \lambda_0 & \omega_0 & \tau_0 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \lambda_{d-1} & \omega_{d-1} & \tau_{d-1} \end{bmatrix} \tag{10}$$

- The workflow execution function f at point (workflow setup) $x^{(k)}$ is written as:

$$f(x^{(k)}) = E(x^{(k)}) * R(x^{(k)}) \tag{11}$$

  where $E(x^{(k)})$ and $R(x^{(k)})$ are the workflow execution time and amount of resources, respectively. This reflects the resource estimation optimization objective provided in Equation 7.
- The estimation process is a constrained GDS problem as it relies on constraints such as minimum throughput τ_i and maximum deadline ϑ_i.
- The optimization problem objective is to find the matrix $x^{(*)}$ in which the function has a local minimum and mostly a global minimum (as dealing with convex workflow execution functions).

2) RESOURCE ESTIMATION ALGORITHM

The estimation algorithm includes a set of steps to find the optimized estimation cost (number of cores R) with minimum execution time E. Next, The Constrained-based Gradient Descent Hybrid Workflow (C-GDHW) resource estimation algorithm is described in algorithm 1.

The algorithm receives three inputs including 1) workflow structure w, 2) an initial solution $x^{(0)}$ which represents a $3 \times d$ stream task configuration matrix. As shown in Equation 10, the configuration matrix presents the execution setup for all stream tasks including the arrival rate λ, the window size ω of collecting and processing incoming streams, and the percentage of stream of data to be processed or throughput τ. The window size, so-called batch sizing in Spark Streaming [18], is the time duration to create a batch input and send it to a batch queue. Zhang *et al.* [36] experimented stream parameters tuning and their results illustrated the significance of an adaptive batch streaming processing control algorithm for reducing the end-to-end application processing latency. 3) The maximum deadline relaxation factor ∂.

At each iteration k, the algorithm performs the following steps:

1) Based on the current solution $x^{(k)}$, the workflow execution cost $f(x^{(k)})$ is estimated according to Equation 11. The cost estimation flow is described in lines 4-18. At line 4, the *callculateTaskTimes* is called to calculate the execution times for all workflow tasks including the EST, which is the earliest task starting time and the LFT, which is the maximum execution finish time.
2) Based on task execution times, at line 5, the given solution is validated to satisfy the following constraints for a stream task ${t_i}^S$ (Equations 12 and 13) and for a batch task ${t_i}^B$ (Equation 14):

$$\text{Constraint 1: } \omega_i \geq \omega_i' \geq \frac{\rho * \mu_i * c_i}{\tau_i} \tag{12}$$

$$\text{Constraint 2: } \lambda_{i-1} \geq \lambda_i' \geq \rho * \mu_i * c_i \tag{13}$$

where ω_i' and λ_i' are minimum values of window size and arrival time, respectively.

$$\text{Constraint 3: } \vartheta_i \geq E_i \geq \frac{ET_i}{c_i} \tag{14}$$

where ET_i and E_i are processing time, and task execution time, respectively. A batch task processing time ET_i (as provided in Equation 15) is the total time required to process the total amount of received data L_i based on task service rate μ_i. Accordingly, the task execution time E_i (as provided in Equation 14) is computed, such that, it does not exceed the task deadline ϑ_i and maintain the least amount of resources c_i.

$$ET_i = \mu_i * L_i \tag{15}$$

3) If the solution is valid, the grouping process for workflow task V is started. The algorithm tries to fit the task t to a group in the group list G in respect to a certain level of deadline relaxation, otherwise, a new group is created.
4) Once all tasks are assigned to execution groups, the cumulative execution time E and the total number of cores R are calculated in lines 15-19. Accordingly, the cost function value $f(x^{(k)})$ at the given solution is computed based on Equation 11.
5) At line 21, the algorithm convergence is checked against the threshold ∂. If not converged yet, the gradient value $g^{(k)}$ is computed and the next step (solution) $x^{(k+1)}$ is generated as presented in lines 24 and 25, respectively. Lastly, the best workflow configuration $x^{(best)}$ is returned.
6) The time complexity of the C-GDHW algorithm is $\mathcal{O}(kn^2)$ where k is the cost of computing the function derivative and n is the number of iterations for the function to converge [57].
7) The space complexity of the C-GDHW algorithm is $\mathcal{O}(nd + d)$ where d is the number of tasks and also can be the number of groups in the worst case.

Algorithm 1 Workflow Resource Estimation

```
1:  procedure C-GDHW(w, x^(0), ∂)
2:      x^(best) = x^(0)
3:      for k = 0 to n do
4:          callculateTaskTimes(w, V, x^(k))
5:          if isValidSolution (x^(k)) then
6:              V = w.Tasks
7:              G = {}
8:              for each t ∈ V do
9:                  if t.isInGroup() == false then
10:                     g = allocateTaskGroup(t, G, ∂)
11:                     if g ≠ null then
12:                         addTaskToGroup(t,G)
13:                     else
14:                         createNewGroup(t,G)
15:             E = R = 0
16:             for each g ∈ G do
17:                 E = E + g.leadTaskET()
18:                 R = R + g.totalCores()
19:             R = R + groupStreamTasks()
20:             f(x^(k)) = E * R
21:             if f(x^(k)) - f(x^(k−1)) ≤ ϵ then
22:                 x^(best) = x^(k)
23:                 break
24:         g^(k) = γ_k Δf(x^(k))
25:         x^(k+1) = x^(k) − g^(k)
26:     return x^(best)
```

B. HYBRID WORKFLOW PROVISIONING AND SCHEDULING ON EDGE CLOUD COMPUTING

The next step after estimating the required amount of resources is to provision them on the edge cloud environment and then map the allocated resources to workflow tasks. In this work, we assume prior knowledge about the incoming workload, i.e., the amount of data to be processed and transferred during the execution cycles. Thus, offline scheduling is a preferred option [58]. The workflow scheduling is formulated as a multi-objective optimization problem to jointly reduce the end-to-end workflow execution time T and monetary cost C. Both the workload computation and data transfer costs are included in the optimization process, which approaches to achieve the following objective function:

$$\min(T * C) \tag{16}$$

A Cluster-based Hybrid Workflow Provisioning and Scheduling (C-HWPS) technique is adopted to meet the requirements of hybrid workflow scheduling to provision demanded resources and schedule tasks in an edge cloud computing system. The cluster-based technique works by constructing sequential execution paths where each path represents the execution pathway of dependent tasks which should lead to the last workflow task. As a result, one task can be founded on different execution paths. Figure 2 provides an example of the execution of three paths based on the input workflow. Three execution paths are constructed $\{P_1, P_2, P_3\}$.

Algorithm 2 Cluster-Based Hybrid Workflow Scheduling

```
procedure C-HWPS(G, D)
    P = A = {}
    T = C = 0
    P = findExecutionPaths(G)
    while P ≠ ∅ do
        T_iter = C_iter = T_min = C_min = 0
        p_target = null
        U = getUnScheduledPaths(P)
        for each u ∈ U do
            p = u − A
            T_iter, C_iter, S =computePathInEdgeCloud
            (p, D)
            if T_iter * C_iter < T_min * C_min then
                T_min, C_min, = T_iter, C_iter
                p_target, S_target = p, S
        p = p_target − A
        schedulePathInEdgeCloud(p, S_target)
        A = A + t
        P = P − p_target
        T = T + T_iter
        C = C + C_iter
    return A, T, C
```

The C-HWPS technique is an iterative process of provisioning and scheduling tasks on edge nodes D_1 and two public clouds D_2 and D_3 based on the selection of an execution path at each iteration, which has the minimum of the objective function ($T * C$). The execution time T is the sum of computation time (for unscheduled tasks) and the data transfer time (between scheduled and unscheduled tasks). Similarly, execution cost C (processing and data transfer) on edge cloud resources is computed. The scheduling process will continue by adding paths and optimizing time T and cost C until allocating all tasks.

Algorithm 2 presents the cluster-based hybrid workflow resource provisioning and scheduling which aims to minimize the objective function $T * C$. The algorithm takes the inputs of task groups G as the outcome of the resource estimation phase and the current status of the edge cloud system resources D. The status provides details about resources capacity, availability, connectivity and cost. At line 4, the algorithm calls the *findExecutionPaths* function to construct workflow execution paths P based on tasks dependencies structure.

Next, the algorithm iterates over execution paths P to find the one which has the minimum optimization value of the current iteration $T_{iter} * C_{iter}$ where execution time T and cost C on edge cloud system D. The algorithm starts with the first unallocated path u which has at least one unscheduled task, in other words, $[u - A] \neq \emptyset$ where A is a set of

Algorithm 3 Compute Execution Path in Edge Cloud

1: **procedure** computePathInEdgeCloud(p, D)
2: $T = C = 0$
3: $S = null$
4: $ET_e, EC_e, S_e =$ scheduleInEdge(p, D)
5: $TT_e, TC_e =$ computeTransferTimeAndCost()
6: OptimCost$_e = (ET_e + TT_e) * (EC_e + TC_e)$
7: $ET_c, EC_c, S_c =$ scheduleInCloud(p, D)
8: $TT_c, TC_c =$ computeTransferTimeAndCost()
9: OptimCost$_c = (ET_c + TT_c) * (EC_c + TC_c)$
10: **if** OptimCost$_e <$ OptimCost$_c$ **then**
11: $T = ET_e + TT_e$
12: $C = EC_e + TC_e$
13: $S = S_e$
14: **else**
15: $T = ET_c + TT_c$
16: $C = EC_c + TC_c$
17: $S = S_c$
18: **return** T, C, S

scheduled tasks. Unscheduled task set $[u - A]$ computation is evaluated in the edge cloud system D using the function *computePathInEdgeCloud*, line 11, which calls the routine provided in Algorithm 3. In addition to the optimization values, the function returns the associated resource provisioning and scheduling plan S on the edge cloud system D. The path with minimum time and cost p_{target} will be selected and scheduled based on the resource provisioning and scheduling plan S_{target}. Finally, the algorithm returns scheduled tasks A, and optimization object variables T and C. The time and space complexity of the C-HWPS algorithm is $\mathcal{O}(n)$ where n is the number of execution paths and also can be the number of tasks groups in the worst case.

Algorithm 3 describes the resource provisioning and task scheduling steps for executing a path p in an edge cloud system. This work does not assume collaboration between edge and cloud resources. Thus, the scheduler will effectively try to keep the execution path unity in one computing system and reduces the amount of data exchange at different computing layers. The algorithm provides two main functions. The first is to schedule tasks in either edge or cloud, lines 4 and 7. Based on the available resource configuration, execution time and cost are calculated. Stream tasks have the priority for scheduling at the edge layer. The second function *computeTransferTimeAndCost*, lines 5 and 8, computes the data migration time and cost computing nodes based on the data transmission (bandwidth) quality and cost. Finally, the algorithm returns the resource provisioning and scheduling plan S with the expected execution time T and cost C for the given execution path p. The time and space complexity of the Algorithm 3 is $\mathcal{O}(1)$ as execution time and cost calculations are performed at constant time.

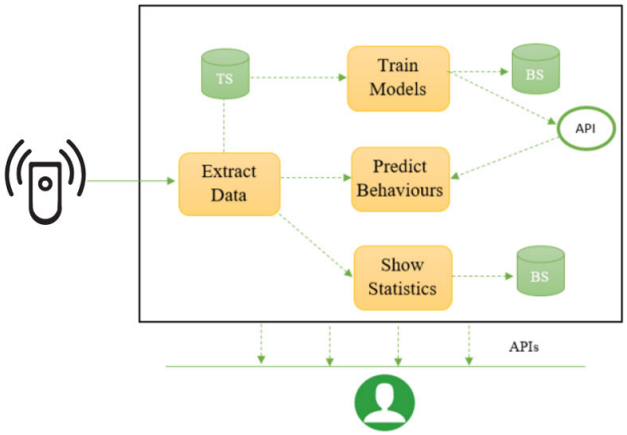

FIGURE 3. High level stream and batch applications dependencies for data analytics workflows [60].

V. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the proposed resource estimation technique, C-GDHW, against two nominated and comparable techniques. The first technique is a meta-heuristics technique using Particle Swarm Optimization (PSO) proposed in our previous work [59] and the second one is the full-mode technique which conducts a semi-optimization process at queuing system utilization level with no reduction mechanism to be applied for stream arrival rate and window size. This technique works as a baseline to assess the performance parameters' tuning of the other two techniques. In addition, this section evaluates the contribution of edge capability in optimizing the execution of hybrid workflows. Finally, the optimization time of the proposed algorithm is investigated in comparison to the PSO-based algorithm.

Next, we provide the experimental setup which includes building various hybrid workflow models from an existing IoT application benchmarking and the resource configuration of the edge cloud system.

A. EXPERIMENTAL SETUP

For the purpose of experimenting with hybrid workflow scheduling, we used IoT dataflow benchmarking results provided by Shukla *et al.* [60] to build three types of workload models that satisfy the design concepts of hybrid workflow with a reasonable level of structural complexity and data dependency. In brief, dataflow tasks were evaluated as sub-workflows and then micro-benchmarking has been conducted by running each sub-task in a single-core machine. Figure 3 shows a typical IoT dataflow with four types of IoT data processing tasks. "Extract Data" block involves stream pre-processing, data normalization and quality check. "Predict Behaviours" block includes applications for determining the future state of an IoT system based on a pre-trained models. "Train Models" involves building and training statistical models based on historical data. In "Show Statistics" block, related tasks like statistical aggregation and analytics

TABLE 3. Hybrid workflows characteristics.

Workflow	#Stream Tasks	#Batch Tasks
Small	12	17
Medium	20	45
Large	36	73

TABLE 4. Resource types for edge and cloud systems.

Provider	Type	#Cores	Price ($/h)
CloudA, CloudB	large	2	[0.093, 0.117]
	xlarge	4	[0.186, 0.232]
	2xlarge	8	[0.371, 0.465]
Edge (Low capability)	small	1	$0.002
	medium	2	$0.070
Edge (High capability)	large	4	$0.090
	xlarge	8	$0.103

TABLE 5. Bandwidth setup at edge cloud system.

Provider	Bandwidth (MB/s)	Bandwidth Cost ($/GB)
Cloud DC	[60.0, 80.0]	[0.14, 0.20]
Edge (Low)	[20.0, 25.0]	[0.03, 0.04]
Edge (High)	[20.0, 40.0]	[0.07, 0.08]

are applied over data streams to understand the behavior of an IoT system. more related to batch processing. Table 3 presents the main characteristics of the three hybrid workflow models: small, medium and large.

We extended CloudSim [61] simulator to reflect the adopted resource model as an edge cloud computing environment. The simulated scenario comprises two cloud data centers (CloudA and CloudB) and edge nodes that reflect the edge cloud resource model. For each data center, various VM configurations, including processing power and cost were applied. CloudA and CloudB VM configurations are based on configurations provided by AWS and Microsoft Azure, respectively. Table 4 provides the resource model setup for the experiments. At the edge layer, we simulated two edge resource setups, high and low capability. The former is used for resource estimation evaluation, while the latter is used for edge capability analysis in the context of hybrid workflow computation. Moreover, we consider the variation in networking quality and cost among the edge cloud system resources. Table 5 shows the bandwidth setup for the three type of resources.

Three experiments were conducted. The first experiment evaluates the performance of the proposed resource estimation technique against PSO-based and Full-mode techniques and with variation in estimation model parameters (arrival rate, window size and throughput). The second experiment measures scheduler adaptability and stability to maintain an optimized workflow execution in response to windows size and arrival reductions. The third experiment sought to identify the contribution of edge resource capability to optimization objectives (execution time T and cost C).

TABLE 6. 95% Confidence interval (CI) - execution time for large workflow and window size scenario.

Model parameter	Technique	$[\bar{X}, S]$ (h)	95% CI (h)
Window size (180%)	C-GDHW	[5.2, 0.53]	[5.01, 5.39]
	PSO	[6.1, 0.68]	[5.86, 6.34]
	Full-Mode	[6.6, 0.01]	[6.60, 6.60]
Arrival Rate (180%)	C-GDHW	[4.05, 0.24]	[3.96, 4.14]
	PSO	[4.38, 0.42]	[4.23, 4.53]
	Full-Mode	[4.65, 0.05]	[4.65, 4.65]
Throughput (90%)	C-GDHW	[4.2, 0.64]	[3.97, 4.43]
	PSO	[4.3, 0.81]	[4.01, 4.59]
	Full-Mode	[4.6, 0.04]	[4.60, 4.60]

B. RESOURCE ESTIMATION EVALUATION

The evaluation of C-GDHW is based on its capability to optimize the execution of hybrid workflows in terms of execution time T and cost C concerning the variation of model parameters (stream window size ω, stream arrival rate λ, and execution throughput τ) and workflow structure complexity (small, medium and large). The window size is the time interval of processing incoming streams (data messages) to generate the desired outcome. This involves applying functions for stream pre-processing, validation, aggregation, etc. The arrival rate is the number of received messages per unit of time. We defined the throughput as the minimum percentage of incoming messages to be processed during the window size. The throughput is a user-defined parameter which reflects the business requirements associated with the workflow execution. Maintaining a highly accurate predictive model is an example of high-throughput demanding applications.

To attain stable estimation results, simulation was carried out 30 times for each model parameter and workflow structure combination, and average values are recorded for each estimation technique. Model parameter values are varied in a range of 20% to 180% for both windows size and arrival rate, and in a range of 10% to 90% for throughput. Relative percentages for the default workflow configuration were proposed due to the variation in parameter values among workflow tasks. At 95% confidence level, results show an acceptable level of variation with reasonable data normality of sample size of 30. Table 6 shows a statistical analysis of large workflow execution time for each technique at the maximum percentage of each model parameter. The full-mode technique has almost no variance in results as it applies a deterministic model with minimal optimization. The PSO-based technique shows the highest variation in reflection to the randomization behaviour of the meta-heuristics technique. Next, each optimization case based on each parameter variation is discussed.

1) WINDOW SIZE

Figures 4, 5 and 6 show the results of varying window size on execution time and monetary cost. The C-GDHW technique shows high stability in controlling the workflow execution

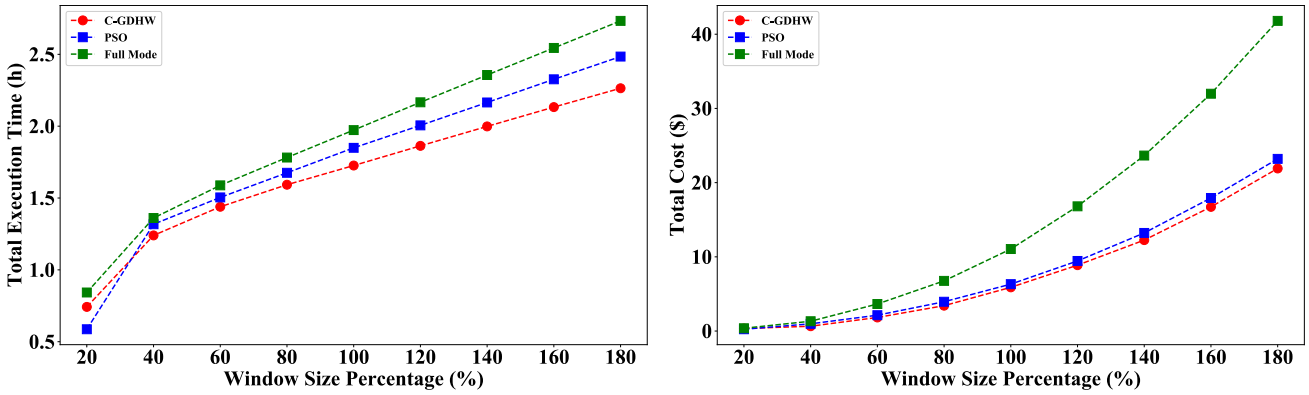

FIGURE 4. Window size variation impact on the small workflow.

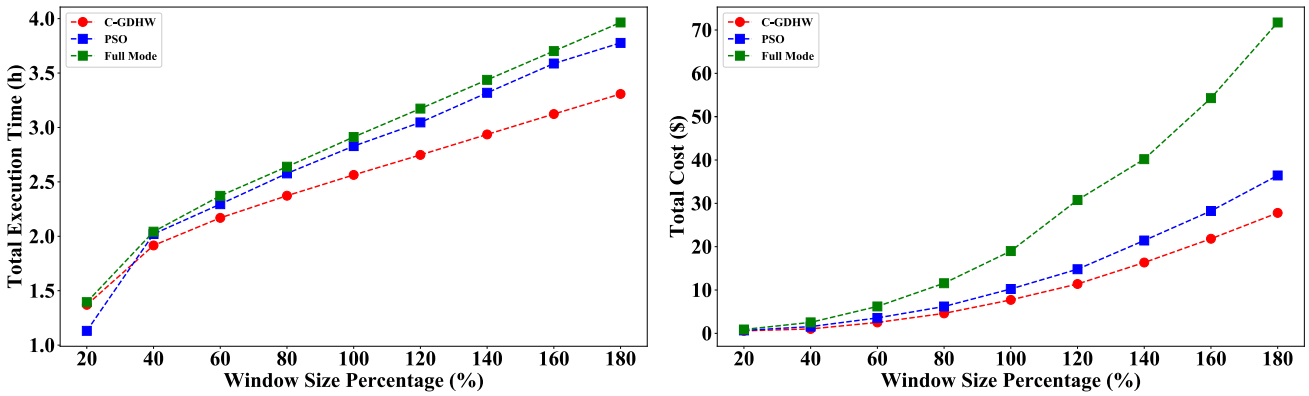

FIGURE 5. Window size variation impact on the medium workflow.

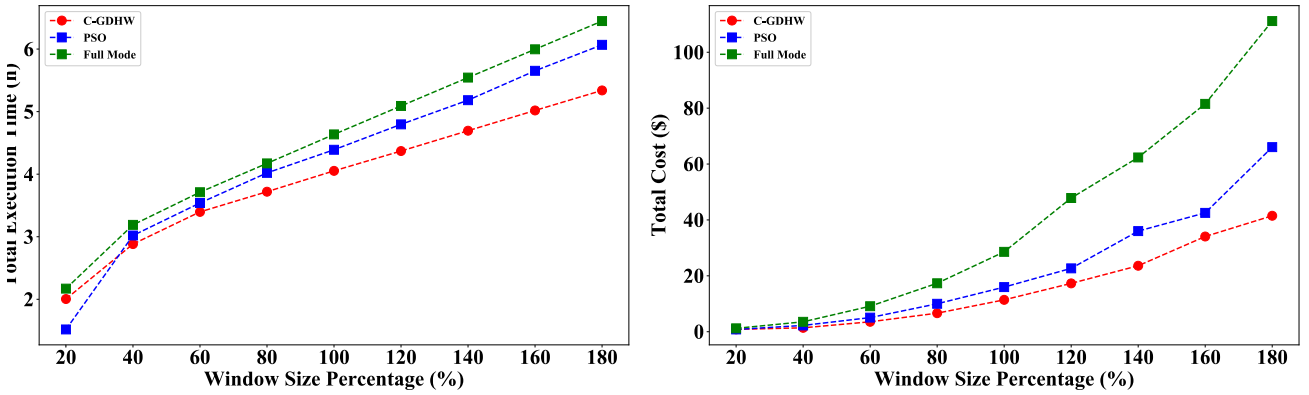

FIGURE 6. Window size variation impact on the large workflow.

time with the increase in window size length. Even though all techniques are linearly responded to the change in window size (after 40%), the C-GDHW accomplished the smallest execution time increase, particularly for medium and large workflows with 45% and 60%, respectively compared to 60% and 75% execution time increase in the case of PSO-based for the same workflow scenarios. In addition, the proposed estimator was able to maintain the cost of adding more resources since more and more data streams are collected during extended data collection intervals. In comparison to the PSO-based technique, the C-GDHW performed 58% less cost for running the large workflow with 180% window size. This potential reduction implies the C-GDHW's ability to group a larger number of tasks and effectively perform periodic resource provisioning. This feature allows IoT applications to produce more data by adding more IoT devices or distributing incoming workload on current computation resources.

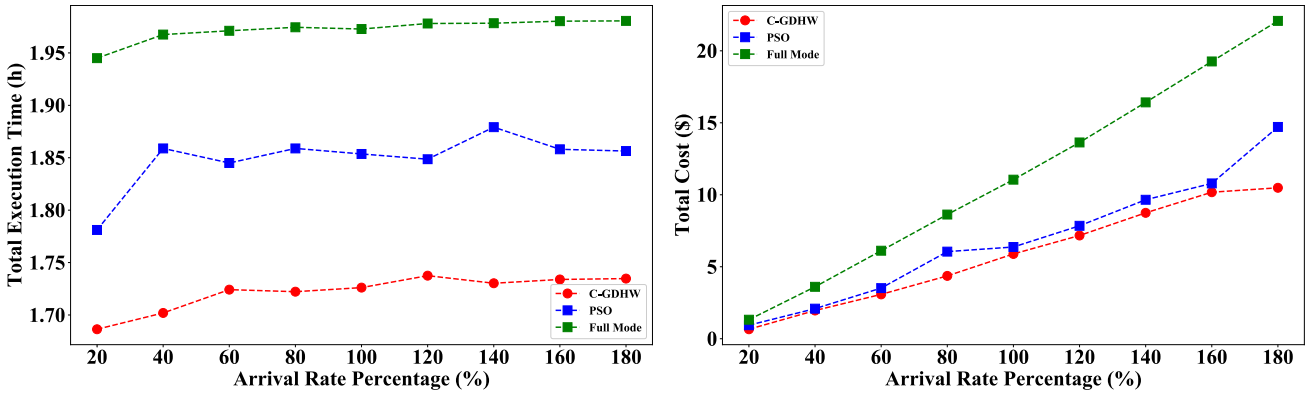

FIGURE 7. Arrival rate variation impact on the small workflow.

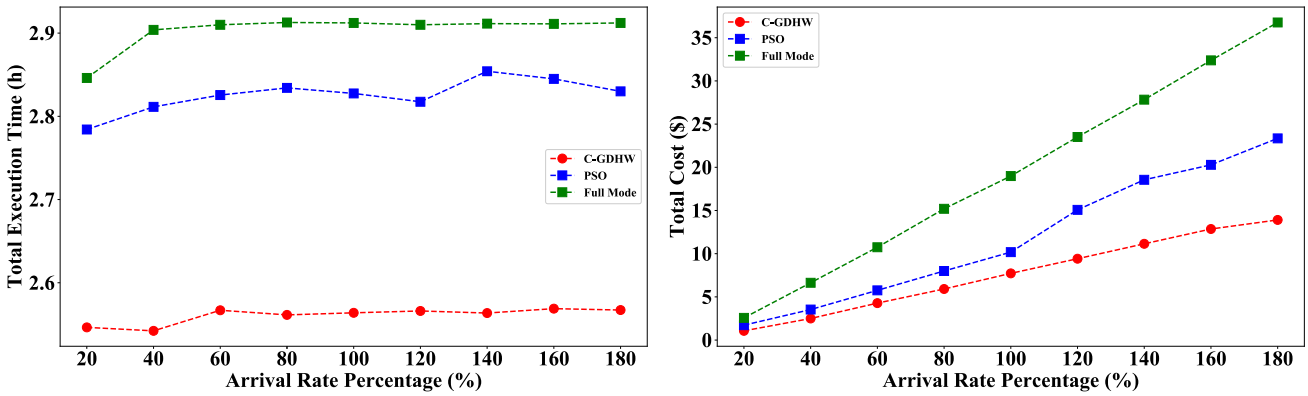

FIGURE 8. Arrival rate variation impact on the medium workflow.

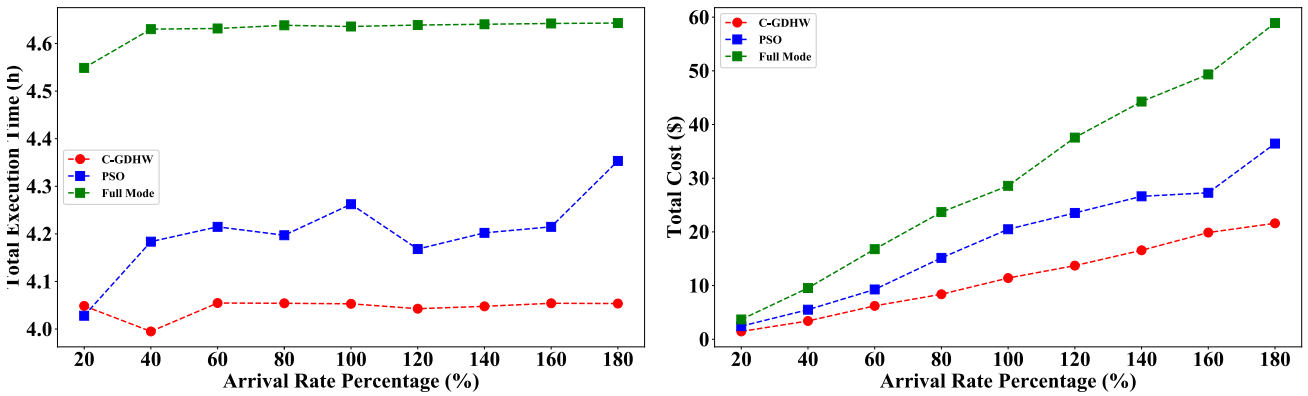

FIGURE 9. Arrival rate variation impact on the large workflow.

2) ARRIVAL RATE

Figures 7, 8 and 9 compare the behaviour of the three estimation techniques regarding the variation of stream arrival rate. Under all hybrid workflow execution scenarios, the C-GDHW shows steady control over the incoming arrival rate under the constraints of the queuing system and throughput. The results revealed C-GDHW capability to maintain a minimal increase in the execution time curve regardless of the stream speed. The PSO optimizer requires more execution time in response to change in stream speed, particularly with long-term execution workflow.

Figure 7 shows that for a small workflow with 12 stream tasks, both estimators demonstrate relatively similar cost-saving behaviour. Contrarily, with 36 stream tasks, for large workflow in Figure 9, the proportional difference in cost reduction is increased as the arrival rate is getting higher. Reduction percentage increased from 35% to 70%, in the case of 40% and 180% arrival rate, respectively. An additional

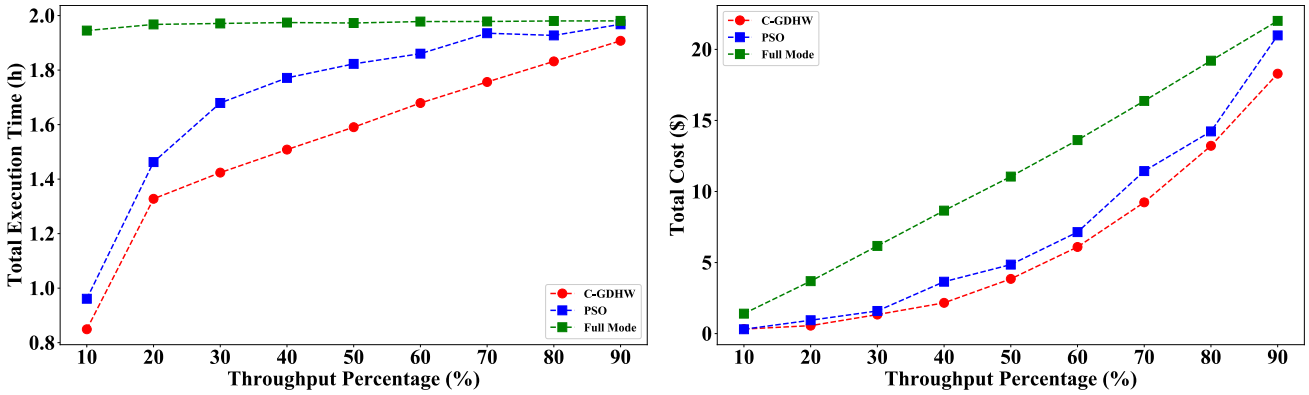

FIGURE 10. Throughput variation impact on the small workflow.

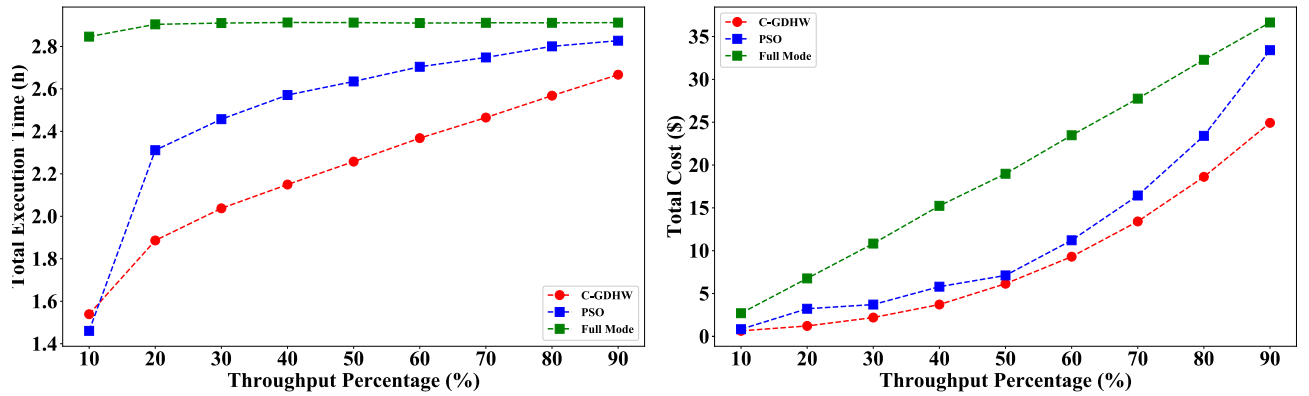

FIGURE 11. Throughput variation impact on the medium workflow.

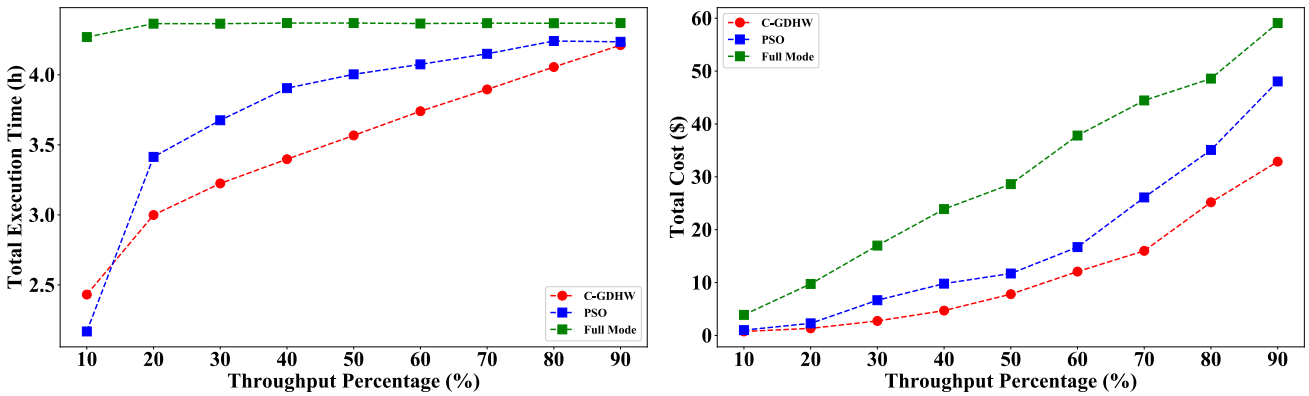

FIGURE 12. Throughput variation impact on the large workflow.

advantage of the proposed optimizer is the linearity behaviour in provisioning more resources (related to the monetary cost) to overcome high rate streams.

3) THROUGHPUT

In this work, we studied the contribution of application throughput on workflow execution time and cost. The proposed optimizer was capable of reducing the influence of high throughput, particularly for the monetary cost. For small workflows, as shown in Figure 10, the cost increases exponentially with the increase in throughput. Interestingly, the C-GDHW optimizer shows ability to control the cost (efficient resource utilization) as workflow model is getting more complex as depicted in Figures 11 and 12.

For execution time, the proposed optimizer linearly handles the increase in application throughput. Likewise, the optimizer produced potential cost saving compared to PSO of 46% with the highest throughput of 90%.

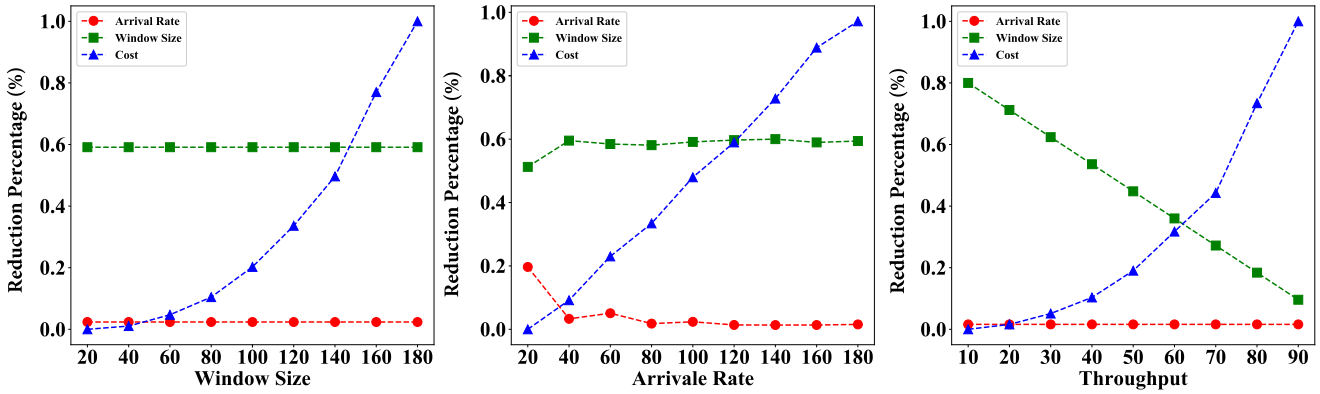

FIGURE 13. Arrival rate and window size reduction with cost increase: large workflow.

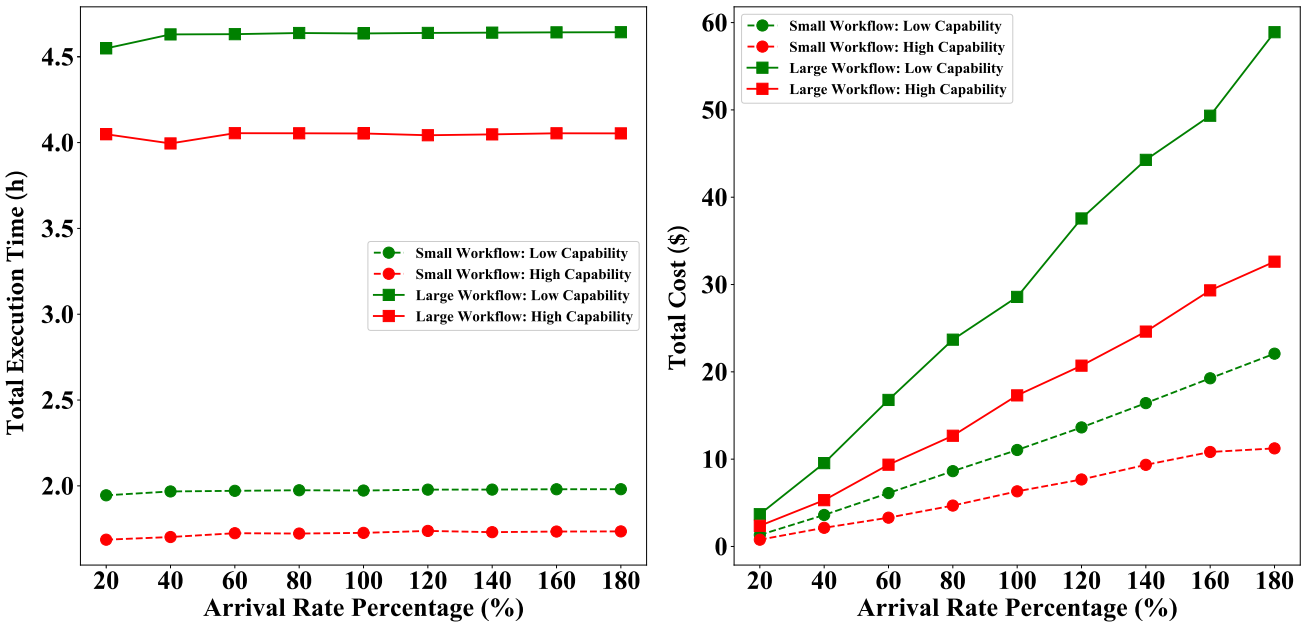

FIGURE 14. Compare the variation of execution time for small and large workflows based on edge capability and change in arrival rate.

Furthermore, as the system progresses to high throughput, the C-GDHW and PSO are not sufficiently outperforming the full execution mode in reducing the end-to-end workflow execution time. This clearly stated the contribution of migrating more data to the cloud since edge nodes do not collaborate to handle data flooding from IoT devices. Moreover, the high workload at the network edge can degrade the performance of delay-sensitive applications [62]. Thus, a problem still to be solved is how to support the execution of delay-sensitive applications at the edge layer.

4) ADAPTABILITY ANALYSIS

We studied the C-GDHW optimizer behavior in reducing stream task arrival rate and window size to optimize workflow execution cost. Such reduction in optimization parameters has a direct impact on workflow execution outcomes. For example, in predictive analytics, the amount of data plays a vital role in training prediction models and produce highly accurate results. On the other hand, processing more data produces additional overhead on computation cost. Thus, it is essential to provide a balanced estimation to handle the trade-off between performance and cost. Figure 13 demonstrates the optimizer potential in providing a stable arrival rate and window size reduction for a large-scale hybrid workflow. For example, with variation in throughput, the optimizer provides the ability to preserve stable reduction for the arrival rate and linear reduction for the windows size in responding to the exponential cost increase.

5) EDGE CAPABILITY ANALYSIS

Figure 14 shows the relation between the capability of edge resources and workflow execution time and cost with the variation in stream arrival rate. Edge capability indicates the computation performance of edge resources. Table 4 provides

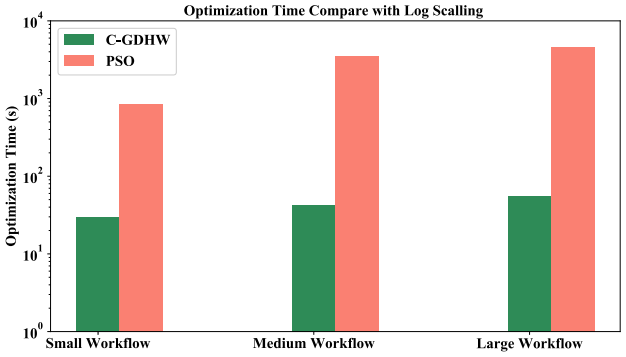

FIGURE 15. Compare optimization time between PSO and C-GDHW for different hybrid workflows (log scale).

details about low and high edge resource configuration, low capability refers to the small and medium types and high capability refers to large and xlarge VM types, respectively. Despite variation in stream arrival rate, results indicate a steady workflow execution time with low and high edge capability, with a 15% average difference and minimal contribution of the workflow complexity. On the other hand, the figure implies a linear increase in monetary cost, particularly for the large hybrid workflow scenario. The cost increases gradually from 56% (lowest arrival rate) to 8% (highest arrival rate). Overall, the graph shows that as the stream arrival rate increases, dependence on high-performance edge machines becomes costly compared to low-performance machines. This conclusion will motivate researchers to consider and move toward commodity edge computing.

6) SCALABILITY ANALYSIS

The increase in optimization time can have a significant impact on the scheduling process, particularly for large workflows. In a more complex scenario, the scheduling process could be repeated at short intervals, for example, to provide an accurate decision about model parameters, such as arrival rate and window size. Figure 15 shows the advantage of the proposed estimation algorithm, over our previous work using the PSO algorithm, in sustaining the optimization time with variation in workflow complexity and number of tasks. The optimization time increases by 45%, 55% and 60% for small, medium and large workflows, respectively. For the GDS-based optimizer, the increase in optimization time remains constant with 25% on average as the workflow structure is getting more complex, which reveal the scalability of the proposed scheduling. Moreover, The low optimization time of the GDS-based optimizer suggests that linear optimization models should be adopted to solve complex workflow estimation and scheduling problems.

VI. CONCLUSION

We presented a hybrid workflow scheduling framework for an edge cloud computing resource model. The framework includes two stages. In the first stage, we proposed a resource estimation algorithm based on a linear optimization approach, gradient descent search (GDS), while in the second stage, we proposed a cluster-based provisioning and scheduling technique for hybrid workflows in heterogeneous edge cloud resources. The aim was to achieve a multi-objective optimization of execution time and monetary cost under constraints of deadline and throughput. In comparison to the two nominated scheduling techniques, namely, PSO and Full-mode, the proposed technique performed better to control the execution of hybrid workflows by efficiently tuning several parameters, including stream arrival rate, processing throughput and workflow complexity. For large and complex workflows, the proposed scheduler provides significant improvement in terms of execution time and monetary cost in comparison to the PSO-based optimizer with an average of 8% and 35%, respectively. Moreover, the GDS-based technique shows a significant reduction in the optimization complexity compared to the PSO technique, with 50% on average.

For future work, we are planning to experiment with hybrid workflow provisioning and scheduling on cooperative edge cloud computing. We are also planning to extend the work to provide a convenient cost model from user and service provider perspectives.

REFERENCES

[1] S. Aheleroff, X. Xu, Y. Lu, M. Aristizabal, J. P. Velásquez, B. Joa, and Y. Valencia, "IoT-enabled smart appliances under industry 4.0: A case study," *Adv. Eng. Informat.*, vol. 43, Jan. 2020, Art. no. 101043.

[2] F. Cirillo, D. Gomez, L. Diez, I. Elicegui Maestro, T. B. J. Gilbert, and R. Akhavan, "Smart city IoT services creation through large-scale collaboration," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5267–5275, Jun. 2020.

[3] A. Girma, N. Bahadori, M. Sarkar, T. G. Tadewos, M. R. Behnia, M. N. Mahmoud, A. Karimoddini, and A. Homaifar, "IoT-enabled autonomous system collaboration for disaster-area management," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 5, pp. 1249–1262, Jul. 2020.

[4] A. Taherkordi, F. Eliassen, and G. Horn, "From IoT big data to IoT big services," in *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 485–491.

[5] R. Tönjes, P. Barnaghi, M. Ali, A. Mileo, M. Hauswirth, F. Ganz, S. Ganea, B. Kjærgaard, D. Kuemper, S. Nechifor, and D. Puiu, "Real time IoT stream processing and large-scale data analytics for smart city applications," in *Proc. Poster Session, Eur. Conf. Netw. Commun.*, Jun. 2014, p. 10.

[6] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," *Distrib. Parallel Databases*, vol. 3, no. 2, pp. 119–153, Apr. 1995.

[7] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *J. Grid Comput.*, vol. 13, no. 4, pp. 457–493, 2015.

[8] M. R. Anawar, S. Wang, M. Azam Zia, A. K. Jadoon, U. Akram, and S. Raza, "Fog computing: An overview of big IoT data analytics," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–22, May 2018.

[9] R. Alsurdeh, R. N. Calheiros, K. M. Matawie, and B. Javadi, "Hybrid workflow provisioning and scheduling on edge cloud computing using a gradient descent search approach," in *Proc. 19th Int. Symp. Parallel Distrib. Comput. (ISPDC)*, Jul. 2020, pp. 68–75.

[10] A. I. Maarala, M. Rautiainen, M. Salmi, S. Pirttikangas, and J. Riekki, "Low latency analytics for streaming traffic data with apache spark," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct. 2015, pp. 2855–2858.

[11] R. Cao, W. Tu, C. Yang, Q. Li, J. Liu, J. Zhu, Q. Zhang, Q. Li, and G. Qiu, "Deep learning-based remote and social sensing data fusion for urban region function recognition," *ISPRS J. Photogramm. Remote Sens.*, vol. 163, pp. 82–97, May 2020.

[12] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact," *MIS Quart.*, vol. 36, no. 4, pp. 1165–1188, Dec. 2012.

[13] I. S. Udoh and G. Kotonya, "Developing IoT applications: Challenges and frameworks," *IET Cyber-Phys. Syst., Theory Appl.*, vol. 3, no. 2, pp. 65–72, Jun. 2018.
[14] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," in *Proc. IEEE 4th Int. Conf. eScience*, Dec. 2008, pp. 640–645.
[15] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[16] S. Kailasam, P. Dhawalia, S. J. Balaji, G. Iyer, and J. Dharanipragada, "Extending MapReduce across clouds with BStream," *IEEE Trans. Cloud Computing*, vol. 2, no. 3, pp. 362–376, Jul. 2014.
[17] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bull. IEEE Comput. Soc. Tech. Committee Data Eng.*, vol. 36, no. 4, pp. 1–12, 2015.
[18] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proc. 24th ACM Symp. Operating Syst. Princ.*, Nov. 2013, pp. 423–438.
[19] B. Pishgoo, A. A. Azirani, and B. Raahemi, "A hybrid distributed batch-stream processing approach for anomaly detection," *Inf. Sci.*, vol. 543, pp. 309–327, Jan. 2021.
[20] Y. Yao, J. Cao, S. Qian, and S. Feng, "Decentralized executions of privacy awareness data analytics workflows in the cloud," *Concurrency Comput., Pract. Exper.*, vol. 31, no. 15, p. e5063, Aug. 2019.
[21] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *Proc. 10th Int. Conf. Autonomic Comput. (ICAC)*, 2013, pp. 23–27.
[22] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *J. Grid Comput.*, vol. 12, no. 4, pp. 559–592, Dec. 2014.
[23] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, May 2013, pp. 67–78.
[24] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Towards an autonomic auto-scaling prediction system for cloud resource provisioning," in *Proc. IEEE/ACM 10th Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst.*, May 2015, pp. 35–45.
[25] N. Wagner and Z. Michalewicz, "An analysis of adaptive windowing for time series forecasting in dynamic environments: Further tests of the DyFor GP model," in *Proc. 10th Annu. Conf. Genetic Evol. Comput. (GECCO)*, 2008, pp. 1657–1664.
[26] M. Deypir, M. H. Sadreddini, and S. Hashemi, "Towards a variable size sliding window model for frequent itemset mining over data streams," *Comput. Ind. Eng.*, vol. 63, no. 1, pp. 161–172, Aug. 2012.
[27] D. Warneke and O. Kao, "Exploiting dynamic resource allocation for efficient parallel data processing in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 985–997, Jun. 2011.
[28] M. Adhikari, T. Amgoth, and S. N. Srirama, "A survey on scheduling strategies for workflows in cloud environment and emerging trends," *ACM Comput. Surveys*, vol. 52, no. 4, pp. 1–36, Sep. 2019.
[29] A. R. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Gener. Comput. Syst.*, vol. 91, pp. 407–415, Feb. 2019.
[30] M. Kumar, S. C. Sharma, A. Goel, and S. P. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *J. Netw. Comput. Appl.*, vol. 143, pp. 1–33, Oct. 2019.
[31] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[32] J. J. Durillo, H. M. Fard, and R. Prodan, "MOHEFT: A multi-objective list-based method for workflow scheduling," in *Proc. 4th IEEE Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2012, pp. 185–192.
[33] Z.-G. Chen, Z.-H. Zhan, Y. Lin, Y.-J. Gong, T.-L. Gu, F. Zhao, H.-Q. Yuan, X. Chen, Q. Li, and J. Zhang, "Multiobjective cloud workflow scheduling: A multiple populations ant colony system approach," *IEEE Trans. Cybern.*, vol. 49, no. 8, pp. 2912–2926, Aug. 2018.
[34] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generat. Comput. Syst.*, vol. 29, no. 1, pp. 158–169, 2013.
[35] J. Sahni and D. P. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 2–18, Mar. 2015.
[36] Q. Zhang, Y. Song, R. R. Routray, and W. Shi, "Adaptive block and batch sizing for batched stream processing system," in *Proc. IEEE Int. Conf. Autonomic Comput. (ICAC)*, Jul. 2016, pp. 35–44.
[37] A. Madej, N. Wang, N. Athanasopoulos, R. Ranjan, and B. Varghese, "Priority-based fair scheduling in edge computing," 2020, *arXiv:2001.09070*. [Online]. Available: http://arxiv.org/abs/2001.09070
[38] R. K. Naha, S. Garg, A. Chan, and S. K. Battula, "Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment," *Future Gener. Comput. Syst.*, vol. 104, pp. 131–141, Mar. 2020.
[39] Z. Zhou, X. Chen, W. Wu, D. Wu, and J. Zhang, "Predictive online server provisioning for cost-efficient IoT data streaming across collaborative edges," in *Proc. 20th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Jul. 2019, pp. 321–330.
[40] J. Huang, S. Li, and Y. Chen, "Revenue-optimal task scheduling and resource management for IoT batch jobs in mobile edge computing," *Peer Peer Netw. Appl.*, vol. 13, no. 5, pp. 1776–1787, Sep. 2020.
[41] V. Farhadi, F. Mehmeti, T. He, T. F. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 779–792, Apr. 2021.
[42] Q. Zhang, L. Gui, S. Zhu, and X. Lang, "Task offloading and resource scheduling in hybrid edge-cloud networks," *IEEE Access*, vol. 9, pp. 85350–85366, 2021.
[43] J. Wu, G. Zhang, J. Nie, Y. Peng, and Y. Zhang, "Deep reinforcement learning for scheduling in an edge computing-based industrial Internet of Things," *Wireless Commun. Mobile Comput.*, vol. 2021, pp. 1–12, Aug. 2021.
[44] V. Arabnejad, K. Bubendorfer, and B. Ng, "Scheduling deadline constrained scientific workflows on dynamically provisioned cloud resources," *Future Gener. Comput. Syst.*, vol. 75, pp. 348–364, Oct. 2017.
[45] X. Lin and C. Q. Wu, "On scientific workflow scheduling in clouds under budget constraint," in *Proc. 42nd Int. Conf. Parallel Process.*, Oct. 2013, pp. 90–99.
[46] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *Proc. IEEE Int. Conf. Smart Cloud (SmartCloud)*, Nov. 2016, pp. 20–26.
[47] J. Cao, Q. Zhang, and W. Shi, "Challenges and opportunities in edge computing," in *Edge Computing: A Primer*. Cham, Switzerland: Springer, 2018, pp. 59–70.
[48] K. Kanoun, M. Ruggiero, D. Atienza, and M. V. D. Schaar, "Low power and scalable many-core architecture for big-data stream computing," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2014, pp. 468–473.
[49] P.-O. Östberg, J. Byrne, P. Casari, P. Eardley, A. F. Anta, J. Forsman, J. Kennedy, T. L. Duc, M. N. Marino, R. Loomba, M. A. L. Pena, J. L. Veiga, T. Lynn, V. Mancuso, S. Svorobej, A. Torneus, S. Wesner, P. Willis, and J. Domaschka, "Reliable capacity provisioning for distributed cloud/edge/fog computing applications," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2017, pp. 1–6.
[50] V. Scoca, A. Aral, I. Brandic, R. D. Nicola, and R. B. Uriarte, "Scheduling latency-sensitive applications in edge computing," in *Proc. 8th Int. Conf. Cloud Comput. Services Sci.*, 2018, pp. 158–168.
[51] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *Proc. 3rd IEEE Workshop Hot Topics Web Syst. Technol. (HotWeb)*, Nov. 2015, pp. 73–78.
[52] L. Kleinrock, *Queueing Systems: Computer Applications*, vol. 2. New York, NY, USA: Wiley, 1976.
[53] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Rec.*, vol. 34, no. 4, pp. 42–47, 2005.
[54] P. Hokstad, "On the steady-state solution of the M/G/2 queue," *Adv. Appl. Probab.*, vol. 11, no. 1, pp. 240–255, Mar. 1979.
[55] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[56] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*. [Online]. Available: http://arxiv.org/abs/1609.04747
[57] J. Fliege and B. F. Svaiter, "Steepest descent methods for multicriteria optimization," *Math. Methods Oper. Res.*, vol. 51, no. 3, pp. 479–494, 2000.
[58] L. F. Bittencourt, A. Goldman, E. R. M. Madeira, N. L. S. da Fonseca, and R. Sakellariou, "Scheduling in distributed systems: A cloud computing perspective," *Comput. Sci. Rev.*, vol. 30, pp. 31–54, Nov. 2018.

[59] R. Alsurdeh, R. N. Calheiros, K. M. Matawie, and B. Javadi, "Cloud resource provisioning for combined stream and batch workflows," in *Proc. IEEE 37th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Nov. 2018, pp. 1–8.
[60] A. Shukla, S. Chaturvedi, and Y. Simmhan, "RIoTBench: An IoT benchmark for distributed stream processing systems," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 21, p. e4257, Nov. 2017.
[61] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.
[62] E. Ahmed, A. Ahmed, I. Yaqoob, J. Shuja, A. Gani, M. Imran, and M. Shoaib, "Bringing computation closer toward the user network: Is edge computing the solution?" *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 138–144, Nov. 2017.

RAED ALSURDEH received the master's degree in computer science and information system from the Arab Academy for Banking and Financial Sciences University, in 2007. He is currently pursuing the Ph.D. degree with the School of Computer, Data and Mathematical Sciences, Western Sydney University. He worked as an ICT Trainer with the New Horizons Learning Centres, Saudi Arabia, from 2010 to 2015. He had participated in several IT research and development projects. His research interests include hybrid workflow scheduling, big data analytics, edge cloud computing, and streaming processing.

RODRIGO N. CALHEIROS (Senior Member, IEEE) is currently a Senior Lecturer with the School of Computer, Data and Mathematical Sciences, Western Sydney University, Australia. He has been conducting research in the area of cloud computing, since 2008, and contributed to diverse aspects in the field, including multiclouds, energy-efficient cloud computing, and edge computing. He is also one of the original designers and developers of CloudSim, a widely used simulator of cloud environments. He has coauthored more than 70 papers, which attracted together 12,000 Google Scholar citations. His research interests include big data, the Internet of Things, internet computing, and their application. He is a fellow of the Higher Education Academy, U.K., and a Senior Member of the ACM.

KENAN M. MATAWIE received the B.Sc. (Hons.), M.Stat., and Ph.D. degrees from the University of New South Wales, Australia. In the last 20 years, he has held a range of faculty appointments at the University of New South Wales, the University of Technology Sydney, AMP Society, and the University of Newcastle, where he is currently a Senior Lecturer of statistics and data science with Western Sydney University. His research interests include the application of statistics, especially statistical modeling, advance data analysis, and developing statistical methods motivated by real-world problems. Most of his research was generated from application in computing, business, management, psychology, actuarial, medicine, geology, engineering, and information technology. These research contributions have been published and presented in more than 70 international journals and conferences. He is a member of many professional societies particularly and has been a Committee and Representative Member of the International Statistical Modelling Society, since 2005.

BAHMAN JAVADI (Senior Member, IEEE) is currently an Associate Professor in networking and cloud computing with Western Sydney University, Australia. He is the Co-Founder of the Failure Trace Archive, which serves as a Public Repository of failure traces and algorithms for distributed systems. Prior to this appointment, he was a Research Fellow with the University of Melbourne and a Postdoctoral Fellow with the INRIA Grenoble Rhone-Alpes, France. He has published more than 100 papers in high quality journals and international conferences and received numerous Best Paper Awards at IEEE/ACM conferences for his research papers. His research interests include cloud computing, edge computing, performance evaluation of large-scale distributed computing systems, and reliability and fault tolerance. He is a Senior Member of ACM and a Senior Fellow of the Higher Education Academy, U.K. He served as a program committee for many international conferences and workshops. He has presented many keynote and invited talks in several conferences and universities around the world. He has also guest edited many special issue journals.