Cluster Computing
https://doi.org/10.1007/s10586-021-03327-y

Orchestrating real-time IoT workflows in a fog computing environment utilizing partial computations with end-to-end error propagation

Georgios L. Stavrinides[1] · Helen D. Karatza[1]

Received: 26 March 2021 / Revised: 16 May 2021 / Accepted: 31 May 2021

Abstract
With the explosive growth of the Internet of Things (IoT), fog computing emerged as a new paradigm, in an attempt to minimize network latency. Fog computing extends the cloud to the network edge, closer to where the IoT data are generated. Typically, fog resources are of limited capacity. On the other hand, IoT applications are becoming more and more complex and computationally demanding, requiring a certain level of Quality of Service (QoS) within strict time constraints. In such a real-time setting, it is often more desirable for a job to meet its deadline by producing an approximate—but still of acceptable quality—result, rather than producing an overdue precise result. Based on this concept, in this paper we examine the orchestration of real-time IoT workflows in a heterogeneous fog computing environment, utilizing partial computations. When a workflow task produces an imprecise result, the error may be propagated not only to its immediate child tasks, but also across subsequent successor tasks of the workflow, ultimately affecting its end-result. The proposed scheduling technique is compared to a baseline algorithm, where partial computations are not used, under various result precision thresholds and input error propagation probabilities. The simulation results reveal that the proposed heuristic can provide on average a 32.71% lower deadline miss ratio than the baseline policy, by trading off an average result precision of 2.43%.

Keywords Fog computing · IoT workflows · Real-time scheduling · Partial computations · Error propagation · Performance evaluation

1 Introduction

The rapid technological advances continue to contribute to the ongoing growth of the Internet of Things (IoT). More and more everyday objects, such as sensors, actuators and mobile devices are connected to the Internet, generating at staggering speeds an unprecedented volume and variety of data. It is often necessary to transfer the generated data from the IoT layer to centralized cloud data centers. However, this would lead to heavy data traffic and significant service delays [8, 33, 51].

In an attempt to tackle this problem, fog computing emerged as a new paradigm. The fog supplements and extends the cloud to the network edge, close to where the IoT data are generated. The data are processed by fog nodes that are in physical proximity to the IoT layer, instead of being transferred to the cloud [10, 29]. As fog computing shares many characteristics with the cloud computing paradigm, such as resource pooling and virtualization, a fog node can often be a virtual machine (VM) [19].

IoT data usually require processing within firm deadlines, in a timely manner. In such a *real-time* setting, the correctness and usefulness of the computations depend not only on their logical results, but also on the time at which the results are produced [5, 47]. Some examples include weather forecasting, healthcare monitoring, road and air traffic control, drug discovery, as well as the supervision of critical infrastructures, such as power grids and water supply systems [3, 7].

✉ Georgios L. Stavrinides
gstavrin@csd.auth.gr

Helen D. Karatza
karatza@csd.auth.gr

1 Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

The data generated by such IoT applications are usually processed by real-time complex jobs, consisting of tasks with precedence constraints and data dependencies among them. Thus, the component tasks of a job typically form a *workflow* with an end-to-end deadline, where the output data of a task are used as input by other subsequent tasks. An *entry task* is a workflow task without any parent tasks, whereas an *exit task* is a task without any child tasks. In order to start execution, an entry task requires input data from the IoT layer. Subsequent tasks cannot start execution unless all of their parent tasks have been completed. Furthermore, all of their required input data must be transferred to the fog node where their processing will take place (if not already locally available) [11].

The effective orchestration and scheduling of such workloads constitute one of the most important aspects of a fog computing platform [4]. In its general form, the scheduling problem in a distributed environment concerns the assignment of a set of tasks to a set of interconnected computational resources. Its goal is to process all tasks in such an order, so that one or more objectives are met [21]. In this general form the scheduling problem is NP-hard. Therefore, heuristic methods are typically employed in an attempt to provide a near-optimal solution [14].

1.1 Motivation

Typically, fog resources are of limited capacity. On the other hand, IoT applications are becoming more and more complex and computationally demanding, requiring a certain level of Quality of Service (QoS) within strict time constraints. In such a real-time setting, it is often more desirable for a job to meet its deadline by producing an approximate result, rather than producing an overdue precise result.

Based on this observation, Lin et al. proposed the *partial (imprecise or approximate) computations* technique [22]. According to this approach, a real-time job is allowed to return imprecise—but still of acceptable quality—results when its deadline cannot be met [50]. Partial computations can be utilized especially in the case of applications with *monotone* component tasks, where the quality of a task's results is improved as more time is spent to produce them (e.g., statistical estimation and video processing tasks) [16].

Each monotone task typically consists of a *mandatory part*, followed by an *optional part*. In order for a task to return an acceptable result of the minimum quality, its mandatory part must be completed, whereas the optional part enhances the result produced by the mandatory part [35]. Consequently, this technique provides flexibility, by trading off precision for timeliness. It is important to note that in the case of workflow jobs, when a component task produces an imprecise result which is then consumed by its child tasks, the effects of input error should be taken into account.

Partial computations have only recently been used in fog computing platforms [25, 26]. Nevertheless, none of these research efforts took into account the effects of input error. On the other hand, previous works in non-fog environments, even though they considered the impact of input error, they generally assumed that it affected only the immediate child tasks of the tasks that provided the imprecise results. That is, they typically considered that the error could always been compensated by the child tasks that received it in their input data. However, this is not always the case. When a workflow component task produces an approximate result, the error may be propagated not only to its immediate child tasks, but also across subsequent successor tasks of the workflow, ultimately affecting its end-result. Consequently, the case of end-to-end error propagation across the tasks of a workflow, along with its impact on the processing time of each affected task, should be investigated.

1.2 Contribution

To this end, in this paper we examine the orchestration of multiple real-time IoT workflows in a heterogeneous fog computing environment, utilizing partial computations with end-to-end error propagation. In this context, we propose a partial computations and error propagation model and a dynamic scheduling heuristic. The proposed scheduling technique also takes into account the impact of transferring initial IoT input data from the IoT layer to the fog layer. It is compared to a baseline algorithm, where partial computations are not used, under various result precision thresholds and input error propagation probabilities. The experimental evaluation is performed via simulation. Several metrics are employed in order to gain useful insights into how the proposed approach performs.

To the best of our knowledge, in the context of partial computations, error propagation across not only the immediate child tasks, but also subsequent successor tasks in workflows, while also taking into account its impact on the processing time of each affected task, has never been investigated in the literature before—not only in a fog computing setting, but also in general.

The rest of the paper is organized as follows: Sect. 2 provides an overview of related literature. Section 3 presents the fog computing environment, the workload model, as well as the partial computations and error propagation model. The proposed scheduling heuristic is described in Sect. 4. Section 5 presents the employed evaluation metrics, describes the experimental setup and analyzes the simulation results. Finally, Sect. 6 summarizes and concludes the paper, providing directions for future work.

2 Related work

Scheduling in fog computing environments is a relatively new research area. However, a large body of work has already been focused on this domain [9, 20, 24, 31]. On the other hand, only a small number of scheduling methodologies that utilized partial computations in fog computing platforms have been investigated in the literature.

2.1 Scheduling in fog environments with no partial computations

Our review of previous research efforts on scheduling in fog environments (with no partial computations) is concentrated on three important aspects related to this work:

(a) Whether the workload under study consisted of *workflow* jobs.
(b) Whether workload *deadlines* were taken into account.
(c) Whether the proposed methodology considered the performance impact of transferring *initial IoT input data* from the IoT layer to the fog layer, which is typically required in order to start processing the workload on the fog resources.

2.1.1 Generic efforts

Wu and Lee [48] presented a novel scheduling heuristic based on an integer linear programming model, in order to minimize energy consumption in a heterogeneous fog architecture with IoT jobs. They showed through simulations that the proposed energy minimization scheduling algorithm could achieve near-optimal energy efficiency while maintaining fast execution times. However, no workflows, deadlines or initial IoT input data were considered.

De Souza Toniolli and Jaumard [12] presented an adaptation of the path-clustering heuristic in a fog-cloud environment for the scheduling of multiple workflows. The objective was to achieve the best tradeoff between workflow makespan and monetary cost. The proposed approach was evaluated with extensive simulations. The results showed that the proposed heuristic achieved better performance compared to other methods, while keeping monetary costs at similar levels. Even though workflows were considered in their study, no deadlines or initial IoT input data were taken into account.

2.1.2 Efforts considering workflows with deadlines but no initial IoT input data

Pham et al. [32] proposed a static real-time workflow scheduling strategy based on the collaboration between fog and cloud computing. Their goal was to achieve a tradeoff between performance and monetary cost of the cloud resources. Their approach considered that both the performance and monetary cost factors had equal weight during the scheduling process. While workflows and deadlines were taken into account, no initial IoT input data were considered.

Xu et al. [49] presented a real-time task scheduling algorithm that took into account the laxity of the tasks, as well as the energy consumption of the fog and cloud resources. The proposed approach formulated the scheduling problem as a constrained optimization problem. An ant colony system algorithm was utilized for its solution. The experimental results revealed that the proposed method could lead to energy savings, while minimizing the number of deadline misses. Workflows and deadlines were considered. However, initial IoT input data were not taken into account.

Ahmed et al. [2] proposed an approach for scheduling multiple scientific workflows with deadlines in a multi-fog environment, taking into account the effects of Distributed Denial of Service (DDoS) attacks, which could keep fog resources busy. A hybrid optimization technique was proposed, based on particle swarm optimization and salp swarm algorithms. In order to address the effects of DDoS attacks on the fog resources, they developed two discrete-time Markov chain models. These models were used for computing the average available bandwidth and the average number of available VMs in the fog environments, respectively. The simulation results revealed that the proposed strategy decreased the number of deadline misses under DDoS attacks. While the workload consisted of workflows with deadlines, no initial IoT input data were taken into account.

2.1.3 Efforts considering deadlines and initial IoT input data but no workflows

Gazori et al. [17] focused on the scheduling of fog-based IoT applications with the objective of minimizing long-term service delay and computation cost under the resource and deadline constraints. The proposed approach utilized the reinforcement learning approach, using the target network and experience replay techniques. The evaluation results showed that the proposed algorithm outperformed baseline strategies, in terms of service delay, computation cost, energy consumption and deadline misses. Even though deadlines and initial IoT input data were taken into account, the workload consisted of independent tasks.

Naha et al. [28] addressed the problem of meeting deadline-based dynamic user requirements in a fog-cloud environment through resource allocation and scheduling algorithms. Resource ranking and provision of resources were used in a hybrid and hierarchical fashion. The simulation results demonstrated that the proposed approach outperformed existing algorithms, in terms of overall data processing time, monetary cost and network delay. Deadlines and initial IoT input data were taken into account. However, no workflows were considered.

Aburukba et al. [1] proposed an approach to minimize the overall latency of IoT workload in a fog-cloud environment. Integer linear programming was employed in order to model the minimum service time for IoT requests. They proposed a heuristic approach that utilized a customized genetic algorithm in order to obtain a feasible solution with a good quality in a reasonable computational time. However, while deadlines and initial IoT input data were taken into account, no workflows were considered.

2.1.4 Efforts considering workflows with deadlines and initial IoT input data

Ding et al. [13] proposed a cost-effective scheduling strategy for multiple parallel IoT workflows with time constraints in a fog-cloud environment. They proposed a novel multi-workflow scheduling policy based on particle swarm optimization. A fitness function was utilized in order to evaluate the workflow execution cost under given deadlines. The experimental results demonstrated that the proposed technique could significantly reduce the execution cost of multiple parallel workflows, compared to other strategies. In this work workflows with deadlines and initial IoT input data were considered.

In our previous work in [42, 44–46] we considered the scheduling of real-time IoT workflows in fog computing platforms. The performance impact of transferring initial IoT input data from the IoT layer to the fog layer was taken into account and incorporated into the scheduling heuristic. In this paper, we consider all of these aspects in our scheduling approach, along with the utilization of partial computations with end-to-end error propagation.

2.2 Scheduling workloads with partial computations

Scheduling workloads with partial computations has been studied extensively in the literature, with some works considering the effects of input error. However, only a few recent research efforts investigated partial computations in a fog environment, but without considering the effects of input error. Furthermore, to the best of our knowledge, end-to-end error propagation across the component tasks of workflows, while also taking into account its impact on the processing time of each affected task, has never been discussed in the literature before, not only in a fog computing environment, but also in general.

2.2.1 Efforts considering input error in non-fog environments

Feng and Liu [16] proposed a real-time scheduling technique utilizing partial computations. They investigated the impact of input error on the static scheduling of a single linear workflow of tasks with an end-to-end deadline on a single processor. When a parent task provided its child task with imprecise results, the mandatory and optional parts of the child task were extended, in order to handle and correct the error. That is, the case where the input error could propagate along the subsequent tasks of the linear chain was not considered.

Ravindran et al. [34] presented a heuristic for the scheduling of real-time workflows with partial computations on multiprocessors. The objective was to maximize output quality in the cases where the resources were limited in terms of time and energy. Even though a simplified model was proposed for calculating recursively the accumulated output error of the exit tasks, which resulted from the input error and partial completion of predecessor tasks, no proper propagation of the input error and its effects on the processing time of each affected workflow task were considered.

Esmaili et al. [15] proposed a real-time scheduling heuristic for workflows with partial computations and end-to-end deadlines on multiprocessor platforms. Their approach was evaluated against a mixed integer linear program formulation of the same problem, which provided the optimal reference scheduling solutions. The proposed heuristic was also capable of finding feasible schedules even under tight energy budgets. Through extensive simulation experiments, it was demonstrated that in some cases, the proposed approach yielded the same QoS as the ones found by the mixed integer linear program. Even though both strategies took into account the impact of input error in the case where a parent task's results were imprecise, it was considered that the input error could always be compensated by extending the mandatory part of each affected child task. Thus no end-to-end error propagation was taken into account.

In all of our previous research efforts [36, 37, 39–41, 43], even though we considered the impact of input error on the real-time scheduling of workflows with partial computations, no end-to-end error propagation was taken into account. Furthermore, since our approaches were not applied to a fog computing environment, no initial IoT input data were considered.

2.2.2 Efforts considering fog environments but no input error

Mora Mora et al. [27] proposed an approach that utilized partial computations in order to provide flexible implementation frameworks for embedded or mobile devices in fog-cloud platforms. The proposed real-time scheduling technique took into account the initial IoT input data of the workload. However, the workload consisted of independent tasks and not workflows. Furthermore, in the proposed framework, the tasks did not have mandatory and optional parts, but it was assumed that some of the tasks were mandatory and some optional. Consequently, no input error or end-to-end error propagation were considered.

Cao et al. [6] investigated the QoS optimization of real-time applications in fog computing systems equipped with reusable IoT end devices and powered by hybrid energy of renewable generations and grid electricity. They proposed a two-level scheduling approach that utilized partial computations. At the IoT layer, a local scheduling solution was produced by consecutively conducting application-level and component-level energy allocation. At the fog layer, a local-remote scheduling solution was subsequently derived via renewable-adaptive computation offloading. This research effort took into account the initial IoT input data of the workload. However, the real-time tasks were independent and no input error or error propagation were considered.

Mo and Kritikakou [25] proposed a mathematical model for the energy-efficient scheduling of deadline-constrained workflows in a fog cyber-physical networked system, utilizing partial computations. The problem was first formulated as a mixed integer non-linear programming problem, due to its complex nature. Subsequently, they provided a linearization method that resulted in a mixed integer linear programming formulation, in order to efficiently solve the problem. The simulation experiments demonstrated the effectiveness of the proposed approach. However, in this work, as in an extension of this work by Mo et al. [26], no initial IoT input data were considered. Furthermore, no input error or error propagation were considered among the workflow tasks.

All of the previous works analyzed above are shown in Table 1. They are categorized according to their characteristics related to this work.

3 Problem definition

3.1 Fog computing environment

The fog computing environment under study is depicted in Fig. 1. The first layer comprises the IoT sensors and devices, whereas the second layer consists of the fog computational resources. The IoT sensors and devices of the first layer transmit data to the fog resources in the second layer, in order to be processed. The fog layer contains a set $\mathcal{H} = \{h_1, ..., h_{|\mathcal{H}|}\}$ of $|\mathcal{H}|$ heterogeneous physical hosts. Each physical host h_i provides a pool of m_i VMs. Collectively, there is a set $\mathcal{V} = \{vm_1^{h_1}, ..., vm_{|\mathcal{V}|}^{h_{|\mathcal{H}|}}\}$ of $|\mathcal{V}| = \sum_{h_i \in \mathcal{H}} m_i$ heterogeneous VMs in the fog layer. The superscript of each VM in $\mathcal{V}$ indicates the physical host that the particular VM is run on. Hereafter, the superscripts of the VMs will be omitted for simplicity purposes. Each VM vm_i is assigned a virtual CPU (vCPU) with operating frequency f_i. It is noted that each vCPU—and thus each VM—has its own queue of assigned tasks that need to be processed.

It is assumed that the processors of the physical hosts, and thus the provided VMs, support the same instruction set. Consequently, they require the same number of clock cycles per instruction. This is a reasonable assumption, as the VMs utilized by major cloud vendors, such as Microsoft Azure, Google Cloud and Amazon Web Services, typically support the x86-64 instruction set. The VMs in the fog layer are fully connected by a virtual network.

The data transfer rate between two fog VMs vm_i and vm_j is denoted by $r_{\text{fog}_{ij}}$ and is uniformly distributed in the interval:

$$r_{\text{fog}_{ij}} \sim U[\rho_{\text{fog}} \times (1 - H_{\text{fog}}/2), \rho_{\text{fog}} \times (1 + H_{\text{fog}}/2)] \quad (1)$$

where H_{fog} and ρ_{fog} are the heterogeneity degree and the mean data transfer rate, respectively, of the virtual network in the fog layer.

The data between the IoT and fog layers are transmitted with a rate r_{IoT}, which is uniformly distributed in the range:

$$r_{\text{IoT}} \sim U[\rho_{\text{IoT}} \times (1 - H_{\text{IoT}}/2), \rho_{\text{IoT}} \times (1 + H_{\text{IoT}}/2)] \quad (2)$$

where H_{IoT} is the heterogeneity degree of the network that connects the IoT and fog layers, whereas ρ_{IoT} is the mean data transfer rate between the two layers.

The resource allocation and scheduling of the incoming workload on the available VMs in the fog layer is performed by a fog orchestrator, running on a dedicated node in the fog layer. The fog orchestrator has a global waiting queue where the tasks of all of the workflow jobs that arrive at the system wait until they become ready to be scheduled on the VMs. The queueing model of the fog resources is illustrated in Fig. 2.

3.2 Workload model

The data generated by the IoT sensors and devices in the first layer of the environment under study require

Table 1 Comparison of the characteristics of previous research efforts to the work presented in this paper

Reference	Fog	Workflows	Deadlines	Initial IoT input data	Partial computations	Input error	End-to-end error propagation
Wu and Lee [48]	✓						
De Souza Toniolli and Jaumard [12]	✓	✓					
Pham et al. [32]	✓	✓	✓				
Xu et al. [49]	✓	✓	✓				
Ahmed et al. [2]	✓	✓	✓				
Gazori et al. [17]	✓		✓	✓			
Naha et al. [28]	✓		✓	✓			
Aburukba et al. [1]	✓		✓	✓			
Ding et al. [13]	✓	✓	✓	✓			
Our previous works [42, 44–46]	✓	✓	✓	✓			
Feng and Liu [16]		✓	✓		✓	✓	
Ravindran et al. [34]		✓	✓		✓	✓	
Esmaili et al. [15]		✓	✓		✓	✓	
Our previous works [36, 37, 39–41, 43]		✓	✓		✓	✓	
Mora Mora et al. [27]	✓		✓	✓	✓		
Cao et al. [6]	✓		✓	✓	✓		
Mo and Kritikakou [25]	✓	✓	✓		✓		
Mo et al. [26]	✓	✓	✓		✓		
This work	✓	✓	✓	✓	✓	✓	✓

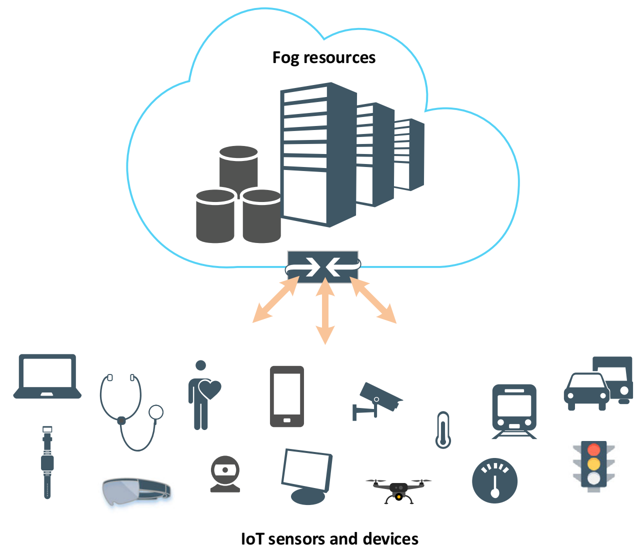

Fig. 1 The fog computing environment under study

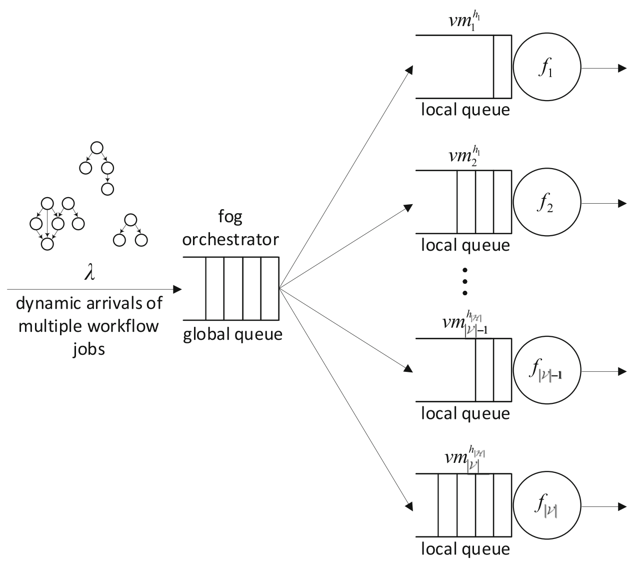

Fig. 2 The queueing model of the fog resources

processing within specific time constraints. Therefore, the data are transferred to the fog resources in the second layer, where they are processed by real-time workflows. Consequently, multiple real-time workflow jobs arrive dynamically at the fog orchestrator, in a Poisson stream with rate λ. Each workflow job is represented by a *directed acyclic graph (DAG)* $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the set of the nodes of the graph, whereas $\mathcal{E}$ is the set of the directed edges between the nodes. Each node represents a component task n_i of the workflow, whereas a directed edge e_{ij} between two tasks n_i and n_j represents the data that must be transferred from task n_i to task n_j. The number of tasks $|\mathcal{N}|$ in a workflow is an integer uniformly distributed in the interval:

$$|\mathcal{N}| \sim U[N_{\min}, N_{\max}] \tag{3}$$

where $N_{\min}$ and $N_{\max}$ are respectively the minimum and maximum number of tasks in a workflow job.

The component tasks of a workflow are considered to be non-preemptible, as preemption in a real-time context may eventually lead to performance degradation [5]. An example of a workflow job with three entry tasks and five exit tasks, is depicted in Fig. 3. The number in each node denotes the average computational cost of the task. The number on each edge denotes the average communication cost between the two component tasks it connects. The arrows pointing to the entry tasks of the graph denote their required initial input data, which are transferred from the IoT layer to the fog layer.

3.2.1 Computational characteristics

Each component task n_i of a workflow job has a weight w_i, which denotes its *computational volume*. The computational volume of each task is expressed as the number of clock cycles required to execute the instructions of the particular task. It is exponentially distributed with mean ω. Hence, the *computational cost* of task n_i on a VM vm_k is given by:

$$Comp(n_i, vm_k) = w_i / f_k \tag{4}$$

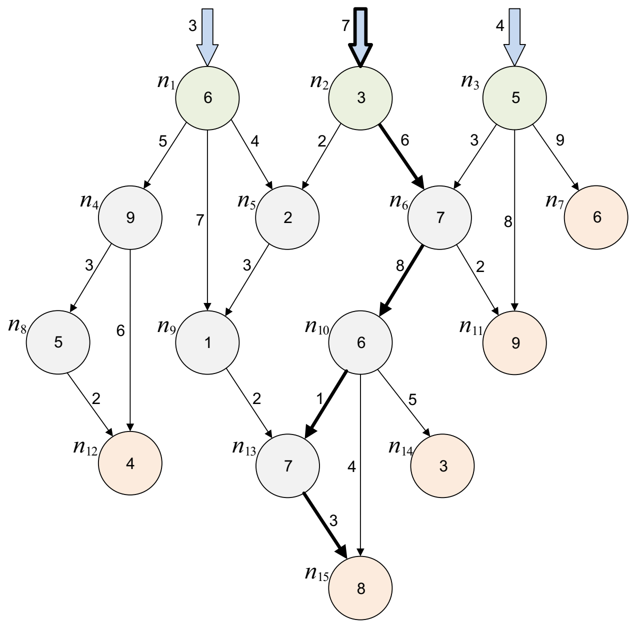

Fig. 3 A workflow job represented by a directed acyclic graph with three entry tasks and five exit tasks

where f_k is the operating frequency of vm_k.

3.2.2 Communication characteristics

An edge e_{ij} between two component tasks n_i and n_j has a weight z_{ij}, which represents its *communication volume*. The communication volume of each edge is expressed as the number of GB of data required to be transferred between the two tasks it connects. It is exponentially distributed with mean ζ. The *communication cost* of the edge e_{ij} is incurred when data are transferred from task n_i, scheduled on VM vm_k, to task n_j, scheduled on VM vm_l. It is defined as:

$$Comm((n_i, vm_k), (n_j, vm_l)) = z_{ij} / r_{\mathrm{fog}_{kl}} \tag{5}$$

where $r_{\mathrm{fog}_{kl}}$ is given by (1) and is the data transfer rate of the communication link between the two VMs, vm_k and vm_l. It is noted that in case both tasks n_i and n_j are scheduled on the same VM or on VMs that run on the same physical host, the communication cost of the edge e_{ij} is considered negligible.

Each entry task of a workflow job requires initial input data that are generated by the sensors and devices in the IoT layer. The *input data size* d_i of an entry task n_i is exponentially distributed with mean γ and is expressed in GB. The communication cost incurred by transferring the required input data from the IoT layer to an entry task n_i scheduled on a VM vm_k in the fog layer, is given by:

$$Comm(n_i, vm_k) = d_i / r_{\mathrm{IoT}} \tag{6}$$

where r_{IoT} is given by (2) and is the rate of transferring data from the IoT layer to the fog layer.

3.2.3 Other workload characteristics

The *communication to computation ratio CCR* of a workflow job is the ratio of its average communication cost to its average computational cost on the target environment. It is defined as:

$$CCR = \frac{\sum_{e_{ij} \in \mathcal{E}} \overline{Comm(e_{ij})}}{\sum_{n_i \in \mathcal{N}} \overline{Comp(n_i)}} \tag{7}$$

where $\mathcal{N}$ and $\mathcal{E}$ are the sets of the nodes and edges of the workflow job, respectively. $\overline{Comm(e_{ij})}$ is the average communication cost of the edge e_{ij} over all of the communication links that connect the fog VMs, whereas $\overline{Comp(n_i)}$ is the average computational cost of task n_i over all of the VMs in the fog layer.

The *makespan* M of a workflow job is its total processing time. It is defined as:

$$M = FT - ST \tag{8}$$

where FT is the *finish time* of the job, i.e., the time its last component task finished execution. On the other hand, ST is the *start time* of the job, i.e., the time its first component task started execution.

The *response time* RT of a workflow job is the time interval between the arrival of the job at the fog orchestrator and the completion of its last component task. It is given by:

$$RT = FT - AT \tag{9}$$

where AT is the *arrival time* of the job.

3.2.4 Real-time constraints

Each workflow job has a firm *end-to-end deadline* D within which all of its component tasks must finish processing. It is defined as:

$$D = AT + RD \tag{10}$$

where AT is the arrival time of the job, whereas RD is its *relative deadline*. The latter, is uniformly distributed in the range:

$$RD \sim U[CPL, 2CPL] \tag{11}$$

where CPL is the *critical path length* of the graph—i.e., it is the length of the longest path from an entry task to an exit task in the graph. The length of a path in the graph is the sum of the average computational and communication costs of all of the tasks and edges, respectively, on the path, including the input data communication cost of the respective entry task on the particular path. The critical path of the example workflow job in Fig. 3 is depicted with thick arrows.

3.3 Partial computations and error propagation model

The end-to-end deadline of each workflow job that arrives at the fog orchestrator must be met, otherwise its results would be useless. In case the deadline of a job is reached and not all of its component tasks have finished execution, the job would normally be considered lost and its unfinished component tasks would be discarded from the system. However, this can be addressed by trading off result precision for timeliness, using partial computations. Partial computations can allow a component task of a workflow job to be partially completed. In order to utilize partial computations, it is considered that the computational volume w_i of each workflow component task n_i consists of a *mandatory part* mp_i, followed by an *optional part* op_i:

$$w_i = mp_i + op_i \tag{12}$$

where $0 < mp_i < w_i$.

A task is completed when at least its mandatory part has been completed. The task can either complete its optional part entirely, partially or it may skip its whole optional part, depending on the decision of the orchestrator. The results of a partially completed task n_i are imprecise and therefore the task has *output error*. Since the output data of the task are used as input by its child tasks, the input data of the child tasks contain *input error*. Furthermore, there is a chance that a child task may not be able to correct the error in its input data (e.g., by performing more computations) and thus its input error is propagated to its output. This is determined by the *input error propagation factor* of the task.

Consequently, the output error of a task n_i is given by:

$$OE_i = \frac{\delta_i}{op_i} + \phi_i \times IE_i \tag{13}$$

where δ_i is the discarded fraction of the optional part op_i of the task, whereas ϕ_i is the input error propagation factor of the task. The output error of a task takes values in the interval [0, 1], i.e.:

$$0 \le OE_i \le 1 \tag{14}$$

The value of ϕ_i is determined by the *input error propagation probability* p, such that:

$$\phi_i = \begin{cases} 1, & \text{with probability } p \\ 0, & \text{with probability } 1-p \end{cases} \tag{15}$$

From (13) and (15) it is inferred that in case a task n_i has non-zero input error IE_i, there is a chance that its output error OE_i will be non-zero, even when it completes its whole optional part op_i, i.e., even when $\delta_i = 0$. From (13), (14) and the fact that the discarded fraction of the optional part of task n_i must be $\delta_i > \; = 0$, it follows that:

$$0 \le \delta_i \le op_i \times (1 - \phi_i \times IE_i) \tag{16}$$

This indicates that the fraction of the optional part of the task that can be omitted is dependent on the input error of the task that is going to be propagated to its child tasks.

The input error of a task n_i is considered to be equal to the average output error of its parent tasks, i.e.:

$$IE_i = \sum_{n_j \in \mathcal{P}_i} \frac{OE_j}{|\mathcal{P}_i|} \tag{17}$$

where $\mathcal{P}_i$ is the set of the parent tasks n_j of task n_i. From (14) and (17) it follows that:

$$0 \le IE_i \le 1 \tag{18}$$

If a task has input error, then there is an impact on its execution time. Additional computations (i.e., instructions and thus clock cycles) are required in order to handle the error and produce an acceptable result. For this reason, in such a case the mandatory part of the task is extended. Specifically, the *mandatory part extension* of a task n_i due to its input error is defined as:

$$mpe_i = mp_i \times IE_i \tag{19}$$

The mandatory part extension of the task is added to its initial mandatory part. It is noted that the optional part of the task is not affected by its input error.

The *result precision* of a task n_i is defined as:

$$RP_i = RPT + (1 - RPT)(1 - OE_i) \tag{20}$$

where RPT is the universal *result precision threshold*, under which the results of a task are not acceptable. Consequently, RPT constitutes an aspect of the required QoS level. The result precision of a task takes values in the interval [RPT, 1], i.e.:

$$RPT \le RP_i \le 1 \tag{21}$$

The result precision of a job is considered to be equal to the average result precision of its exit tasks, i.e.:

$$RP = \sum_{n_i \in \mathcal{N}_{\text{exit}}} \frac{RP_i}{|\mathcal{N}_{\text{exit}}|} \tag{22}$$

where $\mathcal{N}_{\text{exit}}$ is the set of exit tasks of the job.

From (13) and (20) it can be inferred that even when the whole extended computational volume of a task n_i is executed (i.e., $\delta_i = 0$), if there is propagated input error into the output of the task (i.e., $\phi_i \times IE_i > 0$), then the results of the task will still be imprecise (i.e., $RP_i < 1$). Hence, the input error of a task cannot always be compensated by the additional computations performed by the task due to its extended mandatory part. In case there is no propagated input error into the output of the task (i.e., $\phi_i \times IE_i = 0$), then the result precision threshold is equal to the ratio of the task's mandatory part over its computational volume, i.e.:

$$RPT = mp_i / w_i \tag{23}$$

It is noted that according to this partial computations and error propagation model, when a workflow task produces an imprecise result, its output error may be propagated not only to its immediate child tasks, but also across subsequent successor tasks of the workflow, ultimately affecting the end-result of the workflow job (i.e., there is end-to-end error propagation). The error propagation across the tasks of a workflow is measured by the *input error propagation index*, which is defined as:

$$IEPI = \frac{|\mathcal{E}'| + |\mathcal{N}'_{\text{exit}}|}{|\mathcal{E}| + |\mathcal{N}_{\text{exit}}|} \tag{24}$$

where $\mathcal{E}'$ is the set of edges that convey propagated input error from a parent task to a child task, $\mathcal{N}'_{\text{exit}}$ is the set of exit tasks that give results containing propagated input error, $\mathcal{E}$ is the set of edges, whereas $\mathcal{N}_{\text{exit}}$ is the set of exit tasks of the workflow job.

Consequently, in the case of partial computations, when the deadline of an uncompleted job is reached, the job is not always considered lost. In case the uncompleted tasks of the job are all exit tasks and all of them have completed their mandatory part and their output error is less than or equal to 1 (i.e., condition (14) holds), then the job is considered completed. In this case, even though the results of the job are imprecise, they are still of acceptable quality.

4 Scheduling methodology

A dynamic two-phased scheduling strategy is proposed for the orchestration of the workflow jobs onto the fog VMs, comprising a *task prioritization phase* and a *VM selection phase*. The component tasks of all of the workflow jobs that arrive at the system are first placed into the global waiting queue of the fog orchestrator. A task waits in the global waiting queue until it becomes ready to be scheduled. This happens when the task has either no predecessor tasks (i.e., it is an entry task of its respective workflow job) or all of its predecessor tasks have finished execution.

Whenever a task becomes ready to be scheduled, the orchestrator performs the two phases of the scheduling heuristic described below. The proposed method utilizes partial computations and idle schedule gaps in order to trade off result precision for timeliness. Its objective is to meet the deadline of each workflow job, providing results of acceptable quality—i.e., with precision above the defined result precision threshold RPT. Furthermore, the scheduling approach takes into account the impact of input error propagation on the execution time and output of the tasks.

4.1 Task prioritization phase

The ready tasks in the global waiting queue of the fog orchestrator are prioritized according to the end-to-end deadline D of their respective workflow job. The ready task with the earliest respective deadline has the highest priority for selection. Consequently, tasks are prioritized according to the *Earliest Deadline First (EDF)* policy. In case two or more tasks have the same respective deadline, the task with the largest average computational cost $\overline{Comp}$ is selected first. We use this tie-breaking rule, as it has been showcased in [30] that this method, when applied to the EDF policy, gives better results than other methods, such as random selection.

4.2 VM selection phase

Each ready task n_i is selected by the fog orchestrator according to its priority and is allocated to the VM vm_k that can provide it with the earliest *estimated finish time EFT*, which is defined as:

$$EFT(n_i, vm_k) = \max\{t_{\text{data}}(n_i, vm_k), t_{\text{idle}}(n_i, vm_k)\} + Comp(n_i, vm_k) \tag{25}$$

The term $t_{\text{data}}(n_i, vm_k)$ indicates the time at which all input data of the ready task n_i will be available on vm_k. It is noted that in case n_i is an entry task of its respective workflow job, the term $t_{\text{data}}(n_i, vm_k)$ concerns its initial IoT input data that should be transferred from the IoT layer to the fog layer. In all other cases, $t_{\text{data}}(n_i, vm_k)$ concerns the data generated by n_i's parent tasks. The term $t_{\text{idle}}(n_i, vm_k)$ provides an estimation of the time at which vm_k will be able to execute task n_i, according to the current state of its queue.

In order to determine the term $t_{\text{idle}}(n_i, vm_k)$, the following steps are performed:

1. First, we find the position at which the ready task n_i would be placed in the queue of vm_k, according to its priority. The term $t_{\text{idle}}(n_i, vm_k)$ is initially calculated based on this potential position.
2. In case all of the required input data of the ready task n_i are already available on vm_k, we check whether a schedule gap exists. A schedule gap is formed when vm_k is idle and the task n_q placed at the head of vm_k's queue is still in the process of receiving its required input data from other hosts or from the IoT layer. The capacity g of the schedule gap is given by:

$$g = t_{\text{data}}(n_q, vm_k) - t_{\text{current}} \tag{26}$$

where $t_{\text{data}}(n_q, vm_k)$ is the time at which all of the required input data of task n_q will be received, whereas t_{current} is the current time.

3. If a schedule gap exists, we determine the *maximum possible discarded fraction* $\delta_i^{\max}$ of n_i's optional part, according to (16), i.e.:

$$\delta_i^{\max} = op_i \times (1 - \phi_i \times IE_i) \tag{27}$$

In case n_i is an exit task of its respective workflow job, we consider that $\delta_i^{\max} = 0$. This is due to the fact that exit tasks ultimately determine the result precision of their respective jobs. Consequently, their whole computational volume should be allowed to be processed.

4. Subsequently, we try to fill in the schedule gap with the ready task n_i:

4.1 First, we check whether the whole task fits into the schedule gap, i.e.:

$$g \geq w_i / f_k \tag{28}$$

where w_i is the computational volume of task n_i, whereas f_k is the operating frequency of vm_k. In case condition (28) holds, n_i can be inserted into the gap. In this case the whole computational volume w_i of task n_i will be processed. The term $t_{\text{idle}}(n_i, vm_k)$ is recalculated based on n_i's new position.

4.2 In case the whole task does not fit into the schedule gap (i.e., condition (28) does not hold), we try to insert only a part of it, utilizing partial computations. Specifically, we check whether the minimum possible part of n_i fits into the schedule gap:

$$g \geq \left(w_i - \delta_i^{\max}\right) / f_k \tag{29}$$

Additionally, we check whether the processing time that can be saved by skipping a fraction of the ready task n_i's optional part equal to $\delta_i^{\max}$, is greater than or equal to the total average additional processing time that would be imposed by the extended mandatory part of n_i's child tasks, which is calculated by also considering n_i's output error:

$$\delta_i^{\max} / f_k \geq \sum_{n_j \in \mathcal{C}_i} \sum_{vm_l \in \mathcal{V}} \frac{\left(mp_j \times IE'_j\right) / f_l}{|\mathcal{V}|} \tag{30}$$

where $\mathcal{C}_i$ is the set of the child tasks n_j of the ready task n_i. IE'_j is the *potential input error* of a child task n_j. It is given by:

$$IE'_j = IE_j + \frac{\delta_i^{\max} / op_i + \phi_i \times IE_i}{|\mathcal{P}_j|} \tag{31}$$

where IE_j is the current input error of child task n_j, whereas $\mathcal{P}_j$ is the set of its parent tasks. In case both conditions (29) and (30) hold, only a part of the ready task n_i can be inserted into the schedule gap. The part of the task that would be processed in this case would be equal to its computational volume w'_i that fits into the gap, i.e.:

$$w'_i = g \times f_k \tag{32}$$

It is noted that we allow the maximum possible part (that fits into the schedule gap) of ready task n_i to be processed, in an attempt to minimize n_i's output error. The term $t_{\mathrm{idle}}(n_i, vm_k)$ is recalculated based on n_i's new position. In case the ready task n_i has to wait for input data or it does not fit into a schedule gap or a schedule gap does not exist, the position of task n_i in vm_k's queue—and thus the term $t_{\mathrm{idle}}(n_i, vm_k)$—is determined only by n_i's priority (i.e., as in step 1).

The pseudocode for the proposed *partial computations (PC)* scheduling technique is given in Algorithm 1. Schedule gaps are also utilized in the following cases, in the same manner as in the steps 2-4 of the VM selection phase:

(a) in the case where a task waiting in the queue of a VM has finished receiving its required input data,

(b) in the case where the processing of a task on a VM has been completed, and

(c) in the case where the deadline of a job is reached and thus its component tasks that are in service or waiting for data in VM local queues are discarded.

In case (a), if a schedule gap exists, we examine whether the particular task waiting for service can fit into the gap. In cases (b) and (c), if a schedule gap exists, it is utilized by a task waiting for service in the respective VM queue (eligible tasks are examined starting from the head of the queue).

Algorithm 1 The proposed two-phased partial computations (PC) scheduling technique.

```
Input: A set of ready workflow component tasks in the fog orchestrator global queue $\mathcal{N}_{\text{ready}} = \{n_1, ..., n_{|\mathcal{N}_{\text{ready}}|}\}$ and a set of fog VMs $\mathcal{V} = \{vm_1, ..., vm_{|\mathcal{V}|}\}$.
Output: A schedule of $\mathcal{N}_{\text{ready}}$ onto $\mathcal{V}$.
    //Task Prioritization Phase:
 1: Sort all ready tasks in $\mathcal{N}_{\text{ready}}$ in nondecreasing order of respective deadline $D$ values, according to the EDF policy (tie-breaking rule: the task with the largest average computational cost $\overline{Comp}$ has higher priority).
    //VM Selection Phase:
 2: repeat
 3:   Select the first unscheduled ready task $n_i \in \mathcal{N}_{\text{ready}}$.
 4:   for each fog VM $vm_k \in \mathcal{V}$ do
 5:     Calculate $t_{\text{data}}(n_i, vm_k)$.
        //Steps for calculating $t_{\text{idle}}(n_i, vm_k)$:
        //Step 1:
 6:     Determine $n_i$'s potential position in $vm_k$'s queue based on $n_i$'s priority.
 7:     Calculate $t_{\text{idle}}(n_i, vm_k)$ according to $n_i$'s potential position in $vm_k$'s queue.
        //Step 2:
 8:     $gapExists \leftarrow$ false
 9:     if $t_{\text{data}}(n_i, vm_k) = t_{\text{current}}$ then
10:       if $vm_k$ is idle and $t_{\text{data}}(n_q, vm_k) > t_{\text{current}}$ then
11:         $gapExists \leftarrow$ true
12:         $g \leftarrow t_{\text{data}}(n_q, vm_k) - t_{\text{current}}$
13:       end if
14:     end if
        //Step 3:
15:     if $gapExists$ then
16:       if $n_i$ is an exit task then
17:         $\delta_i^{\max} \leftarrow 0$
18:       else
19:         $\delta_i^{\max} \leftarrow op_i \times (1 - \phi_i \times IE_i)$
20:       end if
          //Step 4:
          //Step 4.1:
21:       if $g \geq w_i / f_k$ then
22:         $w_i' \leftarrow w_i$
23:         Recalculate $t_{\text{idle}}(n_i, vm_k)$ based on $n_i$'s new position.
          //Step 4.2:
24:       else if $g \geq (w_i - \delta_i^{\max})/f_k$ and
          $\delta_i^{\max}/f_k \geq \sum_{n_j \in \mathcal{C}_i} \sum_{vm_l \in \mathcal{V}} \frac{(mp_j \times IE_j')/f_l}{|\mathcal{V}|}$ then
25:         $w_i' \leftarrow g \times f_k$
26:         Recalculate $t_{\text{idle}}(n_i, vm_k)$ based on $n_i$'s new position.
27:       end if
28:     end if
29:     $EFT(n_i, vm_k) \leftarrow \max\{t_{\text{data}}(n_i, vm_k), t_{\text{idle}}(n_i, vm_k)\} + Comp(n_i, vm_k)$
30:   end for
31:   Remove ready task $n_i$ from the fog orchestrator global queue and assign it to the fog VM $vm_u$ that minimizes $n_i$'s $EFT$.
32:   Mark task $n_i$ as scheduled.
33: until all tasks in $\mathcal{N}_{\text{ready}}$ are scheduled.
```

For comparison purposes, a baseline policy is defined as a simplified version of the proposed heuristic, PC. Specifically, the baseline strategy has the same task prioritization phase as the proposed technique, but in the VM selection phase it utilizes schedule gaps only in the case where the whole task fits into the gap (i.e., only the steps 1-4.1 of the VM selection phase are performed). Hence, the baseline policy does not utilize partial computations.

5 Performance evaluation

The proposed scheduling technique, PC, was examined under several input error propagation probabilities p and result precision thresholds RPT. Specifically, we investigated the performance of the proposed heuristic for each combination of the following values of the two parameters, $p = \{0, 0.25, 0.5, 0.75, 1\}$ and $RPT = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. It is noted that in the case of the baseline scheduling policy, which was used for comparison purposes, the completed workflow jobs always gave precise results, since partial computations were not used. The particular baseline policy, through its comparison to the proposed heuristic, enabled us to focus on the impact of partial computations and propagated input error on the performance of the framework under study.

5.1 Evaluation metrics

In order to evaluate the performance of the proposed approach, the following metrics were employed:

1. *Deadline miss ratio*: It is the ratio of the number of workflow jobs that could not be completed within their deadline—and thus lost—over the number of jobs that arrived at the fog orchestrator during the observed time period.
2. *Weighted average result precision*: It is the weighted average result precision of the workflow jobs that were completed within the observed time period. The result precision of each completed workflow job was weighted using the number of exit tasks of the particular workflow.
3. *Weighted average makespan*: It is the weighted average makespan of the workflow jobs that were completed within the observed time period. The makespan of each completed workflow job was weighted using the critical path length of the particular workflow.
4. *Weighted average response time*: It is the weighted average response time of the workflow jobs that were completed within the observed time period. The response time of each completed workflow job was weighted using the critical path length of the particular workflow.

Furthermore, in order to gain additional insights into how other aspects of the proposed approach affected the framework under study, the following statistics were also calculated:

1. *Average percentage of tasks placed in schedule gaps*: It is the average percentage of tasks of completed workflow jobs that were placed in schedule gaps for processing by the fog orchestrator, within the observed time period.
2. *Weighted average input error propagation index*: It is the weighted average input error propagation index of workflow jobs that were completed within the observed time period. The input error propagation index of each completed workflow job was weighted using the total number of edges and exit tasks of the particular workflow.
3. *Percentage of partially completed jobs*: It is the percentage of completed workflow jobs that had at least one exit task partially completed upon deadline expiration—giving imprecise, but still acceptable results—during the observed time period.
4. *Average percentage of exit tasks with imprecise results*: It is the average percentage of exit tasks of completed workflow jobs that gave imprecise results, during the observed time period.
5. *Average percentage of exit tasks with imprecise results containing propagated input error*: It is the average percentage of exit tasks of completed workflow jobs that gave imprecise results containing propagated input error, during the observed time period.

5.2 Experimental setup

The performance evaluation of the proposed heuristic was performed using simulation rather than analytical methods. Analytical solutions would require several simplifying assumptions that would ultimately lead to misleading results. Even though simulation packages are available for fog environments, such as iFogSim [18] and FogWorkflowSim [23], we decided to implement our own custom discrete-event simulation program in C++, due to the complexity of the framework under study and in order to have full control over all of the model parameters. Synthetic workload was used instead of available traces, in order to obtain unbiased results, not restricted to only a particular type of workload.

The workflows were generated using our own random DAG generator, as described in [38]. Specifically, each generated DAG was a weakly connected graph, having a path between any pair of tasks, without taking into account the direction of the edges. In the case where a graph had more than one task, there was at least one entry and one exit task in the graph. In order to generate a DAG, we first determined the number of tasks $|\mathcal{N}|$ in the graph, as in (3), where the minimum and maximum number of tasks were $N_{\min} = 1$ and $N_{\max} = 64$, respectively. In case $|\mathcal{N}| > 1$, we determined the number of entry tasks $|\mathcal{N}_{\text{entry}}|$ in the graph, which was uniformly distributed in the interval $[1, |\mathcal{N}| - 1]$.

Algorithm 2 Creation of randomly structured workflows (DAGs).

```
Input: The minimum ($N_{\min}$) and maximum ($N_{\max}$) number of tasks in the DAG.
Output: A randomly structured DAG $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $N_{\min} \le |\mathcal{N}| \le N_{\max}$.
 1: Determine the number of tasks in the DAG,
    $|\mathcal{N}| \sim U[N_{\min}, N_{\max}]$.
 2: if $|\mathcal{N}| > 1$ then
 3:     Determine the number of entry tasks in the DAG,
        $|\mathcal{N}_{\text{entry}}| \sim U[1, |\mathcal{N}| - 1]$.
 4:     Create an entry task $n_1$.
 5:     Place $n_1$ in the set of created tasks $\mathcal{N}_{\text{created}}$.
 6:     repeat
 7:         Create a non-entry task $n_i$.
 8:         Determine the number of $n_i$'s parent tasks,
            $|\mathcal{P}_i| \sim U[1, |\mathcal{N}_{\text{created}}|]$.
 9:         Randomly select $n_i$'s parent tasks from $\mathcal{N}_{\text{created}}$ and place them in $\mathcal{P}_i$.
10:         Connect $n_i$ to each parent task $n_j \in \mathcal{P}_i$, by creating an edge $e_{ji}$.
11:         Place $n_i$ in the set of created tasks $\mathcal{N}_{\text{created}}$.
12:     until $|\mathcal{N}| - |\mathcal{N}_{\text{entry}}|$ non-entry tasks are created.
13:     if $|\mathcal{N}_{\text{entry}}| > 1$ then
14:         repeat
15:             Create an entry task $n_k$.
16:             Determine the number of $n_k$'s child tasks,
                $|\mathcal{C}_k| \sim U[1, |\mathcal{N}| - |\mathcal{N}_{\text{entry}}|]$.
17:             Randomly select $n_k$'s child tasks from the non-entry tasks and place them in $\mathcal{C}_k$.
18:             Connect $n_k$ to each child task $n_l \in \mathcal{C}_k$, by creating an edge $e_{kl}$.
19:         until the rest $|\mathcal{N}_{\text{entry}}| - 1$ entry tasks are created.
20:     end if
21: else
22:     Create the only task of the DAG, $n_1$.
23: end if
```

Subsequently, the first entry task n_1 was created, followed by the $|\mathcal{N}| - |\mathcal{N}_{\text{entry}}|$ non-entry tasks of the graph. The non-entry tasks were created one by one. At each step, a non-entry task n_i was created and connected to its parent tasks, which were randomly selected from the already created tasks, forming a set $\mathcal{P}_i$. The number of a non-entry task's parent tasks $|\mathcal{P}_i|$ was uniformly distributed in the interval $[1, |\mathcal{N}_{\text{created}}|]$, where $|\mathcal{N}_{\text{created}}|$ was the number of tasks that had already been created in the previous steps. After creating the $|\mathcal{N}| - |\mathcal{N}_{\text{entry}}|$ non-entry tasks (which some of them ended up being exit tasks), in case $|\mathcal{N}_{\text{entry}}| > 1$, the construction of the DAG continued, by creating the rest $|\mathcal{N}_{\text{entry}}| - 1$ entry tasks, one by one. At each step, an entry task n_k was created and connected to its child tasks, which were randomly selected from the non-entry tasks created in the previous steps, forming a set $\mathcal{C}_k$. The number of an entry task's child tasks $|\mathcal{C}_k|$ was uniformly distributed in the interval $[1, |\mathcal{N}| - |\mathcal{N}_{\text{entry}}|]$. The uniform distribution was used in the various steps of the generation of the task graphs, so that all DAG structures were equally probable. The pseudocode for the DAG generator is given in Algorithm 2.

As fog resources are typically limited and heterogeneous, we considered that there were $|\mathcal{H}| = 5$ physical hosts in the fog layer, providing a total of $|\mathcal{V}| = 64$ heterogeneous VMs. The computational characteristics of the physical hosts were based on real-world processors, as shown in Table 2. Each VM was assigned a vCPU, corresponding to a host's physical core. Since the workload in fog environments is usually communication intensive, the communication to computation ratio of the workflow jobs was chosen to be $CCR = 2$. For the same reason, the mean entry task input data size was selected to be $\gamma = 1$ GB. Due to the fact that IoT workflows processing vast amounts of data are also computationally intensive, the mean computational volume of the workflow component tasks was selected to be equal to $\omega = 8.93 \times 10^{11}$ clock cycles, so that on average, a task would take 5 minutes to execute on a fog VM. The mean edge communication volume was calculated using (7) and was equal to $\zeta = 44.74$ GB.

In order for the fog resources to be stable, the arrival rate of the workflow jobs was selected to be $\lambda = 0.0045$. The network heterogeneity degree in the IoT and fog layers was chosen to be equal to 0.5 (i.e., $H_{\text{IoT}} = 0.5$ and $H_{\text{fog}} = 0.5$), since typically most networks feature moderate heterogeneity. On the other hand, since the data transfer rate between the IoT and fog layers is usually lower than that between the fog resources, the mean data transfer rate between the two layers was selected to be $\rho_{\mathrm{IoT}} = 50$ Mbps, whereas the mean data transfer rate of the fog virtual network was chosen to be higher, at $\rho_{\mathrm{fog}} = 1$ Gbps. All of the input parameters and their respective values used in our simulation model are shown in Table 3. Table 4 includes all of the notations used in this paper.

The simulation experiments were conducted using the independent replications method. Specifically, we conducted 30 replications of the simulation for each set of input parameters, using different seeds of random numbers in each run. Each replication was terminated when 3×10^4 workflow jobs had been completed. We found by experimentation that this simulation run length was enough to minimize the effects of warm-up time. For every mean value, a 95% confidence interval was calculated. The half-widths of all of the confidence intervals were less than 5% of their respective mean values. Furthermore, in order to examine whether the differences between the obtained mean values were statistically significant, a 95% confidence interval was also calculated for the difference between each pair of mean values. The calculated confidence intervals did not include 0, which proved that the difference between each pair of mean values was statistically significant.

5.3 Simulation results analysis

The comparison between the proposed heuristic, PC, and the baseline policy, under various input error propagation probabilities p and result precision thresholds RPT, with regard to the deadline miss ratio metric, is shown in Fig. 4. It can be observed that, in all of the examined cases, the proposed approach, which leveraged partial computations, outperformed the baseline strategy, which did not utilize partial computations. Especially for low result precision thresholds, the deadline miss ratio decrease was more significant (up to 80.74%), as shown in Table 5. This was due to the fact that lower result precision thresholds allowed the proposed algorithm to utilize more effectively the schedule gaps by placing more (partial) tasks into them, compared to the baseline policy. Therefore fewer workflow jobs missed their deadline.

In contrast, for higher result precision thresholds, the difference in schedule gaps utilization between the two scheduling algorithms was less significant. However, still more gaps were utilized by the PC policy than the baseline strategy. This is illustrated in Fig. 5. In terms of the impact of input error propagation probability on the performance of the proposed heuristic, based on the simulation results shown in Fig. 4 and Table 5, it can be concluded that, in general, for higher values of p, more jobs missed their deadline. Nevertheless, even in the cases where there was always input error propagation across the component tasks of a job (i.e., $p = 1$), still more deadlines were met by the proposed approach than the baseline policy. This shows that the proposed heuristic was resilient to the effects of input error propagation among the component tasks of the workflows.

In Fig. 5, it can also be observed that for higher input error propagation probabilities, fewer tasks were placed in schedule gaps by the proposed policy. This can be explained by the fact that in the VM selection phase of the proposed heuristic, fewer tasks met the conditions defined in (29) and (30), since there was a greater chance that the output error of a task would propagate as input error to its child tasks. However, even for the highest input error propagation probability ($p = 1$), PC utilized more schedule gaps than the baseline policy, taking advantage of the flexibility provided by partial computations, which allowed the fog orchestrator to place partial tasks into the gaps. On the other hand, according to the baseline strategy, a schedule gap was utilized only when it could accommodate the whole task.

Figure 6 shows the comparison between the two scheduling approaches with respect to the weighted average result precision metric. With the PC heuristic, for lower result precision thresholds and higher input error propagation probabilities, the result precision of the jobs that met their deadline was lower (but still above the required threshold). Specifically, for $RPT = 0.1$ and $p = 1$ (i.e., the

Table 2 Characteristics of the physical hosts used in the fog resources model

Host	Real-world processor host was based on	No. of cores	Oper. freq.	No. of VMs
h_1	Intel Xeon Gold 5318H	18	2.5 GHz	18
h_2	Intel Xeon Gold 6226R	16	2.9 GHz	16
h_3	Intel Xeon Platinum 8354H	18	3.1 GHz	18
h_4	Intel Xeon Gold 6144	8	3.5 GHz	8
h_5	Intel Xeon Platinum 8256	4	3.8 GHz	4

case where there was always input error propagation), the proposed approach yielded a weighted average result precision 21.11% lower than the baseline policy, which always gave precise results as it did not utilize partial computations. However, as shown in Table 6, as the result precision threshold increased, the difference between the two scheduling strategies became more insignificant. For example, for $RPT = 0.9$, the decrease in the quality of the results was only 0.01%, compared to the baseline policy. The reason behind this was that when higher result precision thresholds were imposed, the optional part of the tasks was smaller, and hence the fraction of the optional part that could be discarded by the proposed approach was smaller as well.

Taking into account the simulation results in Figs. 4 and 6, it can be observed that the proposed heuristic traded off result precision for timeliness. However, the decrease in the job result precision was relatively insignificant compared to the decrease in deadline misses. Specifically, as shown in Tables 5 and 6, for the lowest result precision threshold ($RPT = 0.1$), the average deadline miss ratio decrease was 77.71%, whereas the average decrease in result precision was much smaller, at 8.83%. Similarly, for the highest result precision threshold ($RPT = 0.9$), the average deadline miss ratio decrease was 2.77%, which was still several orders of magnitude larger than the average decrease in result precision, which was 0.01%. Consequently, for a relatively insignificant loss in result quality, the proposed heuristic allowed more jobs to meet their deadline, compared to the baseline policy.

Figures 7 and 8 show the performance of the two scheduling algorithms with respect to the weighted average makespan and weighted average response time metrics, respectively. It can be observed that the two metrics followed a similar pattern: as the result precision threshold increased, the makespan and response time of the jobs increased in the case of the proposed heuristic. This was due to the fact that with higher result precision thresholds, fewer tasks could be placed in schedule gaps (as shown in Fig. 5) and therefore the total processing time of the jobs—and thus their response time—increased.

It can also be observed that for higher result precision thresholds, the PC strategy under some input error propagation probabilities yielded a slightly larger makespan (and thus response time) than the baseline policy. However, the maximum difference between the two approaches was negligible compared to the average makespan and average response time of the workflows. It was only 0.80 minutes in the case of makespan and only 0.72 minutes in the case of response time, whereas the average makespan and average

Table 3 Simulation input parameters

Parameter	Value
Fog Environment	
IoT Layer	
IoT-fog mean data transfer rate	$\rho_{\mathrm{IoT}} = 50$ Mbps
IoT-fog network heterogeneity degree	$H_{\mathrm{IoT}} = 0.5$
Fog Layer	
Number of physical hosts	$\lvert\mathcal{H}\rvert = 5$
Number of fog VMs	$\lvert\mathcal{V}\rvert = 64$
Fog VM vCPU operating frequency	$f = \{2.5, 2.9, 3.1, 3.5, 3.8\}$ GHz
Fog virtual network mean data transfer rate	$\rho_{\mathrm{fog}} = 1$ Gbps
Fog virtual network heterogeneity degree	$H_{\mathrm{fog}} = 0.5$
Workload Characteristics	
Number of completed workflows	3×10^4
Workflow arrival rate	$\lambda = 0.0045$
Minimum number of tasks per workflow	$N_{\mathrm{min}} = 1$
Maximum number of tasks per workflow	$N_{\mathrm{max}} = 64$
Mean entry task input data size	$\gamma = 1$ GB
Communication to computation ratio	$CCR = 2$
Mean task computational volume	$\omega = 8.93 \times 10^{11}$ clock cycles
Mean edge communication volume	$\zeta = 44.74$ GB
Result precision threshold	$RPT = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$
Input error propagation probability	$p = \{0, 0.25, 0.5, 0.75, 1\}$

Table 4 List of notations

Notation	Definition
$\mathcal{H}$	Set of physical hosts in the fog layer
$\|\mathcal{H}\|$	Number of physical hosts in the fog layer
h_i	A physical host in the fog layer
m_i	Number of VMs running on host h_i
$\mathcal{V}$	Set of VMs in the fog layer
$\|\mathcal{V}\|$	Number of VMs in the fog layer
vm_i	A VM in the fog layer
f_i	Operating frequency of VM vm_i
$r_{\mathrm{fog}_{ij}}$	Data transfer rate between fog VMs vm_i and vm_j
H_{fog}	Fog virtual network heterogeneity degree
ρ_{fog}	Fog virtual network mean data transfer rate
r_{IoT}	Data transfer rate between IoT and fog layers
H_{IoT}	IoT-fog network heterogeneity degree
ρ_{IoT}	IoT-fog network mean data transfer rate
λ	Arrival rate of workflow jobs
$\mathcal{G} = (\mathcal{N}, \mathcal{E})$	A workflow job represented as a DAG
$\mathcal{N}$	Set of tasks of a workflow job
$\|\mathcal{N}\|$	Number of tasks of a workflow job
$N_{\min}$	Minimum number of tasks per workflow job
$N_{\max}$	Maximum number of tasks per workflow job
$\mathcal{N}_{\mathrm{entry}}$	Set of entry tasks of a workflow job
$\|\mathcal{N}_{\mathrm{entry}}\|$	Number of entry tasks of a workflow job
$\mathcal{N}_{\mathrm{exit}}$	Set of exit tasks of a workflow job
$\|\mathcal{N}_{\mathrm{exit}}\|$	Number of exit tasks of a workflow job
$\mathcal{N}'_{\mathrm{exit}}$	Set of exit tasks of a workflow job that give results containing propagated input error
$\|\mathcal{N}'_{\mathrm{exit}}\|$	Number of exit tasks of a workflow job that give results containing propagated input error
$\mathcal{N}_{\mathrm{ready}}$	Set of ready tasks of a workflow job
$\|\mathcal{N}_{\mathrm{ready}}\|$	Number of ready tasks of a workflow job
$\mathcal{N}_{\mathrm{created}}$	Set of created tasks of a workflow job (during workflow generation)
$\|\mathcal{N}_{\mathrm{created}}\|$	Number of created tasks of a workflow job (during workflow generation)
$\mathcal{E}$	Set of edges of a workflow job
$\|\mathcal{E}\|$	Number of edges of a workflow job
$\mathcal{E}'$	Set of edges of a workflow job that convey propagated input error from a parent task to a child task
$\|\mathcal{E}'\|$	Number of edges of a workflow job that convey propagated input error from a parent task to a child task
n_i	A task of a workflow job
e_{ij}	An edge between tasks n_i and n_j of a workflow job
$\mathcal{P}_i$	Set of parent tasks of task n_i
$\|\mathcal{P}_i\|$	Number of parent tasks of task n_i
$\mathcal{C}_i$	Set of child tasks of task n_i
$\|\mathcal{C}_i\|$	Number of child tasks of task n_i
w_i	Computational volume of task n_i
w'_i	Computational volume of task n_i that fits into a schedule gap
ω	Mean task computational volume
$Comp(n_i, vm_k)$	Computational cost of task n_i on VM vm_k
z_{ij}	Communication volume of edge e_{ij}
ζ	Mean edge communication volume
$Comm((n_i, vm_k), (n_j, vm_l))$	Communication cost of edge e_{ij} when n_i is scheduled on VM vm_k and n_j is scheduled on VM vm_l
d_i	Input data size of entry task n_i

Table 4 (continued)

Notation	Definition
γ	Mean entry task input data size
$Comm(n_i, vm_k)$	Communication cost of IoT input data of entry task n_i scheduled on VM vm_k
CCR	Communication to computation ratio of a workflow job
$\overline{Comm(e_{ij})}$	Average communication cost of edge e_{ij} over all of the communication links that connect the fog VMs
$\overline{Comp(n_i)}$	Average computational cost of task n_i over all of the fog VMs
AT	Arrival time of a workflow job
ST	Start time of a workflow job
FT	Finish time of a workflow job
M	Makespan of a workflow job
RT	Response time of a workflow job
D	End-to-end deadline of a workflow job
RD	Relative deadline of a workflow job
CPL	Critical path length of a workflow job
mp_i	Mandatory part of task n_i
op_i	Optional part of task n_i
mpe_i	Mandatory part extension of task n_i
OE_i	Output error of task n_i
IE_i	Input error of task n_i
IE'_i	Potential input error of task n_i
δ_i	Discarded fraction of the optional part of task n_i
$\delta_i^{\max}$	Maximum possible discarded fraction of the optional part of task n_i
ϕ_i	Input error propagation factor of task n_i
p	Input error propagation probability
RP_i	Result precision of task n_i
RP	Result precision of a workflow job
RPT	Result precision threshold
$IEPI$	Input error propagation index of a workflow job
$EFT(n_i, vm_k)$	Estimated finish time of ready task n_i on VM vm_k
$t_{\text{data}}(n_i, vm_k)$	Estimated time at which all input data of ready task n_i will be available on VM vm_k
$t_{\text{idle}}(n_i, vm_k)$	Estimated time at which VM vm_k will be able to execute ready task n_i
t_{current}	Current time
g	Capacity of a schedule gap

response time of the workflows were 101.63 and 102.80 minutes, respectively. Moreover, in the case of real-time workloads, meeting deadlines is of the utmost importance. This does not necessarily mean minimizing at the same time the makespan and response time of the workload [5]. Consequently, in this context, the decrease in the deadline miss ratio provided by the proposed heuristic was significantly more important than the slight increase in the makespan and response time of the jobs, compared to the baseline policy.

Figure 9 shows the percentage of partially completed jobs in the case of the proposed scheduling heuristic, PC. A partially completed job occurred when its exit tasks could not be fully processed before the job's deadline. For lower result precision thresholds, due to the utilization of schedule gaps by the intermediate tasks of the workflows, the makespan of the jobs was smaller. Therefore, there was a greater chance for their exit tasks to be fully completed before their deadline. On the other hand, as the result precision threshold increased, it became less likely for the jobs to be fully completed upon their deadline. Hence, the number of partially completed jobs increased.

However, it can be observed that for moderate to high result precision thresholds, the percentage of partially completed jobs decreased. The reason behind this was that in those cases, the makespan of the jobs increased since

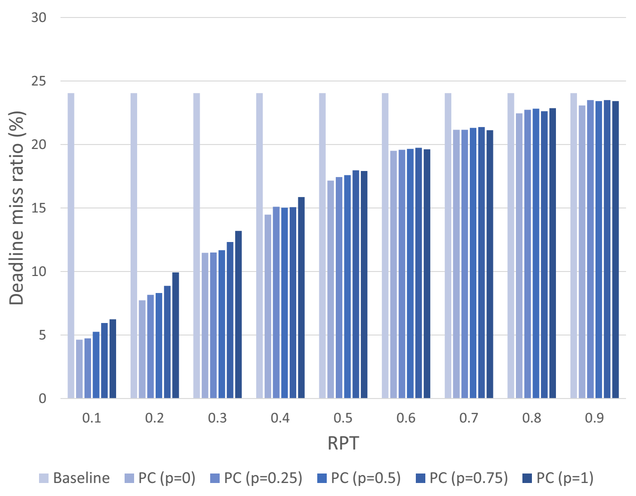

Fig. 4 Deadline miss ratio vs. result precision threshold (*RPT*), for baseline and partial computations (PC) scheduling techniques. PC was examined under input error propagation probability $p = \{0, 0.25, 0.5, 0.75, 1\}$

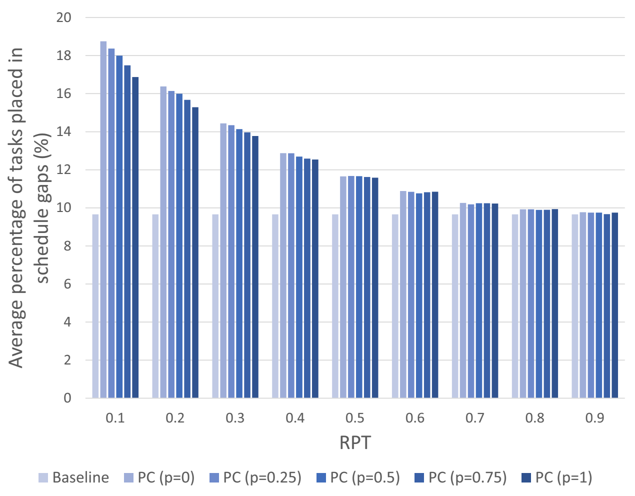

Fig. 5 Average percentage of tasks placed in schedule gaps vs. result precision threshold (*RPT*), for baseline and partial computations (PC) scheduling techniques. PC was examined under input error propagation probability $p = \{0, 0.25, 0.5, 0.75, 1\}$

Table 5 Deadline miss ratio % decrease (PC vs. Baseline)

RPT	$p = 0$	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 1$	Avg
0.1	80.74	80.29	78.15	75.28	74.07	77.71
0.2	67.83	66.06	65.45	63.11	58.71	64.23
0.3	52.31	52.21	51.42	48.76	45.10	49.96
0.4	39.78	37.22	37.50	37.34	34.03	37.18
0.5	28.67	27.48	26.87	25.29	25.51	26.76
0.6	18.90	18.57	18.26	17.92	18.39	18.41
0.7	12.01	12.00	11.39	11.13	12.15	11.74
0.8	6.63	5.47	5.11	5.94	4.94	5.62
0.9	4.04	2.30	2.62	2.29	2.62	2.77
				Overall Avg:		32.71

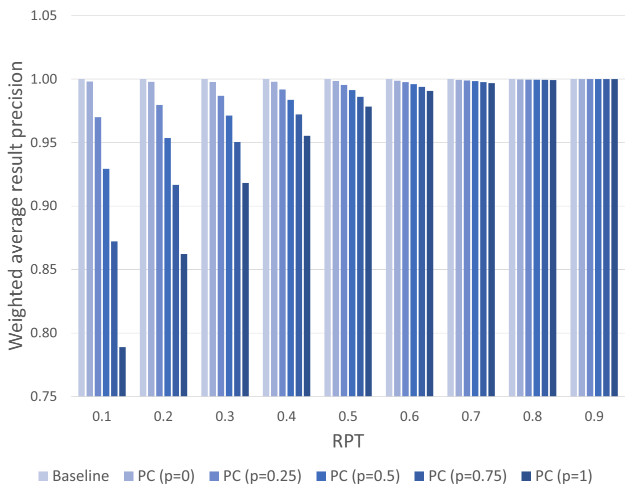

Fig. 6 Weighted average result precision vs. result precision threshold (*RPT*), for baseline and partial computations (PC) scheduling techniques. PC was examined under input error propagation probability $p = \{0, 0.25, 0.5, 0.75, 1\}$

less intermediate tasks could be placed in schedule gaps, as shown in Figs. 7 and 5, respectively. Therefore, it was less likely that the remaining unprocessed tasks of a job upon its deadline would be only exit tasks. Even in those cases where the remaining unprocessed tasks were all exit tasks, it was less likely to meet the higher result quality criteria imposed by the higher values of *RPT*. Consequently, more jobs could not be partially completed and thus missed their deadline.

Figure 10 shows the change in the weighted average input error propagation index for the proposed approach. As expected, for higher input error propagation probabilities, the input error propagation index increased, as their was a greater chance for the input error of a task to be propagated to its successor tasks. On the other hand, for higher result precision thresholds the input error propagation index decreased, because fewer tasks could be placed in schedule gaps, as shown in Fig. 5.

The average percentage of exit tasks with imprecise results in the case of the proposed policy is shown in Fig. 11. In this case, the same explanation applies as in the case of Fig. 9. Figure 12, which shows the average percentage of exit tasks with imprecise results containing propagated input error, follows a similar pattern to Fig. 11 with respect to the result precision threshold. However, in terms of the input error propagation probability, it shows that the higher the value of p was, more exit tasks conveyed in their results propagated input error from their predecessor tasks. This can be justified by the fact that the input

Table 6 Weighted average result precision % decrease (PC vs. Baseline)

RPT	$p = 0$	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 1$	Avg
0.1	0.19	3.01	7.06	12.79	21.11	8.83
0.2	0.22	2.05	4.66	8.33	13.78	5.81
0.3	0.24	1.32	2.87	4.96	8.19	3.52
0.4	0.21	0.83	1.64	2.78	4.46	1.99
0.5	0.16	0.47	0.88	1.40	2.17	1.02
0.6	0.13	0.25	0.41	0.62	0.94	0.47
0.7	0.07	0.12	0.17	0.25	0.33	0.19
0.8	0.03	0.05	0.06	0.07	0.09	0.06
0.9	0.01	0.01	0.01	0.01	0.01	0.01
				Overall Avg:		2.43

error propagation index was higher in those cases, as shown in Fig. 10.

6 Conclusions and future directions

In this paper, we investigated the orchestration of multiple real-time IoT workflows in a heterogeneous fog computing environment, using partial computations. We proposed a dynamic scheduling algorithm, PC, which leveraged partial computations in order to utilize schedule gaps, by trading off result precision for timeliness. The proposed heuristic took into account the effects of end-to-end error propagation among the component tasks of the workflow jobs. PC was compared to a baseline policy, which did not utilize partial computations, under various result precision thresholds and input error propagation probabilities.

The simulation results demonstrated that the proposed heuristic outperformed the baseline policy in terms of the deadline miss ratio, for a relatively insignificant loss in result quality. Specifically, PC provided an average deadline miss ratio decrease of 32.71%, compared to the baseline policy, while the average decrease in the precision of the results was only 2.43%. Furthermore, the simulation results showcased that the proposed approach was resilient to the effects of input error propagation across the tasks of the workflows.

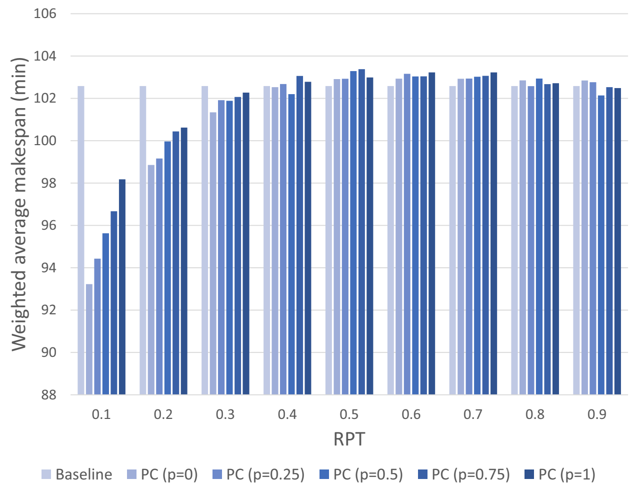

Fig. 7 Weighted average makespan vs. result precision threshold (*RPT*), for baseline and partial computations (PC) scheduling techniques. PC was examined under input error propagation probability $p = \{0, 0.25, 0.5, 0.75, 1\}$

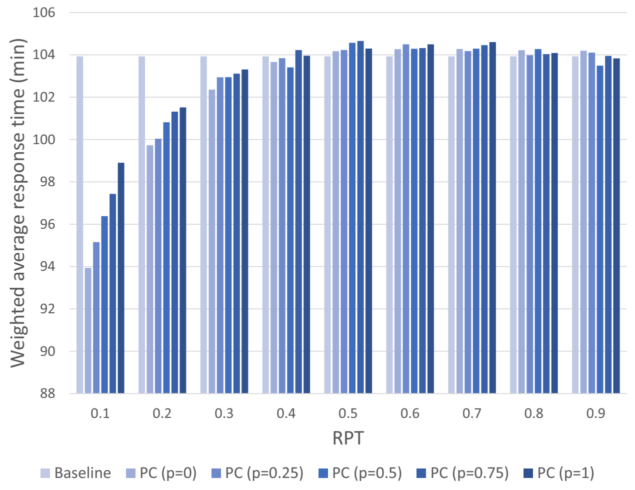

Fig. 8 Weighted average response time vs. result precision threshold (*RPT*), for baseline and partial computations (PC) scheduling techniques. PC was examined under input error propagation probability $p = \{0, 0.25, 0.5, 0.75, 1\}$

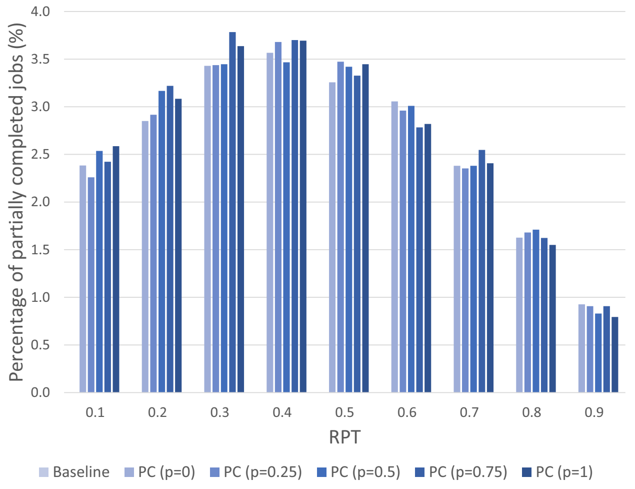

Fig. 9 Percentage of partially completed jobs vs. result precision threshold (*RPT*), for baseline and partial computations (PC) scheduling techniques. PC was examined under input error propagation probability $p = \{0, 0.25, 0.5, 0.75, 1\}$

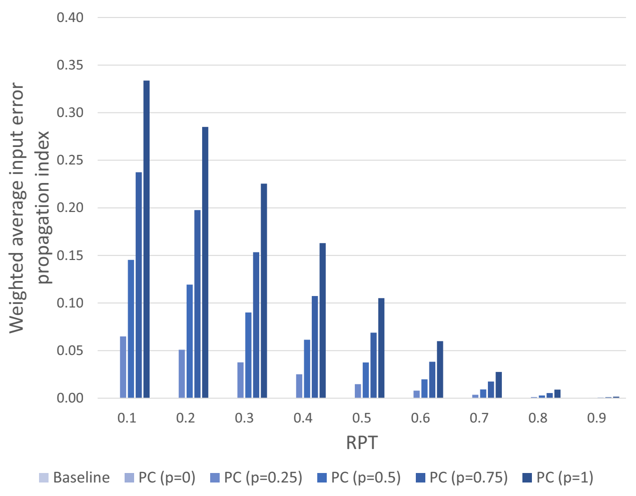

Fig. 10 Weighted average input error propagation index vs. result precision threshold (*RPT*), for baseline and partial computations (PC) scheduling techniques. PC was examined under input error propagation probability $p = \{0, 0.25, 0.5, 0.75, 1\}$

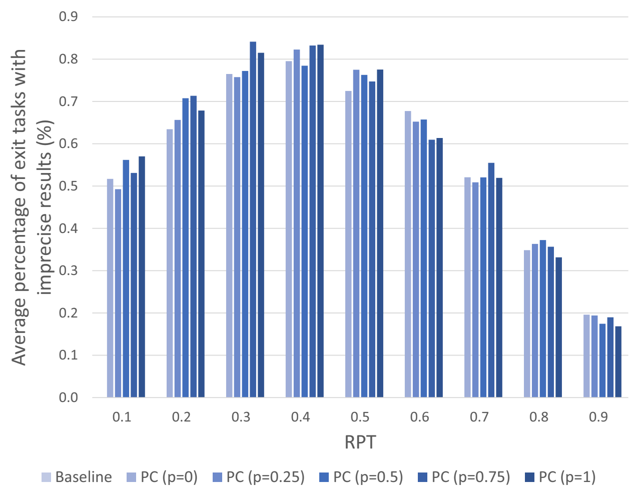

Fig. 11 Average percentage of exit tasks with imprecise results vs. result precision threshold (*RPT*), for baseline and partial computations (PC) scheduling techniques. PC was examined under input error propagation probability $p = \{0, 0.25, 0.5, 0.75, 1\}$

Our future work plans include the incorporation of the proposed scheduling algorithm into a three-tier environment, where an additional cloud layer is present. This will enable us to combine our approach with cloud bursting, i.e., the utilization of supplementary cloud resources for the scheduling of IoT workflows in cases of workload spikes. Moreover, we will consider the implementation of the proposed approach in a real-world environment, utilizing profiling and statistical techniques for the calculation of the computational and communication characteristics of the workload. A real testbed will also allow us to investigate the overhead introduced by the proposed orchestration mechanism.

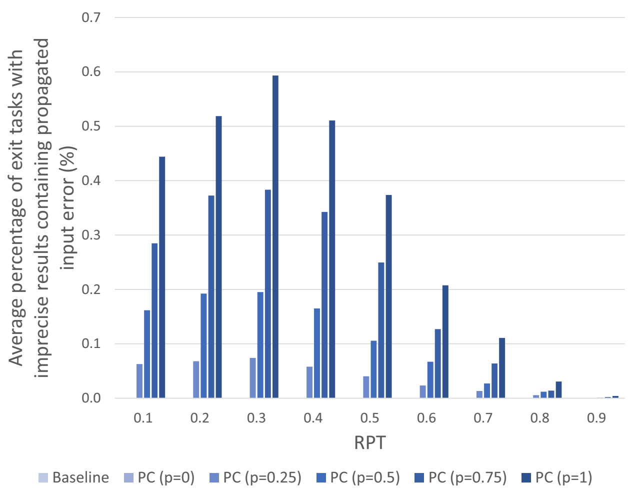

Fig. 12 Average percentage of exit tasks with imprecise results containing propagated input error vs. result precision threshold (*RPT*), for baseline and partial computations (PC) scheduling techniques. PC was examined under input error propagation probability $p = \{0, 0.25, 0.5, 0.75, 1\}$

References

1. Aburukba, R.O., AliKarrar, M., Landolsi, T., El-Fakih, K.: Scheduling Internet of Things requests to minimize latency in hybrid fog-cloud computing. Future Gen. Comput. Syst. **111**, 539–551 (2020). https://doi.org/10.1016/j.future.2019.09.039
2. Ahmed, O.H., Lu, J., Ahmed, A.M., Rahmani, A.M., Hosseinzadeh, M., Masdari, M.: Scheduling of scientific workflows in multi-fog environments using Markov models and a hybrid salp swarm algorithm. IEEE Access **8**, 189404–189422 (2020). https://doi.org/10.1109/ACCESS.2020.3031472
3. Al-Bzoor, M., Al-assem, E., Alawneh, L., Jararweh, Y.: Autonomous underwater vehicles support for enhanced performance in the internet of underwater things. Trans. Emerg. Telecommun. Technol. **32**(3), e4225 (2021). https://doi.org/10.1002/ett.4225
4. Alizadeh, M.R., Khajehvand, V., Rahmani, A.M., Akbari, E.: Task scheduling approaches in fog computing: a systematic review. Int. J. Commun. Syst. **33**(16), e4583 (2020). https://doi.org/10.1002/dac.4583
5. Buttazzo, G.C.: Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, 3rd edn. Springer, Berlin (2011). https://doi.org/10.1007/978-1-4614-0676-1
6. Cao, K., Zhou, J., Xu, G., Wei, T., Hu, S.: Exploring renewable-adaptive computation offloading for hierarchical QoS optimization in fog computing. IEEE Trans. Comput. Aid. Des. Integr. Circuits Syst. **39**(10), 2095–2108 (2020). https://doi.org/10.1109/TCAD.2019.2957374
7. Chen, Y.: Service-Oriented Computing and System Integration: Software, IoT, Big Data, and AI as Services, 7th edn. Kendall Hunt Publishing, Dubuque (2020)
8. Chen, Y., Hu, H.: Internet of Intelligent Things and Robot as a Service. Simul. Model. Pract. Theor. **34**, 159–171 (2013). https://doi.org/10.1016/j.simpat.2012.03.006
9. Choudhari, T., Moh, M., Moh, T.S.: Prioritized task scheduling in fog computing. In: Proceedings of the 2018 Annual ACM

Southeast Conference (ACMSE'18), pp. 22:1–22:8 (2018). https://doi.org/10.1145/3190645.3190699

10. Cisco: Fog computing and the Internet of Things: extend the cloud to where the things are. Tech. Rep. C11-734435-00, Cisco Systems, Inc. (2015)
11. De Maio, V., Kimovski, D.: Multi-objective scheduling of extreme data scientific workflows in fog. Future Gen. Comput. Syst. **106**, 171–184 (2020). https://doi.org/10.1016/j.future.2019.12.054
12. De Souza Toniolli, J.L., Jaumard, B.: Resource allocation for multiple workflows in cloud-fog computing systems. In: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC'19 Companion), pp. 77–84 (2019). https://doi.org/10.1145/3368235.3368846
13. Ding, R., Li, X., Liu, X., Xu, J.: A cost-effective time-constrained multi-workflow scheduling strategy in fog computing. In: Proceedings of the 16th International Conference on Service-Oriented Computing (ICSOC'18), pp. 194–207 (2018). https://doi.org/10.1007/978-3-030-17642-6_17
14. Drozdowski, M.: Scheduling for Parallel Processing, 1st edn. Springer, Berlin (2009). https://doi.org/10.1007/978-1-84882-310-5
15. Esmaili, A., Nazemi, M., Pedram, M.: Energy-aware scheduling of task graphs with imprecise computations and end-to-end deadlines. ACM Trans. Des. Autom. Electron. Syst. **25**(1), 11:1–11:21 (2019). https://doi.org/10.1145/3365999
16. Feng, W.C., Liu, J.W.S.: Algorithms for scheduling real-time tasks with input error and end-to-end deadlines. IEEE Trans. Softw. Eng. **23**(2), 93–106 (1997). https://doi.org/10.1109/32.585499
17. Gazori, P., Rahbari, D., Nickray, M.: Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach. Future Gen. Comput. Syst. **110**, 1098–1115 (2020). https://doi.org/10.1016/j.future.2019.09.060
18. Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K., Buyya, R.: iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments. Softw. Pract. Exp. **47**(9), 1275–1296 (2017). https://doi.org/10.1002/spe.2509
19. Iorga, M., Feldman, L., Barton, R., Martin, M.J., Goren, N., Mahmoudi, C.: Fog computing conceptual model. Tech. Rep. 500-325, National Institute of Standards and Technology, U.S. Department of Commerce (2018). https://doi.org/10.6028/NIST.SP.500-325
20. Kabirzadeh, S., Rahbari, D., Nickray, M.: A hyper heuristic algorithm for scheduling of fog networks. In: Proceedings of the 21st Conference of Open Innovations Association (FRUCT'17), pp. 148–155 (2017). https://doi.org/10.23919/FRUCT.2017.8250177
21. Kołodziej, J.: Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems, 1st edn. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-28971-2
22. Lin, K.J., Natarajan, S., Liu, J.W.S.: Imprecise results: utilizing partial computations in real-time systems. In: Proceedings of the 8th IEEE Real-Time Systems Symposium (RTSS'87), pp. 210–217 (1987)
23. Liu, X., Fan, L., Xu, J., Li, X., Gong, L., Grundy, J., Yang, Y.: FogWorkflowSim: An automated simulation toolkit for workflow performance evaluation in fog computing. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19), pp. 1114–1117 (2019). https://doi.org/10.1109/ASE.2019.00115
24. Matrouk, K., Alatoun, K.: Scheduling algorithms in fog computing: A survey. Int. J. Netw. Distr. Comp. **9**(1), 59–74 (2021). https://doi.org/10.2991/ijndc.k.210111.001
25. Mo, L., Kritikakou, A.: Mapping imprecise computation tasks on cyber-physical systems. Peer-to-Peer Netw. Appl. **12**(6), 1726–1740 (2019). https://doi.org/10.1007/s12083-019-00749-9
26. Mo, L., Kritikakou, A., Sentieys, O., Cao, X.: Real-time imprecise computation tasks mapping for DVFS-enabled networked systems. IEEE Internet Things J. **8**(10), 8246–8258 (2021). https://doi.org/10.1109/JIOT.2020.3044910
27. Mora Mora, H., Gil, D., Colom López, J.F., Signes Pont, M.T.: Flexible framework for real-time embedded systems based on mobile cloud computing paradigm. Mob. Inf. Syst. **2015**, 652462:1–652462:14 (2015). https://doi.org/10.1155/2015/652462
28. Naha, R.K., Garg, S., Chan, A., Battula, S.K.: Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment. Future. Gen. Comput. Syst. **104**, 131–141 (2020). https://doi.org/10.1016/j.future.2019.10.018
29. OpenFog: OpenFog Architecture Overview. Tech. Rep. OPFWP001.0216, OpenFog Consortium Architecture Working Group (2016)
30. Park, M., Han, S., Kim, H., Cho, S., Cho, Y.: Comparison of tie-breaking policies for real-time scheduling on multiprocessor. In: Proceedings of the 2004 International Conference on Embedded and Ubiquitous Computing (EUC'04), pp. 174–182 (2004). https://doi.org/10.1007/978-3-540-30121-9_17
31. Pham, X.Q., Huh, E.N.: Towards task scheduling in a cloud-fog computing system. In: Proceedings of the 18th Asia-Pacific Network Operations and Management Symposium (APNOMS'16), pp. 1–4 (2016). https://doi.org/10.1109/APNOMS.2016.7737240
32. Pham, X.Q., Man, N.D., Tri, N.D.T., Thai, N.Q., Huh, E.N.: A cost- and performance-effective approach for task scheduling based on collaboration between cloud and fog computing. Int. J. Distrib. Sens. Netw. **13**(11), 1–16 (2017). https://doi.org/10.1177/1550147717742073
33. Puliafito, C., Mingozzi, E., Longo, F., Puliafito, A., Rana, O.: Fog computing for the Internet of Things: A survey. ACM Trans. Internet Technol. **19**(2), 18:1–18:41 (2019). https://doi.org/10.1145/3301443
34. Ravindran, R., Krishna, C.M., Koren, I., Koren, Z.: Scheduling imprecise task graphs for real-time applications. Int. J. Embed. Syst. **6**(1), 73–85 (2014). https://doi.org/10.1504/IJES.2014.060919
35. Shioura, A., Shakhlevich, N.V., Strusevich, V.A.: Preemptive models of scheduling with controllable processing times and of scheduling with imprecise computation: A review of solution approaches. Eur. J. Oper. Res. **266**(3), 795–818 (2018). https://doi.org/10.1016/j.ejor.2017.08.034
36. Stavrinides, G.L., Karatza, H.D.: Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations. J. Syst. Softw. **83**(6), 1004–1014 (2010). https://doi.org/10.1016/j.jss.2009.12.025
37. Stavrinides, G.L., Karatza, H.D.: The impact of input error on the scheduling of task graphs with imprecise computations in heterogeneous distributed real-time systems. In: Proceedings of the 18th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'11), pp. 273–287 (2011). https://doi.org/10.1007/978-3-642-21713-5_20
38. Stavrinides, G.L., Karatza, H.D.: Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. Simul. Model. Pract. Theor. **19**(1), 540–552 (2011). https://doi.org/10.1016/j.simpat.2010.08.010
39. Stavrinides, G.L., Karatza, H.D.: Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes.

Future Gen. Comput. Syst. **28**(7), 977–988 (2012). https://doi.org/10.1016/j.future.2012.03.002

40. Stavrinides, G.L., Karatza, H.D.: A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds. In: Proceedings of the 3rd International Conference on Future Internet of Things and Cloud (FiCloud'15), pp. 231–239 (2015). https://doi.org/10.1109/FiCloud.2015.93
41. Stavrinides, G.L., Karatza, H.D.: Energy-aware scheduling of real-time workflow applications in clouds utilizing DVFS and approximate computations. In: Proceedings of the IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud'18), pp. 33–40 (2018). https://doi.org/10.1109/FiCloud.2018.00013
42. Stavrinides, G.L., Karatza, H.D.: Cost-effective utilization of complementary cloud resources for the scheduling of real-time workflow applications in a fog environment. In: Proceedings of the 7th International Conference on Future Internet of Things and Cloud (FiCloud'19), pp. 1–8 (2019). https://doi.org/10.1109/FiCloud.2019.00009
43. Stavrinides, G.L., Karatza, H.D.: An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations. Future Gen. Comput. Syst. **96**, 216–226 (2019). https://doi.org/10.1016/j.future.2019.02.019
44. Stavrinides, G.L., Karatza, H.D.: A hybrid approach to scheduling real-time IoT workflows in fog and cloud environments. Multimed. Tools Appl. **78**(17), 24639–24655 (2019). https://doi.org/10.1007/s11042-018-7051-9
45. Stavrinides, G.L., Karatza, H.D.: Cost-aware cloud bursting in a fog-cloud environment with real-time workflow applications. Concurr. Comput. Pract. Exp. (2020). https://doi.org/10.1002/cpe.5850
46. Stavrinides, G.L., Karatza, H.D.: Orchestration of real-time workflows with varying input data locality in a heterogeneous fog environment. In: Proceedings of the Fifth International Conference on Fog and Mobile Edge Computing (FMEC'20), pp. 202–209 (2020). https://doi.org/10.1109/FMEC49853.2020.9144824
47. Wainer, G., Moallemi, M.: Designing real-time systems using imprecise discrete-event system specifications. Softw. Pract. Exp. **50**(8), 1327–1344 (2020). https://doi.org/10.1002/spe.2831
48. Wu, H.Y., Lee, C.R.: Energy efficient scheduling for heterogeneous fog computing architectures. In: Proceedings of the 42nd IEEE Annual Computer Software and Applications Conference (COMPSAC'18), pp. 555–560 (2018). https://doi.org/10.1109/COMPSAC.2018.00085
49. Xu, J., Hao, Z., Zhang, R., Sun, X.: A method based on the combination of laxity and ant colony system for cloud-fog task scheduling. IEEE Access **7**, 116218–116226 (2019). https://doi.org/10.1109/ACCESS.2019.2936116
50. Yao, S., Hao, Y., Zhao, Y., Shao, H., Liu, D., Liu, S., Wang, T., Li, J., Abdelzaher, T.: Scheduling real-time deep learning services as imprecise computations. In: Proceedings of the IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'20), pp. 1–10 (2020). https://doi.org/10.1109/RTCSA50079.2020.9203676
51. Yu, K.P., Tan, L., Aloqaily, M., Yang, H., Jararweh, Y.: Blockchain-enhanced data sharing with traceable and direct revocation in IIoT. IEEE Trans. Ind. Inf. (2021). https://doi.org/10.1109/TII.2021.3049141

Georgios L. Stavrinides received the B.Sc. degree in Informatics from Aristotle University of Thessaloniki, Greece in 2006 and the M.Sc. degree in Advanced Computing from Imperial College London, UK in 2007. He received the Ph.D. degree in Computer Science from Aristotle University of Thessaloniki, Greece in 2014. He is currently a postdoctoral researcher in the Department of Informatics of the Aristotle University of Thessaloniki, Greece, under the supervision of Professor Emeritus Helen D. Karatza. His research interests include: scheduling algorithms, real-time distributed systems, fog and cloud computing, modeling, simulation and performance evaluation of large-scale distributed systems.

Helen D. Karatza is a Professor Emeritus in the Department of Informatics at the Aristotle University of Thessaloniki, Greece. Her research interests include: computer systems modeling and simulation, performance evaluation, fog and cloud computing, real-time distributed systems, resource allocation and scheduling. She is the Editor-in-Chief of the Elsevier journal "Simulation Modelling Practice and Theory". She has been Guest Editor of special issues in multiple international journals.