De Maio, V.; Kimovski, D. Multi-objective scheduling of extreme data scientific workflows in Fog, In *Future generation computer systems, 106, May 2020, Pp. 171-184*
DOI: https://doi.org/10.1016/j.future.2019.12.054

Multi-Objective Scheduling of Extreme Data Scientific Workflows in Fog

Vincenzo De Maio

Vienna University of Technology, Vienna, Austria
vincenzo@ec.tuwien.ac.at

Dragi Kimovski

Klagefurt University, Klagenfurt, Austria
dragi.kimovski@aau.at

Abstract

The concept of "extreme data" is a recent re-incarnation of the "big data" problem, which is distinguished by the massive amounts of information that must be analyzed with strict time requirements. In the past decade, the Cloud data centers have been envisioned as the essential computing architectures for enabling extreme data workflows. However, the Cloud data centers are often geographically distributed. Such geographical distribution increases offloading latency, making it unsuitable for processing of workflows with strict latency requirements, as the data transfer times could be very high. Fog computing emerged as a promising solution to this issue, as it allows partial workflow processing in lower-network layers. Performing data processing on the Fog significantly reduces data transfer latency, allowing to meet the workflows' strict latency requirements. However, the Fog layer is highly heterogeneous and loosely connected, which affects reliability and response time of task offloading. In this work, we investigate the potential of Fog for scheduling of extreme data workflows with strict response time requirements. Moreover, we propose a novel Pareto-based approach for task offloading in Fog, called Multi-objective Workflow Offloading (MOWO). MOWO considers three optimization objectives, namely response time, reliability, and financial cost. We evaluate MOWO workflow scheduler on a set of real-world biomedical, meteorological and astronomy workflows representing examples of extreme data application with strict latency requirements.

Keywords Extreme scale workflows, Fog computing, Multi-objective optimization

I. Introduction

Data-intensive scientific workflows represent an important class of applications. These applications are present wherever data sources and computations are distributed, in domains such as bioinformatics, astronomy and civil engineering. They are characterized by a wide data distribution due to the large volume of sources (e.g. sensors, experimental processes), which has to be collected and analyzed within given time constraints to obtain meaningful information. The processes of data collection and analysis require a big amount of computational and network resources, usually not available in common scientific settings [1]. For this reason, execution of scientific workflows often employs geographically distributed Cloud resources.

The use of Cloud resources allows scientists in different fields to run computationally intensive scientific workflows exploiting typical Cloud features (e.g. elasticity, virtualization). However, in recent years, we have seen the rise of extreme data scientific application [2] (e.g. eHealth [3], astronomy [4] and other scientific application domains). The main characteristics of these applications are (1) their distributed nature, (2) the huge amount of data (in the order of petabytes) they generate and (3) strict latency requirements. Due to the geographical distance between the data sources and the Cloud, running such workflows on the Cloud will negatively affect latency and cause violation of workflow requirements [5].

Fog Computing has been proposed as a solution to these issues [6]. From an architectural point of view it can be seen as an extension of the Cloud that relies on computational resources deployed in close proximity to the user (Fog nodes). The use of Fog nodes allows to differentiate scheduling for different types of workload components. For example, it allows scheduling of interactive tasks closer to the user, or data-intensive tasks closer to the data sources, to decrease response time and improve user experience.

However, Fog architectures are highly heterogeneous and loosely connected, therefore posing multiple challenges in terms of workflow scheduling and tasks offloading. Moreover, Fog nodes usually host less computational resources, thus preventing offloading of many tasks on the Fog layer. Currently, only a few state-of-the-art research works explored this problem [7, 8, 9]. However, they mostly focus on optimizing only a single constrained objective, omitting important parameters such as cost and reliability.

In this work, we investigate the potential of Fog computing for scheduling of scientific workflows with low latency requirements. First, we propose a model for workflow scheduling considering the use of Cloud and Fog resources, in terms of virtual infrastructure, i.e. virtual machines and containers. Then, we define the problem of workflow scheduling as a multi-objective optimization problem, considering as objectives the global response time, reliability and cost of the scheduling. Based on this definition, we propose MOWO (Multi-Objective Workflow Optimization), an algorithm based on NSGA-II metaheuristic, to solve this problem. Afterwards, we analyze the suitability of the Cloud alone for performing near-real-time workflows and compare it with a hybrid Cloud/Fog environment. Finally, we evaluate the algorithm in comparison to other state-of-the-art workflow scheduling algorithms.

This work focuses on scientific workflows, therefore we will consider real-world scientific and biomedical workflow structures with randomly generated characteristics. Our evaluation employs Monte-Carlo simulations, which have been proven to be very effective in simulating Fog infrastructures [10].

The paper is organized as follows: first, we define our theoretical model in Section II. Based on the theoretical model, we design the multi-objective algorithm in Section III. Then, we define our experimental setup and the selected scientific workflows in Section IV and comment our results in Section V. We discuss related work in Section VI and conclude our paper in Section VII.

II. Model

II.1 Extreme Data Workflow Model

Our definition of extreme data workflow expands from the definition of scientific workflow given by [11, 12]. Formally, a workflow $\mathcal{W}$ is defined as a set of interdependent tasks represented by a DAG (Directed Acyclic Graph), whose nodes are the tasks and edges model dependencies between them. We extend this definition of scientific workflow by considering also latency requirements for data transfer between tasks, to model near-real-time requirements of applications defined as in [13]. Formally,

Definition 1. *$\mathcal{W} \stackrel{\text{def}}{=} \langle \mathcal{T}_\mathcal{W}, \mathcal{L}_\mathcal{W}, \vec{d}_t(\bullet), \vec{d}_l(\bullet) \rangle$, where $\mathcal{T}_\mathcal{W}$ is the set of tasks in workload $\mathcal{W}$, $\mathcal{L}_\mathcal{W}$ is the set of edges connecting the tasks in $\mathcal{T}_\mathcal{W}$ ($\mathcal{L}_\mathcal{W} \subseteq \mathcal{T}_\mathcal{W} \times \mathcal{T}_\mathcal{W}$), $\vec{d}_t(\bullet)$ the vector of requirements of each task in $\mathcal{T}_\mathcal{W}$ and $\vec{d}_l(\bullet)$ is the demand vector for each edge in $\mathcal{L}_\mathcal{W}$.*

Let $\mathcal{T}_\mathcal{W}$ be the set of workflow tasks. The demand vector $\vec{d}_t(t_i)$ for a task $t_i \in \mathcal{T}_\mathcal{W}$, is defined in terms of CPU demand and input/output data:

$$\vec{d}_t(t_i) \stackrel{\text{def}}{=} \langle \texttt{SIZE}(t_i), \texttt{CPU}(t_i), \texttt{DATA}_{in}(t_i), \texttt{DATA}_{out}(t_i) \rangle, \tag{1}$$

respectively, the size of the task t_i in millions of instruction (MI), the number of CPUs required by t_i and the size of input/output data. We assume that the size of input data is equal to the sum of the output data of the task predecessors, namely

$$\texttt{DATA}_{in}(t_i) = \sum_{t_j \in \pi(t_i)} \texttt{DATA}_{out}(t_j). \tag{2}$$

Concerning edges, for each edge $(t_i, t_j) \in \mathcal{L}_\mathcal{W}$ we define the demand vector $\vec{d}_l((t_i, t_j))$ as

$$\vec{d}_l((t_i, t_j)) \stackrel{\text{def}}{=} \texttt{lat}((t_i, t_j)), \tag{3}$$

where $\texttt{lat}((t_i, t_j))$ represents the maximum latency that user requires to transfer data from t_i and t_j. This demand vector is used to model the near-real time requirements of extreme data applications. To have a valid scheduling, we have to ensure that there is a connection between t_i and t_j, which ensures a latency lower or equal to $\texttt{lat}((t_i, t_j))$.

II.2 Infrastructure model

In this work, we target Cloud/Fog infrastructures. In this type of infrastructures, we have heterogeneous computational nodes: (1) Cloud data centers, with high computational capabilities, but far from the source of data and geographically distributed, and (2) Fog nodes, with lower computational capabilities [6] (in comparison with Cloud data centers) but closer to the source of data and with lower latencies [14].

Definition 2. *We define a Cloud/Fog infrastructure $\mathcal{I}$ as a directed graph such that $\mathcal{I} = \{\mathcal{N}_\mathcal{I}, \mathcal{L}_\mathcal{I}\}$, where $\mathcal{N}_\mathcal{I}$ is the set of computational nodes and $\mathcal{L}_\mathcal{I}$ the set of network connections between the nodes, namely $\mathcal{L}_\mathcal{I} \subseteq \mathcal{N}_\mathcal{I} \times \mathcal{N}_\mathcal{I}$.*

Next, we define computational nodes and network connections.

II.2.1 Computational nodes

$\mathcal{N}_\mathcal{I}$ is defined as $\{\mathcal{C}_\mathcal{I} \cup \mathcal{F}_\mathcal{I} \cup \mathcal{E}_\mathcal{I}\}$, respectively the set of Cloud data centers, the set of Fog nodes and the set of end devices. Each computational node can host a limited number of resources, in terms of number of CPUs, the million of instructions per second that the node can execute and its storage capabilities, defined by its capacity vector $\vec{c}$. $\vec{c}$ varies according to the type of node. Concerning Cloud nodes, we define $\vec{c}_c$ as

$$\vec{c}_c(c_i \in \mathcal{C}_\mathcal{I}) = \langle \texttt{CPU}, \texttt{MIPS}, \texttt{STORAGE}, p(c_i) \rangle \tag{4}$$

where $\texttt{CPU}(c_i)$ is the number of CPUs available on the Cloud data center, $\texttt{MIPS}(c_i)$ is the million of instruction per second that each CPU in c_i can execute, $\texttt{STORAGE}(c_i)$ is the amount of storage available on c_i, $p(c_i)$ is the pricing function for using Cloud resources on node c_i (defined in Section II.3.1). Concerning Fog nodes, they are computational nodes whose characteristics are: (1) a limited amount of resource compared to the Cloud nodes, and (2) geographical proximity to the user, i.e. they are located either on-premise, or at locations near to end devices [14]. Therefore, we define a capacity vector of a Fog node $f_i \in \mathcal{F}_\mathcal{I}$ as

$$\vec{c}_f(f_i \in \mathcal{F}_\mathcal{I}) \stackrel{\text{def}}{=} \langle \texttt{CPU}, \texttt{MIPS}, \texttt{STORAGE}, p_f(f_i) \rangle \tag{5}$$

where $\texttt{CPU}(f_i)$ is the number of CPUs available on the Fog node, $\texttt{MIPS}(f_i)$ represents the millions of instruction per second that each CPU in f_i can execute, $\texttt{STORAGE}(f_i)$ is the amount of storage available on the Fog node and $p_f(f_i)$ the price penalty for execution on Fog node f_i (see Section II.3.1). Finally, we assume that each workflow is submitted from an end device $e \in \mathcal{E}_\mathcal{I}$. We define the end devices $e_i \in \mathcal{E}_\mathcal{I}$ as the nodes where each workflow is submitted to the infrastructure and where the final results of the workflow execution are collected. These nodes model different type of user devices used to submit workflows and to retrieve results of their execution (e.g. mobile devices, laptops or desktop PCs). For each end device we define the capacity vector as

$$\vec{c}_e(e_i) \in \mathcal{E}_\mathcal{I}) = \langle \texttt{CPU}, \texttt{MIPS}, \texttt{STORAGE} \rangle. \tag{6}$$

For each network connection $l_{i,j} = (n_i, n_j) \in \mathcal{L}_\mathcal{I}$ we also define a Quality of Service vector $QoS(l_{i,j})$ as follows:

$$QoS(l_{i,j}) = \langle \texttt{lat}, \texttt{bw} \rangle, \tag{7}$$

respectively the latency and the bandwidth available between nodes n_i and n_j. If there is no connection between n_i and n_j, $\texttt{lat}(l_{i,j}) = \infty$ and $\texttt{bw}(l_{i,j}) = 0$. Conversely, we set $\texttt{lat}(l_{i,i}) = 0$ and $\texttt{bw}(l_{i,i}) = \infty$. Formally,

Definition 3. *$l_{i,j} \stackrel{\text{def}}{=} (n_i, n_j) : n_i, n_j \in \mathcal{N}_\mathcal{I}$. Also, $\exists l_{i,j} \iff \texttt{lat}(l_{i,j}) \geq 0, \texttt{bw}(l_{i,j}) \in \mathbb{R}^+$.*

II.3 Problem definition

Scheduling $\mathcal{S}$ of a workload $\mathcal{W}$ over an infrastructure $\mathcal{I}$ is a mapping of the tasks in $\mathcal{T}_\mathcal{W}$ to the nodes in $\mathcal{N}_\mathcal{I}$ and of the edges in $\mathcal{L}_\mathcal{W}$ to the physical links in $\mathcal{L}_\mathcal{I}$. A task can be executed either on an end device or offloaded to a Cloud/Fog node. Since links in $\mathcal{L}_\mathcal{W}$ impose an order on tasks' execution, scheduling of $\mathcal{W}$ over $\mathcal{I}$ is performed in different time steps. In each time step we execute only tasks $t_i \in \mathcal{T}_\mathcal{W}$ such that $\delta_{in}(t_i, \tau) = \emptyset$. Such tasks are defined as *ready*. We define as $\mathcal{T}_\mathcal{W}(\tau)$ the set of tasks in $\mathcal{T}_\mathcal{W}$ that are ready at time step τ. We define the *partial* scheduling of tasks $\mathcal{T}_\mathcal{W}(\tau)$ as a set $\mathcal{S}_\tau$ of pairs (t_i, n_j), with $\mathcal{S}_\tau \subseteq \mathcal{T}_\mathcal{W}(\tau) \times \mathcal{N}_\mathcal{I}$, such that

$$(t_i, n_j) \in \mathcal{S}_{\ll} \iff t_i \text{ is allocated to node } n_j. \tag{8}$$

More tasks can be assigned to a computational node, as soon as node capacity constraints are respected. We define $\mathcal{S}_\tau(n_i)$ as the set of tasks mapped to node n_i at instant τ, namely

$$t_i \in \mathcal{S}_\tau(n_j) \iff \exists j : (t_i, n_j) \in \mathcal{S}_\tau. \tag{9}$$

For simplicity, we define $\mathcal{S}(t_i)$ as the node where t_i is scheduled in schedule $\mathcal{S}$. We define a *valid* scheduling of a workflow $\mathcal{W}$ on infrastructure $\mathcal{I}$, $\mathcal{S}_\tau(\mathcal{W}, \mathcal{I})$. A partial deployment is valid only if (1) all the tasks are deployed only once on the infrastructure and (2) all deployments of tasks on nodes satisfy the capacity constraints of the nodes, namely:

Definition 4. *A scheduling $\mathcal{S}_\tau(\mathcal{W}, \mathcal{I})$ of a workflow $\mathcal{W}$ on infrastructure $\mathcal{I}$ is* valid $\iff$

1. $\bigcup_{n_j \in \mathcal{N}_\mathcal{I}} \mathcal{S}(n_j) = \mathcal{T}_\mathcal{W}(\tau)$;
2. $(t_i, n_j) \in \mathcal{S}_\tau(\mathcal{W}, \mathcal{I}), t_i \in \mathcal{T}_\mathcal{W} \implies n_j \in \mathcal{E}$;
3. $(t_i, n_j) \in \mathcal{S}_\tau(\mathcal{W}, \mathcal{I}) \iff$
   (a) $\sum_{t_i \in \mathcal{S}_\tau(n_j)} \vec{d}(t_i) \leq \vec{c}_\bullet(n_i)$;
   (b) $\bigcup_{t_i \in \mathcal{S}_\tau(n_j)} \delta_{in}(t_i, \tau) \subseteq \delta_{in}(n_i, \tau)$;
   (c) $\bigcup_{t_i \in \mathcal{S}_\tau(n_j)} \delta_{out}(t_i, \tau) \subseteq \delta_{out}(n_i, \tau)$;
   (d) $\forall (t_j, t_i) \in \delta_{in}(t_i, \tau)$ $\texttt{lat}(\mathcal{S}(t_j), \mathcal{S}(t_i)) \leq \vec{d}_l(t_j, t_i)$, $l_{i,j} \in \delta_{in}(n_i, \tau)$;
   (e) $\forall (t_i, t_j) \in \delta_{out}(t_i, \tau)$ $\texttt{lat}(\mathcal{S}(t_i), \mathcal{S}(t_j)) \leq \vec{d}_l((t_i, t_j))$, $l_{i,j} \in \delta_{out}(n_i, \tau)$.

A complete deployment $\mathcal{S}(\mathcal{W}, \mathcal{I})$ is the union of all partial deployments, namely

$$\mathcal{S}(\mathcal{W}, \mathcal{I}) = \bigcup_{\tau=[0, \tau_{end}]} \mathcal{S}_\tau(\mathcal{W}, \mathcal{I}), \tag{10}$$

where τ_{end} is the time when $\mathcal{T}_\mathcal{W} = \emptyset$.

II.3.1 Objectives

In this work, we want to focus on the perspective of the user running the workflow. For this reason, we do not consider objectives such as energy consumption, or profit for the provider. Following the ideas from [10, 15, 16], we focus on three main objectives: running time, reliability and user cost. Therefore, the objectives of our deployment are defined as follows:

$$\begin{cases} \min RT(\mathcal{S}(\mathcal{W}, \mathcal{I})) \\ \max RL(\mathcal{S}(\mathcal{W}, \mathcal{I})) \\ \min COST(\mathcal{S}(\mathcal{W}, \mathcal{I})) \end{cases} \tag{11}$$

subject to the constraints in Definition 4, where RT represents the total response time of the deployment, as defined in Equation 12, RL represents the reliability of the deployment, as defined in Equation 16, and $COST$ represents the cost of the deployment, as defined in Equation 17.

Response time The workload response time is defined as

$$RT(\mathcal{S}(\mathcal{W}, \mathcal{I})) = \tau_{end}(\mathcal{W}), \tag{12}$$

where τ_{end} is the time when $\mathcal{T}_\mathcal{W} = \emptyset$. For each task t_i, $rt(t_i)$ the response time of t_i depends on the maximum response time among each predecessor of t_i, plus (1) the time for running the task on node n_i, namely $rt_{cpu}(t_i, n_i)$, (2) the time for transferring input data of task t_i from node n_j to n_i, $OT_{in}(t_i, n_i, n_j)$ and (3) the time for transferring output data from predecessors to node n_i $OT_{out}(t_i, n_i, n_j)$. Due to the interdependency of tasks, a task t_i cannot be executed before all its predecessors are terminated. For this reason, the response time of each task depends on the maximum response time of its predecessors. We define then the predecessor with maximum response time as $\pi^*(t_i) = \arg\max_{t \in \delta_{in} t_i} RT(t, S(t))$. Finally, we define the response time of each task as follows:

$$RT(t, S(t)) = \begin{cases} rt_{cpu}(t, S(T)) \iff \delta_{in}(t) = \emptyset \\ RT(\pi^*(t), S(\pi^*(t))) + OT_{in}(t, \mathcal{S}(\pi^*(t)), \mathcal{S}(t)) + \\ + OT_{out}(t, \pi^*(t)), \mathcal{S}(t))) \end{cases} \tag{13}$$

We describe each term in the following text. First, we define $rt_{cpu}(t_i, n_i)$ as follows:

$$rt_{cpu}(t_i, n_i) = \frac{\texttt{SIZE}(t_i)}{\texttt{MIPS}(n_i)} \tag{14}$$

Offloading time OT_{in} depends on data transferred to $\mathcal{S}(t_i)$ and bandwidth available between end device e and $\mathcal{D}(t_i)$. Offloading t_i requires transfer of input data $\texttt{DATA}_{in}(t_i)$. Then, OT_{up} and OT_{down} are defined as

$$OT_{in,out}(n_i, t_i, n_j) = \frac{\texttt{DATA}_{in,out}}{\texttt{bw}(n_i, n_j)} + \texttt{lat}(n_i, n_j). \tag{15}$$

Reliability The reliability for executing a workflow $\mathcal{W}$ on Cloud and Fog resources $\mathcal{N}_\mathcal{I}$ is calculated by considering the reliability of the physical devices $\mathcal{R}_i$, where the set of tasks $\mathcal{T}_\mathcal{W}(\tau)$ is scheduled. We model the reliability objective as:

$$RL(\mathcal{S}(\mathcal{W}, \mathcal{I})) = \prod_{i=1}^{\mathcal{T}_\mathcal{W}(\tau)} (R_i)) \tag{16}$$

Due to the strict dependencies among the workflow's tasks, we model the reliability in a sequential manner, meaning that if a task fails due to node failure, the workflow will either not be successfully completed or a fail-safe mechanism will be activated. The reliability objective serves only as a criterion that predicts how high is the probability for the workflow execution to fail and does not include any workflow fail-safe mechanisms, such as roll-back or checkpoints. More concretely, if the reliability of the workflow is higher, than the protective mechanisms will have to be activated less frequently, consequently reducing the execution overhead.

Cost The cost for user depends on where tasks are executed, the amount of resources used and the running time, assuming different cost for each Cloud or Fog node. User cost for a deployment $\mathcal{S}(\mathcal{W},\mathcal{I})$ is defined as the sum of the execution costs of each task, as in

$$COST(\mathcal{S}(\mathcal{W},\mathcal{I})) = \sum_{t_i \in \mathcal{W}} cost(t_i). \tag{17}$$

Where $cost(t_i)$ depends on the VM instance v_i used by t_i and on where this instance is executed. For simplicity, we assume that $\vec{d_t}(t_i) = \vec{c}_c(v_i)$. If t_i is offloaded on Cloud, the user pays the price for running the VM on Cloud during the running time of the task t_i scheduled on the node $c_j \in \mathcal{C}_\mathcal{I}$, namely $p(t_i, c_j)$. Concerning Fog, at the time we write, finding a pricing strategy for Fog nodes is still an open research challenge by [17], since Cloud pricing strategies used for Cloud are not applicable on Fog [18]. Therefore, if task t_i is scheduled on Fog, the user will pay the average price for scheduling the VM on Cloud, $\bar{p}(t_i, \mathcal{C}_\mathcal{I})$, plus an additional quantity, defined by a function p_f, for scheduling tasks on Fog. Such additional cost is because of two main reasons: first, execution on Fog has a positive effect on latency, due to the reduced geographical distance. Moreover, deploying Fog nodes in proximity of the user (e.g. metropolitan areas, highways) has an additional cost for the provider. Therefore, this additional quantity $p_f(t_i, f_j)$ should maximize both provider's revenue and user's QoS satisfaction. We define then the cost for executing task t_i as

$$cost(t_i) = \begin{cases} 0,\ \mathcal{S}(t_i) \in \mathcal{E}_\mathcal{I} \\ p(v_i) \cdot RT(t_i, \mathcal{S}(t_i)),\ \mathcal{S}(t_i) \in \mathcal{C}_\mathcal{I} \\ p(v_i) \cdot RT(t_i, \mathcal{S}(t_i)) + p_f(v_i, \mathcal{S}(t_i), \eta),\ \mathcal{S}(t_i) \in \mathcal{F}_\mathcal{I} \end{cases} \tag{18}$$

Where $p(v_i)$ depends on the `CPU` and `STORAGE` values of the VM v_i, namely

$$p(v_i) = \texttt{CPU}(v_i) \cdot p_{cpu} + \texttt{STORAGE}(v_i) \cdot p_{stor}. \tag{19}$$

Instead, p_f depends on a η parameter that models user preference for lower latency or cheaper price. We define the $p_f(v_i, \eta)$ function in Equation 20.

$$p_f(v_i, \mathcal{S}(t_i), \eta) = \frac{T_f(t_i)}{\eta} - \sqrt{\frac{\eta \cdot p(v_i, \mathcal{S}(t_i)) + T_f}{\eta^2 \cdot \min_{n_j \in \mathcal{F}_\mathcal{I}} RT(t_i, n_j)}}, \tag{20}$$

where $T_f = \sum_{n_i \in \mathcal{C}_\mathcal{I}} \frac{\texttt{lat}(e_j, n_i)}{|\mathcal{C}_\mathcal{I}|} + \frac{1}{\texttt{CPU}(n_i)} - \sum_{n_k \in \mathcal{F}_\mathcal{I}} \frac{\texttt{lat}(e_j, n_k)}{|\mathcal{F}_\mathcal{I}|}$ and η is a value between 0.01 and 1, where a value closer to 0.01 means that user prefers to have lower latency, while a value closer to 1 indicates that user prefers price over latency. In [19] it is shown that this function maximizes both providers' revenue and users' satisfaction, making it a possible pricing model for Fog. In this work, we consider $\eta = 1$, which considers a scenario where users are more interested in saving money than in having a very fast execution, meaning a lower penalty for execution on the Fog.

III. Multi-Objective Algorithm

Workflow scheduling is an optimization problem, which is known to be NP-complete [20]. Moreover, the introduction of the concept of Fog offloading, described in Section II, further aggravates the problem of finding close to optimal scheduling solutions. Therefore, to search for scheduling solutions, we utilize modified optimization algorithm, based on the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [21]. NSGA-II is an evolutionary multi-objective optimization algorithm that searches for Pareto-optimal solutions by promoting the principle of elitism. This principle implies that the solutions with highest fitness in a given population should always be preserved and never deleted. Furthermore, NSGA-II increases the exploration efficiency of the problem space by implementing explicit mechanisms for diversity preservation in the population, and by emphasizing the non-dominated solutions on each iteration. As with any other genetic algorithm, the offspring population P_o is created by applying crossover and mutation operations on the parent population P_p. More concretely, in our implementation we utilize Simulated Binary Crossover (SBX) and polynomial mutation, thus better sustaining the diversity in the population. In order to guarantee that the best solutions are always preserved, the offspring and parents populations are then combined together to form P_c. Only then a non-dominated sorting is applied, thus identifying a set of non-dominated Pareto solutions in the combined P_c population. The above described process is repeated until the maximal number of generations N_g is reached, thus identifying the final set of Pareto optimal solutions. Afterwards, a strategy called crowding distance sorting is applied to select solutions from the least crowded region in the Pareto front.

In relation to the scheduling model, we represent the individuals I in the population P as vectors V_I with a size equal to the number of tasks T_n that are scheduled. The values stored in each vector field corresponds to a single Cloud or Fog resource, where the task can be executed. Respectively, all processing elements in the environment are also assigned with unique IDs that are equal to the values saved in the vector field. In such way, each individual corresponds to a solution vector that represents unique global scheduling of all tasks in the workflow in relation to the Cloud and Fog nodes. Furthermore, we introduce an additional vector in which the dependencies between the tasks and the synchronization barriers, i.e. workflow levels, are uniquely marked. This allows every individual to be correctly represented and evaluated. An example of a single individual that corresponds to a solution vector for scheduling 8 tasks to 5 resources is presented in Figure 1.

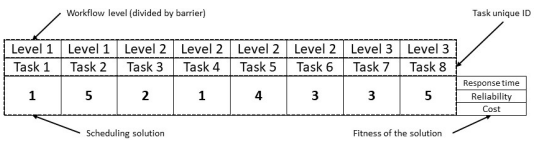


Figure 1: Solution vector

IV. Experimental Setup

IV.1 Workflows

In order to validate and evaluate the performance of the scheduling model, we carefully select four use-case workflows, from the field of natural sciences, with different computing and storage demands.

IV.1.1 Montage workflow

Montage is a distributed application for assembling multiple independent astronomical images, therefore constructing a mosaic that enables creation of user-specified projection views, coordinate systems and spatial scale [22]. The application is composed of independent modules that analyze the geometry of the images, which are later used for creating and managing the mosaics.

The mosaic is costructed in four steps: (1) discovery of the geometry of the input images in relation to the sky and calculation of the geometry of the output mosaic, (2) re-projection of the input images to an unified spatial scale and coordinate system, (3) modeling the background radiation in the input images, and (4) adding the re-projected, background-corrected images into the mosaic. The structure of the workflow of this application is presented in Figure 2.

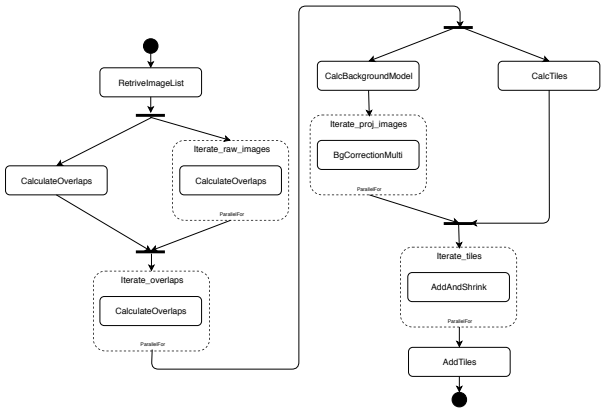


Figure 2: Montage workflow

IV.1.2 MeteoAG workflow

MeteoAG is a distributed grid/cloud scientific application for conduction meteorological simulations. The simulator utilizes the numerical atmospheric model RAMS [23] to produce atmospheric fields of heavy precipitation areas over the western part of Austria at a spatially and temporally much finer granularity in comparison with the current weather models [24]. The main goal of the simulator is to resolve and predict watersheds and thunderstorms in the Austrian alpine region. The structure of the workflow of this application is presented on Figure 3.

IV.1.3 Epigenomics workflow

The Epigenomics workflow is a data processing pipeline that executes various genome sequencing operations [25]. The original DNA

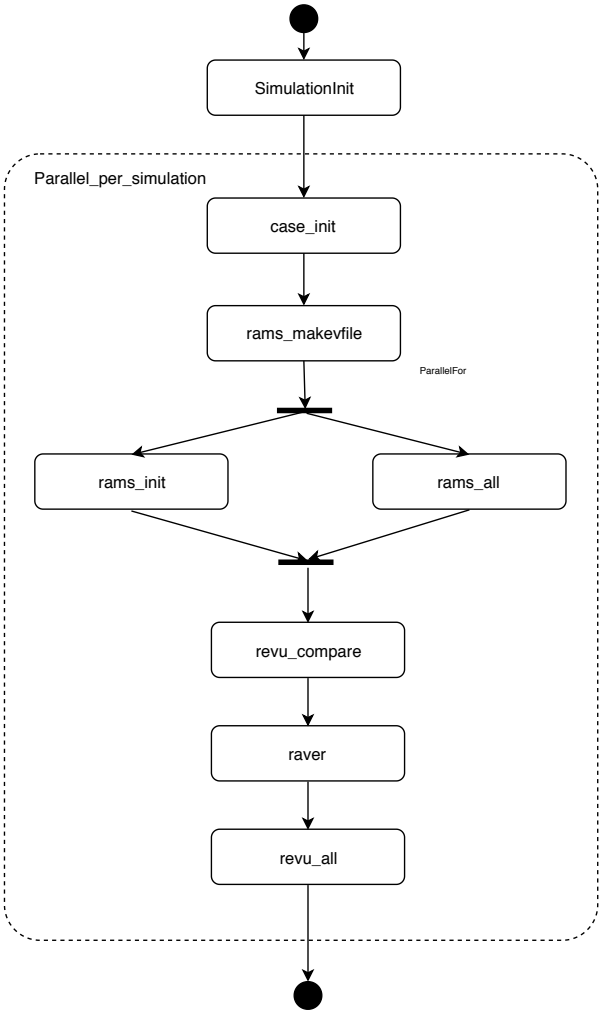


Figure 3: MeteoAG workflow

sequence data is generated by a specific genetic analysis system. After the acquisition, data sequences are split into several chunks that can be processed in parallel. The data in each chunk is converted into a specific format required by the sequence aligner. The rest of the tasks in the workflow involve filtering of noisy or contaminating sequences and reference genome mapping of the sequences into the correct location. Afterwards, a global map is created, which identifies the sequence density at each position in the genome. The structure of the workflow of this application is presented in Figure 4.

IV.1.4 Burrows-Wheeler Aligner workflow

BWA is an alignment tool that is used to perform low-divergent sequencing against a large reference genome [26]. The tool is based on backward search with Burrows-Wheeler Transform (BWT), thus efficiently aligning short sequencing reads against a large reference

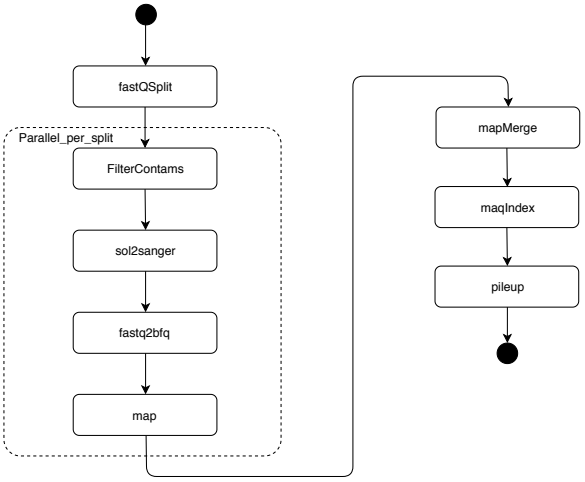


Figure 4: Epigenomics workflow

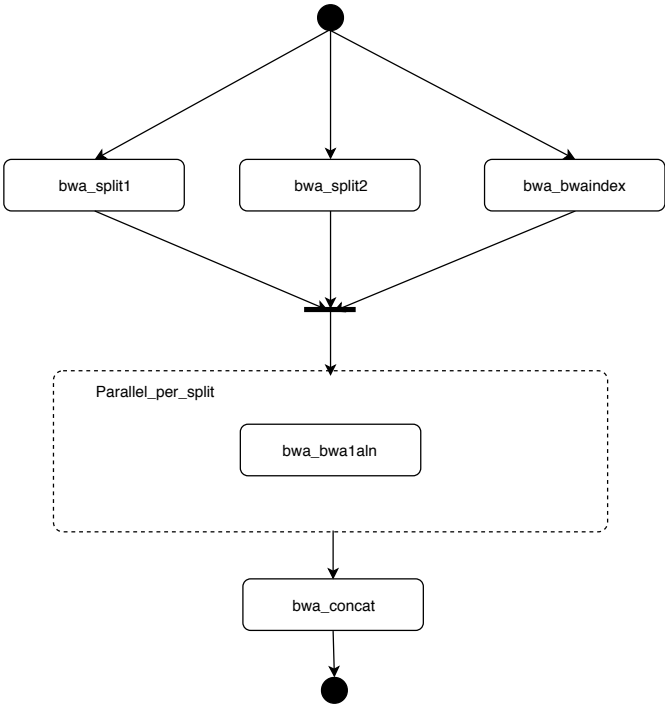


Figure 5: BWA workflow

sequence, allowing mismatches and gaps. BWA supports both base space reads, e.g. from Illumina sequencing machines and color space reads from AB SOLiD machines. It consists of three algorithms: (1) BWA-backtrack optimized for small sequences, (2) BWA-SW for longer sequences and (3) BWA-MEM is an optimized version of the BWA-SW with support for higher density data. The structure of the workflow of this application is presented in Figure 5.

IV.2 Simulation Framework

Different scenarios are evaluated through simulations, due to the unavailability of real-world Fog infrastructure to perform our experiments. At the time we write, several simulation frameworks for Fog/Edge computing have been proposed, such as iFogSim [27] and EdgeCloudSim [28]. However, they neither support workflow scheduling nor allow to specify our cost model. For this reason, we extend FogTorchPI [29] Monte-Carlo simulator to develop SLEIPNIR (Spark-enabled mobiLe Edge offloadIng Platform moNte-carlo sImulatoR). SLEIPNIR is a Monte-Carlo simulation framework for workflow scheduling on a Fog infrastructure. The use of Monte-Carlo simulation has been proved to be more suited for the variability of the target environment [30]. Besides, SLEIPNIR supports several DAG workflows scheduling algorithms, such as [31, 32]. SLEIPNIR is based on the model defined in Section II. In comparison to FogTorchPI, SLEIPNIR includes (1) Fog cost model defined in Section II.3.1, (2) support for DAG workflow scheduling and (3) implementation of typical scientific workflows described in Section IV.1. The extended version of the framework is available online (`https://github.com/vindem/sleipnir`). The input of our Monte-Carlo simulation is the infrastructure setup, consisting of the hardware characteristics of the computational nodes, network configuration and QoS capabilities and requirements, as well as the workflow to be executed. We want to ensure that each result falls in a confidence interval of ± 0.5 with 95% confidence. For this reason, each experiment is run $10,000$ times. The confidence interval is calculated on the samples using the formula $\bar{x} \pm 1.96 \cdot \frac{\sigma}{\sqrt{n}}$, where $\bar{x}$ is the mean of samples, 1.96 is the z^* value from the standard distribution for the desired confidence level, σ is the population standard deviation and n is the number of samples. The Monte-Carlo simulation is described in Figure 6. For each iteration, we perform a sampling of the infrastructure and of the workflow setup, defining for each task and link of the workflows the demand vector $\vec{d}$ (as in Equations 1 and 3), the capability vectors $\vec{c}$ for each computational node and the network link QoS (as in Equations 4, 5 and 7). Infrastructure and application used in sampling are described in Section IV.3. For each sampling, we run an instance of our MOWO algorithm and obtain a Pareto-front. For the implementation of the multi-objective optimization problem we utilize the jMetal multi-objective optimization framework, which allows pre-defined metaheuristics with pluggable objectives to be used [33]. Since different deployments can be generated during these runs, due to the high variability of the simulated environment, we store all the obtained deployments in a histogram. We consider only the deployment with the highest frequency in the histogram for comparisons. If there are two deployments with the same frequency, we consider the one with the best value for each objective.

IV.3 Infrastructure setup

For each iteration, we perform a sampling of both infrastructure and workflow setup and generate different setups, that allow to test the algorithm's performance in different settings. In the next sections, we describe how the sampling is performed.

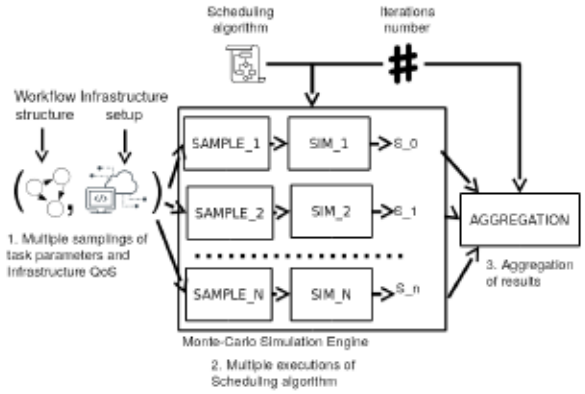


Figure 6: SLEIPNIR overview

$n_j \in \mathcal{N}_\mathcal{I}$	CPU	MIPS	STORAGE	p_{cpu}	p_{star}	lat $[ms]$	bw $[Mbps]$	$\mathcal{R}_i$
c^*	64	20,000	5,000	0.03	0.01	$\mathcal{G}(\mu = 200, \sigma = 33.5)$	1,000	0.9999
f0	16	15,000	5,000	0.03	0.01	lat	bw	0.9795
f1	16	10,000	1,000	0.03	0.01	lat	bw	0.9995
f2	16	12,000	1,000	0.03	0.01	lat	bw	0.9799
f3	16	16,000	1,000	0.03	0.01	lat	bw	0.9843
f4	16	13,000	1,000	0.03	0.01	lat	bw	0.9595
f5	16	8,000	1,000	0.03	0.01	lat	bw	0.9919

Table 1: CLOUD/FOG nodes configuration.

lat $[ms]$	bw $[Mbps]$	Probability
54	7.2	0.74
15	32	0.2
15	4	0.04
∞	0	0.02

Table 2: Network availability distribution.

IV.3.1 Computational nodes

As described in Section II.2, we mainly consider Cloud and Fog nodes. We assume two different configurations for our experiments: CLOUDONLY, where we only have 12 Cloud nodes and CLOUD-FOG, a heterogeneous configuration with 6 Cloud nodes and 6 Fog nodes, that is a realistic setup for the type of workflows that we want to run. The reason for this choice lies in the fact that we want to perform a fair comparison between the two setups, using the same number of computational nodes and showing that the differences that we observe are due to the heterogeneity and the different network setup. Concerning the sampling, at each iteration we randomly generate the latency and bandwidth between each node, without changing the nodes' hardware specification. The configurations for each node are summarized in Table 1. Concerning the cost of execution, it depends on the price for instantiating VMs on Cloud and Fog nodes. We assume, for simplicity, that VMs can be instantiated with the same size of the task, and that each task is deployed to exactly one VM. Therefore, the price of VMs is calculated based on (1) the demand vector of the task that is assigned to it and (2) the location where VM is assigned. The values in Table 1 are used as p_{cpu} and p_{star} values in Equation 19, while the penalty p_f for execution on Fog nodes is calculated based on Equation 17. Latency and bandwidth values refer to the connections outgoing from each node. Cloud latency is assumed to be equal to the Internet latency, that is between 100 and 300 milliseconds according to [34]. Therefore, we simulate it using a random Gaussian variable with the indicated mean and standard deviation. Regarding latency and bandwidth of Fog nodes, we use the distribution identified by [29], summarized in Table 2.

IV.3.2 Workflow setup

Concerning the workflows, our goal is to generate applications that resemble real-world ones in terms of structure and whose tasks and edges simulate, respectively, task requirements and user demands. For this reason, during every execution we generate a new value for *SIZE*, and $DATA_{out}$ for each task, as defined in Equation 1 while $DATA_{in}$ depends on predecessors' $DATA_{out}$ as defined in Equation 2. The first generation is done randomly, by sampling a value using different λ for exponential distribution: for SIZE, $\lambda = \{5,000; 10,000; 15,000; 20000\}$ MI, while for $\texttt{DATA}_{in}$ and $\texttt{DATA}_{out}$ we set $\lambda = \{62,500; 125,000; 250,000; 500,000\}$ bytes. We assume such values to be representatives of real world scenarios in terms of data size and allow us to simulate tasks with different computational demands. For each edge (t_i, t_j) between two tasks, we define user latency requirements as in Equation 3. The latency requirements are generated by using an exponential distribution with $\lambda = 1,000$ ms. This value allows to generate response time requirements that are representatives of typical near-real-time applications, as stated by [13].

IV.4 Baseline algorithms

In this section, we describe the single-objective algorithms used as baseline for our work. In this comparison, we considered different workflow scheduling algorithms: HEFT [35] and PEFT [36]. However,during experimentation, we noticed that results given by the two algorithms were very similar. Accordingly, we describe only the results of the HEFT and HEFT-based heuristics and we focus therefore on the following single-objective algorithms: HEFT, MAXREL and MINCOST.

IV.4.1 HEFT algorithm

The idea behind HEFT algorithm [35, 37] is to schedule each DAG node according to its critical path length to the exit node. This parameter is modelled by the *rank* of a node, defined as

$$rank(t_i) = \begin{cases} \bar{\texttt{RT}}(\texttt{t}_\texttt{i}) \ \text{ if } t_i \text{ is the exit task.} \\ \bar{\texttt{RT}}(\texttt{t}_\texttt{i}) + \max_{t_j \in \delta_{out}(t_i)} (\bar{\texttt{OT}}(\texttt{t}_\texttt{j}) + rank(t_j)) \end{cases} \tag{21}$$

Where $\bar{\texttt{RT}}(\texttt{t}_\texttt{i})$ is the average running time of task t_i calculated over each node in $\mathcal{N}_\mathcal{I}$, namely

$$\bar{\texttt{RT}}(\texttt{t}_\texttt{i}) = \frac{\sum_{n_j \in \mathcal{N}_\mathcal{I}} RT_{local}(t_i, n_j)}{|\mathcal{N}_\mathcal{I}|}, \tag{22}$$

while $\overline{\mathtt{OT}}(\mathtt{t_i})$ is the average offloading time for task t_i

$$\overline{\mathtt{OT}}(\mathtt{t_i}) = \frac{\sum_{n_j \in \mathcal{N}_\mathcal{I}} OT(t_i, md, n_j)}{|\mathcal{N}_\mathcal{I}|}. \quad (23)$$

We compute *rank* for each task of $\mathcal{W}$ recursively, starting from the SINK task. After this, we put tasks in a priority queue according to their decreasing rank. At each iteration we select the node with the highest rank, which is guaranteed to obtain a valid topological sort of tasks. Moreover, to obtain a valid scheduling for $\mathcal{W}$, we consider only nodes respecting constraints defined in Definition 9. Among these nodes, we select the best node for the selected objective. HEFT pseudocode is described by Algorithm 1.

Algorithm 1 *HEFT algorithm.*

```
1: function HEFT(W, I)
2:     computeRank(W, N_I)
3:     S ← ∅
4:     priorityQueue ← sortByRank(W)
5:     while !isEmpty(priorityQueue) do
6:         t ← first(priorityQueue)
7:         var ← +∞
8:         for n ∈ compatibleNodes(t) do
9:             cur ← RT(t, n)[1]
10:            if cur < var then
11:                target ← n
12:                var ← cur
13:            end if
14:        end for
15:        S ← S ∪ (t, target)
16:    end while
17: return S
18: end function
```

IV.4.2 MAXREL algorithm

Concerning *RL* objective, the algorithm performs offloading of task t_i to the node n_j that ensures the highest reliability. Ready tasks are also selected according to the $rank(t_i)$ function, defined in Equation 21. Afterwards, we schedule the task with the highest $rank(t_i)$ on the node n_j with the highest $\mathcal{R}_j$ (see Equation 16). This selection ensures that a valid schedule will be generated, since the task selection based on $rank(t_i)$ value ensures that a topological order of the tasks in $\mathcal{W}$. In addition, it ensures that the maximum reliability for a valid schedule is achieved. MAXREL is summarized in Algorithm 2.

Algorithm 2 *MAXREL algorithm.*

```
1: function MAXREL(W, I)
2:     computeRank(W, N_I)
3:     S ← ∅
4:     priorityQueue ← sortByRank(W)
5:     while !isEmpty(priorityQueue) do
6:         t ← first(priorityQueue)
7:         var ← +∞
8:         for n ∈ compatibleNodes(t) do
9:             cur ← R_n
10:            if cur < var then
11:                target ← n
12:                var ← cur
13:            end if
14:        end for
15:        S ← S ∪ (t, target)
16:    end while
17: return S
18: end function
```

IV.4.3 MINCOST algorithm

Concerning *COST* objective, the best solution for minimizing cost is running all the tasks on the end device. However, this results in a significantly higher response time, due to the its limited capabilities. To this end, the algorithm performs offloading of task t_i to the node n_j that ensures the lower $cost(t_i, n_j)$. Ready tasks are also selected according to the $rank(t_i)$ function, defined in Equation 21. MINCOST is summarized in Algorithm 3.

Algorithm 3 *MINCOST algorithm.*

```
1: function MINCOST(W, I)
2:     computeRank(W, N_I)
3:     S ← ∅
4:     priorityQueue ← sortByRank(W)
5:     while !isEmpty(priorityQueue) do
6:         t ← first(priorityQueue)
7:         var ← +∞
8:         for n ∈ compatibleNodes(t) do
9:             cur ← cost(t, n)
10:            if cur < var then
11:                target ← n
12:                var ← cur
13:            end if
14:        end for
15:        S ← S ∪ (t, target)
16:    end while
17: return S
18: end function
```

V. Results

The goals of our evaluations are twofold: first of all, we want to evaluate the benefits of executing scientific workflows on Fog resources, showing the improvement in terms of resource time and investigating how cost and reliability are influenced. Secondly, we want to investigate the benefits of our multi-objective MOWO approach for scheduling of scientific workflows. For this reason, we design the following two sets of experiments.

V.1 CLOUDONLY vs CLOUD/FOG

In this experiment, we evaluate the response time, the cost and the reliability for executing the workflows described in Section IV.1 using different settings. We compare the three objectives in two different scenarios: (1) CLOUDONLY, where only Cloud data centers are used for the execution of the workflows, and (2) CLOUDFOG, where a combination of Cloud data centers and Fog nodes are used. We use state-of-the-art heuristic to find solutions for each objective separately: HEFT (Heterogeneous Earliest Finish Time), described in [35], for the response time objective; MINCOST, based on HEFT, but selecting the node with the lower cost instead of the lower response time, for the cost objective; MAXREL, based on HEFT, but selecting the node with the higher reliability, for the reliability objective. The goal of this experiment is to check the different possibilities offered by the use of Fog nodes, and to show the necessity of a multi-objective algorithm to explore all the possible trade-offs. First of all, we check the performance of the algorithms by varying the $SIZE$ of each $t_i \in \mathcal{T}_\mathcal{W}$. Each $SIZE(t_i)$ value is sampled at the beginning of each iteration from an exponential distribution with a given λ parameter. To simulate different size values, we perform four simulation runs, each one for a pre-defined number

of iterations, using $\lambda = \{1,000; 5,000; 10,000; 20,000\}$. These values for λ allow us to simulate realistic task execution times on our simulation infrastructure. The results for the objectives are then averaged over 1,000 runs for statistical significance. The results for HEFT are presented in Figure 7, for MINCOST in Figure 8 and for MAXREL in Figure 9. CL results are for CLOUDONLY scenario, while CF are for CLOUDFOG. As we can see, the response time is significantly lower on CLOUDFOG when using smaller MIPS values, i.e. we obtain a lower response time up to 71% compared to CLOUDONLY. This is because for smaller tasks the use of Fog nodes allows to exploit the lower latency of the Fog layer in comparison with the Cloud layer. For larger task sizes, however, the higher computational cost makes execution on Cloud more preferable, since the lower computational capabilities of Fog nodes according to the selected configuration significantly slow down the execution of computationally-intensive tasks, reducing the positive effects of lower latencies. The high cost that we observe when using HEFT can be explained by the fact that the usage of more Fog nodes incurs in higher penalties, due to the cost model designed in Section II.3.1. It should also be noted that in this scenario, both MINCOST and MAXREL tend to select Cloud data centers rather than Fog nodes, due to the higher reliability of Cloud data centers and the absence of cost penalties on these nodes. In conclusion, the results show that the use of Fog is beneficial for task with low computation and high data transmission, making it the best choice for optimizing response time for this type of tasks. However, optimizing only for response time has a negative effect on both cost and reliability. Therefore, it is necessary to explore methods that allow selection of a trade-off solution, such as the one presented in this paper.

V.2 Multi-objective Algorithm Evaluation

In this scenario, we evaluate the efficiency of the MOWO task offloading approach in relation to the other state-of-the-art methods, namely HEFT, MINCOST and MAXREL. We evaluate the response time, the cost and the reliability for executing the workflows described in Section IV.1. For each $t_i \in \mathcal{T}_{\mathcal{W}}$, we generate this value using an exponential distribution with $\lambda = \{1,000; 5,000; 10,000; 20,000\}$ MI for different simulation runs. We set $\lambda = \{50, 0000\}$ bytes for $\texttt{DATA}_{in}$ and $\texttt{DATA}_{out}$ to simulate complex queries. The results for each objective are averaged over 1,000 runs. The results for the Montage workflow are presented in Figure 10, for Epigenomics in Figure 11, for BWA in Figure 12 and for METEO-AG in Figure 13. Through the analysis of the results, clear patterns can be identified. The MOWO approach achieves lower response times of up to 30% in relation to all other methods, especially for workflows composed of lower number of less complex data-intensive tasks, such as BWA and METEO-AG. At the same time, MOWO approach achieves reliability values close to 99%, which are close to those of MAXREL approach. Furthermore, the cost of executing the workflow is very similar to the MINCOST and MAXREL approaches, although for some workflows, such as Montage, it can be slightly higher, due to the tendency of the MOWO approach to utilize the Fog more than the Cloud. Overall, it can be concluded, that for tasks with smaller computing requirements and frequent data transfers, the MOWO approach provides benefits in relation to the three objectives. For more complex tasks, the advantage of the MOWO algorithm is reduced, due to the fact that the Fog has only limited number of computing resources. Therefore, for more complex tasks, the lower communication latency does not provide significant advantage.

As our research deals with the implementation of a NP-complete combinatorial multi-objective problem in large scale Cloud/Fog environment, we present an experimental results that demonstrate how our approach can provide high quality workflows' scheduling solutions across a large set of devices. To begin with, in Figure 14a we present the correlation between the number of evaluations and the quality of the solutions represented through the Hypervolume (HV) [38]. For this experiment the number of evaluations was gradually increased from 1,000 to 14,000 with a population of 100 individuals. The results show that the best quality solutions are achieved when the number of evaluation is around 7,500. In cases in which the number of evaluation is increased above 7,500, no improvement in the quality of the solutions can be observed.

Furthermore, we evaluate the scalability of the MOWO approach in relation to the size of the workflows. Figure 14b presents the correlation between the execution time of the MOWO algorithm and the number of tasks in the workflow. To properly evaluate the overhead of the genetic algorithm we also compare the execution time of MOWO with HEFT. The size of each task is generated using an exponential distribution whose lambda is set to 1,000 and the number of evaluations to 7,500. The results clearly show that the algorithm can scale relatively well in relation to the workflow size. When the length of the workflow is raised from 20 to 100 tasks, the execution time is increased on average only by 29%. Lastly, in Table 3 we provide a detailed evaluation of the quality of the solutions in relation to the workflow characteristics, namely, task requirements and data transfer sizes. We conduct the analysis with $\lambda = \{1,000; 5,000; 10,000; 20,000\}$ MI and $DATA_{in}$ and $DATA_{out}$ set to $\lambda = \{62,500; 125,000; 250,000; 500,000\}$ bytes. We can conclude that MOWO performs well for different MIPS and data requirements, although for higher data sizes and MIPS the quality of the solutions is slightly better, which can be accredited to the increase of the solution space.

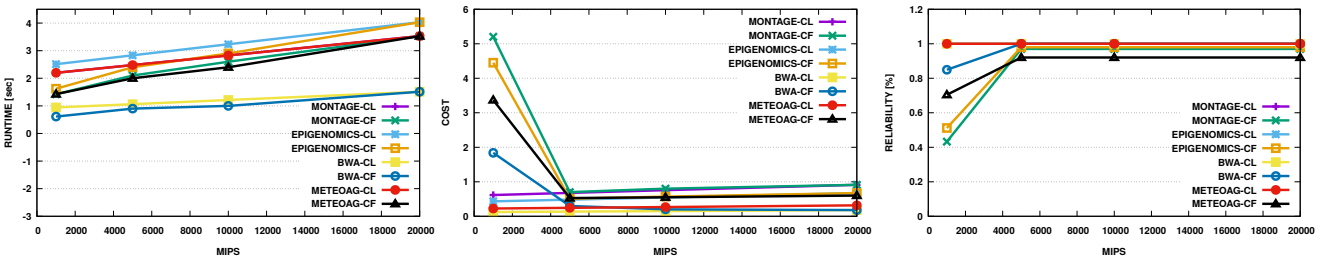
Message (Bytes)	62,500		125,000				250,000				500,000			
Task ($SIZE(t_i)$)	HV	STD	HV	STD	SGN	ABS	HV	STD	SGN	ABS	HV	STD	SGN	ABS
1,000	0.4784	0.0211	0.5428	0.0183	1	0.0643	0.6493	0.0249	1	0.1708	0.6508	0.0395	1	0.1723
5,000	0.5211	0.0441	0.5688	0.0475	1	0.0477	0.5306	0.0560	1	0.0095	0.5413	0.0601	1	0.0202
10,000	0.5844	0.0288	0.5355	0.0508	−1	0.0489	0.6255	0.0531	1	0.0410	0.6802	0.0432	1	0.0957
20,000	0.6063	0.0856	0.5705	0.0733	−1	0.0358	0.5585	0.0433	−1	0.0477	0.6100	0.1024	1	0.0037

Table 3: Solution quality in relation to the workflow specifics. SGN and ABS are calculated using Wilcoxon Signed-Rank Test [39]

VI. Related work

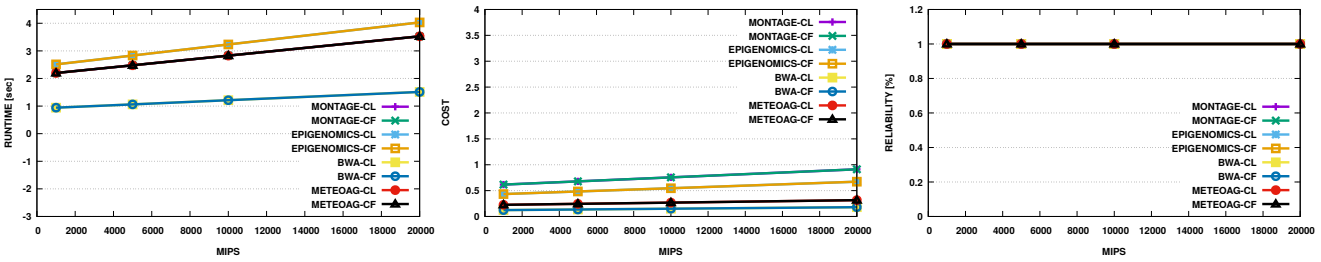
VI.1 Workflow scheduling

The straightforward approaches for multi-objective scheduling usually combine multiple objectives in a single function. This is accomplished by assigning different weights for the objective functions, therefore forming a single aggregated function. To illustrate, in [40] the reliability and makespan functions are combined into a single one through a user-provided weight vector. Similar approach is implemented in [41], in which both objectives are given the same weight without any requirement for user input. In general, these approaches lack the ability to efficiently explore the problem space as they depend on a priori weight assignment, which in many cases is performed without prior knowledge of the problem. Another



(a) Response time *(b) Cost* *(c) Reliability*

Figure 7: HEFT Results, 4 Workflows, CLOUDONLY vs CLOUDFOG



(a) Response time *(b) Cost* *(c) Reliability*

Figure 8: MINCOST Results, 4 Workflows, CLOUDONLY vs CLOUDFOG

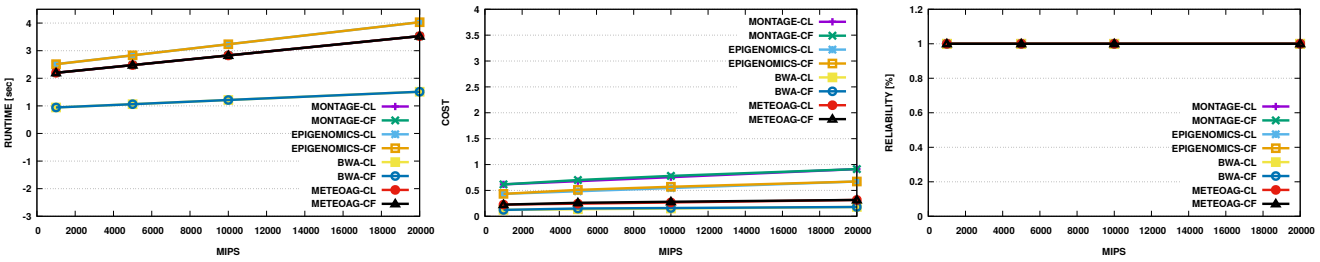
approach for combining conflicting objectives is to form a single function by imposing constraints to each objective. The main idea is to compute a solution which is optimized for a single objective, but with a guarantee that the constraints on the other objectives are not violated. An example work, which utilizes this approach for optimizing the economical costs for executing workflows in the Cloud is presented in [42]. Furthermore, in [43] an interesting approach for time/cost optimization in heterogeneous systems is presented.

In recent years, techniques that compute in parallel a given set of conflicting objectives have been described in the literature. In [44], the suitability of several genetic multi-objective algorithms for workflows scheduling has been analyzed. The results confirm the assumption that these algorithms can provide good solutions, unfortunately with a high computational cost. Therefore, a significant research effort has been made to explore for computationally efficient multi-objective heuristic approaches. To begin with, [15] presents a HEFT based approach for resource scheduling in Cloud, which utilizes multi-objective heuristics together with solution preserving algorithm to optimize execution time and energy efficiency. Furthermore, in [45] and [46] the authors propose workflow scheduling approaches for Cloud environments based on multi-objective genetic algorithms, namely NSGA-II and Particle Swarm Optimization (PSO). Besides, [47] proposes a novel approach for workflow scheduling in cloud–edge environment, which utilizes modified PSO algorithm with two-factor based objective functions to search for tradeoff between makespan and cost. Moreover, complex metaheuristics were also explored in the literature. In [48] the authors proposed a novel approach that improves the efficiency of the HEFT algorithm in two-objective problem space by utilizing gravitational search algorithm.

Unfortunately, all the aforementioned approaches have been designed for a specific Cloud infrastructure or have been simply ported from the Grid to the Cloud. The concurrent scheduling approaches do not consider the specific characteristics of the Fog, such as energy requirements, resources reliability and communication latency. Furthermore, their suitability for highly heterogeneous environments has not been explored.

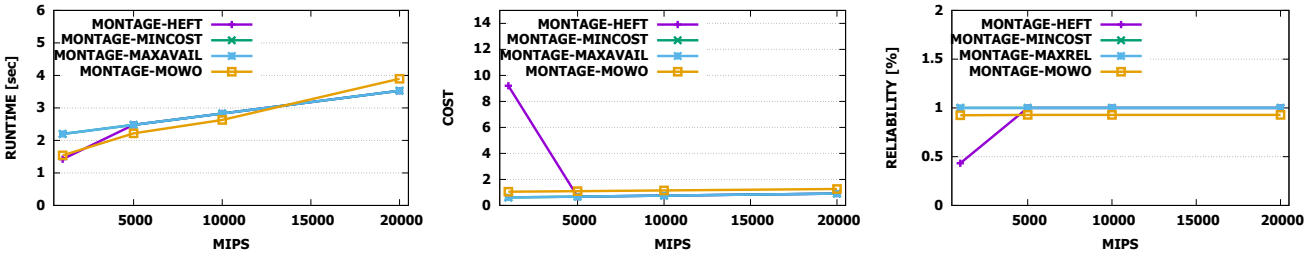
VI.2 Fog Offloading

Studies of different applications for Fog computing can be found in [49, 50], focusing in domains such as smart cities and IoT, with special attention to offloading. Offloading techniques are discussed in [51], with regard to Cloud computing and not to Fog computing. Concerning Fog offloading, several works apply it in the context of IoT, such as [8, 9], considering only a single offloading objective. Similar to our work, multi-objective approaches have been successfully applied in the context of Fog computing, either in the context of mobile offloading [52] or for resource scheduling [53]. However, none of these works consider the context of scientific workflows. In [54], the benefits of accelerating execution of scientific workflows by employing Fog resources are discussed, motivating our work. Typical examples of scientifically workflows are described in [25, 55]. The work in [56] proposed a heuristic-based algorithm for task scheduling in a Cloud/Fog system, allowing providers to utilize their own Fog nodes together with the rented Cloud resources. This work focuses on providing better cost and makespan trade-off for large-scale offloading applications.



(a) Response time *(b) Cost* *(c) Reliability*

Figure 9: MAXREL Results, 4 Workflows, CLOUDONLY vs CLOUDFOG



(a) Response time *(b) Cost* *(c) Reliability*

Figure 10: Results for the Montage Workflow, MOWO vs HEFT, MINCOST and MAXREL

Furthermore, the authors propose in [57] a service placement strategy in a Cloud/Fog environment considering both sequential and parallel service placement approaches, evaluated through a set of non-functional parameters, such as service delay and response time. Unfortunately, the concurrent algorithms for Fog offloading are limited to only two objectives. Furthermore, they implement strategies, in which, similarly to HEFT, all the tasks are first ranked and only then the scheduling is performed. These approaches consider in isolation the dependencies between the tasks and omit to contemplate the importance of the virtual infrastructure.

VII. Conclusion and Future Work

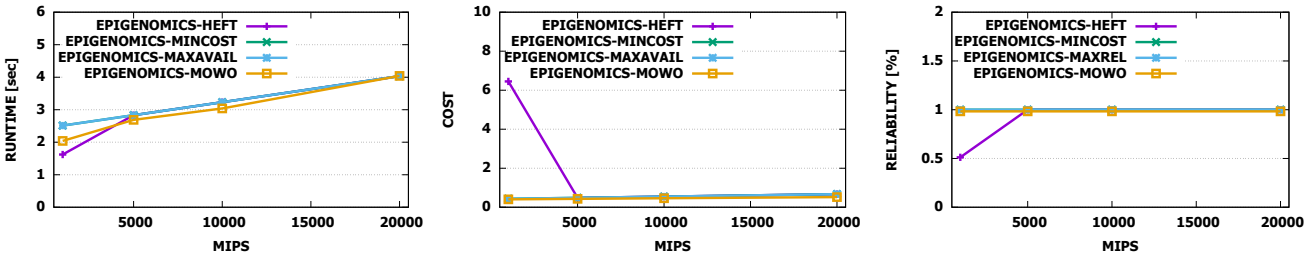
In this work, we have proposed a multi-objective approach for data-intensive workflows scheduling, which considers the response time, the cost and the reliability as objectives. First, we modeled the problem of workflow scheduling and task offloading by considering the functional requirements (i.e. response time, reliability and cost) of the workflows and the specifics of the Cloud and Fog environments. Then, we describe an algorithm based on NSGA-II, which is one of the most popular metaheuristics, in context of identification of optimal trade-off solution. Afterwards, we extend a Fog simulation framework to evaluate the suitability of the cloud offloading concept for data-intensive scientific workflows. We have proven that the use of Fog is beneficial for tasks with low computation and high data transmission, making it the best choice for optimizing response time of data-intensive tasks. Furthermore, we have evaluated the effectiveness of our algorithm, showing that the MOWO multi-objective algorithm can achieve results very close to the single-objective ones, achieving up to 30% lower response time when compared to HEFT for small task sizes, while maintaining cost and reliability values slightly lower than MINCOST and MAXREL respectively.

In the future, we plan to extend this model by considering different types of workloads, i.e. not only scientific but also related to other context, such as mobile gaming and video streaming. To this end, we will also provide a more detailed support for mobile devices, by including mobility and power consumption models. Furthermore, we will extend the model to support the dynamically changing Fog overlays, thus further improving the efficiency of the MOWO approach. Finally, we want to extend our model by considering also objectives that are of interest for the Cloud/Fog provider, such as energy consumption and profit, as in [58]. Moreover, we plan to perform a real-world implementation of this work on commercial Edge platform, such as Huawei Intelligent EdgeFabric[2], Azure IoT Edge[3] and AWS Lambda@Edge[4]

Acknowledgements

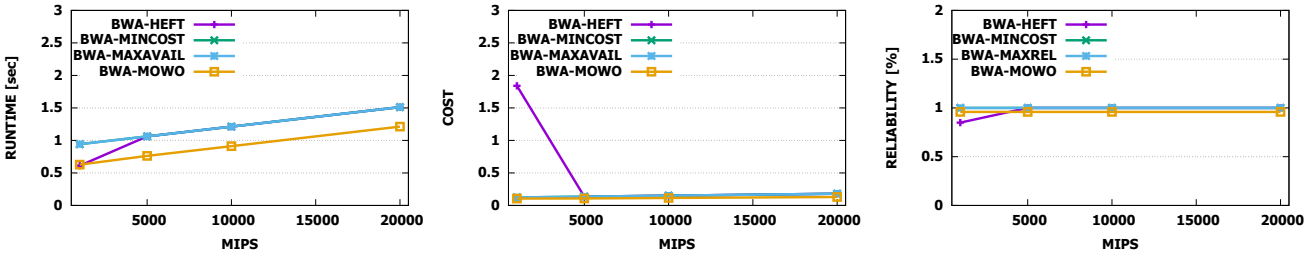
The work described in this paper has been funded through the Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015, the ASPIDE Project funded by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No 801091 and the ATOMICFOG project, OEAD AT/MK 08/2018.

[2] https://www.huaweicloud.com/en-us/product/ief.html
[3] https://azure.microsoft.com/it-it/services/iot-edge
[4] https://aws.amazon.com/it/lambda/edge



(a) Response time *(b) Cost* *(c) Reliability*

Figure 11: Results for the Epigenomics Workflow, MOWO vs HEFT, MINCOST and MAXREL



(a) Response time *(b) Cost* *(c) Reliability*

Figure 12: Results for the BWA Workflow, MOWO vs HEFT, MINCOST and MAXREL

References

[1] Cui Lin and Shiyong Lu. Scheduling scientific workflows elastically for cloud computing. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 746–747. IEEE, 2011.

[2] A. Szalay. Extreme data-intensive scientific computing. *Computing in Science Engineering*, 13(6):34–41, 2011.

[3] Mohammad Adibuzzaman, Poching DeLaurentis, Jennifer Hill, and Brian D Benneyworth. Big data in healthcare–the promises, challenges and opportunities from a research perspective: A case study with a model database. In *AMIA Annual Symposium Proceedings*, volume 2017, page 384. American Medical Informatics Association, 2017.

[4] Joseph C. Jacob, Daniel S. Katz, Thomas Prince, G. Bruce Berriman, John C. Good, Anastasia C. Laity, Ewa Deelman, Gurmeet Singh, and Mei-Hui Su. *The Montage Architecture for Grid-enabled Science Processing of Large, Distributed Datasets*. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2004.

[5] Christina Hoffa, Gaurang Mehta, Tim Freeman, Ewa Deelman, Kate Keahey, Bruce Berriman, and John Good. On the use of cloud computing for scientific workflows. In *2008 IEEE fourth international conference on eScience*, pages 640–645. IEEE, 2008.

[6] Dragi Kimovski, Humaira Ijaz, Nishant Saurabh, and Radu Prodan. Adaptive nature-inspired fog architecture. In *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–8. IEEE, 2018.

[7] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar. Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing*, 4(2):26–35, 2017.

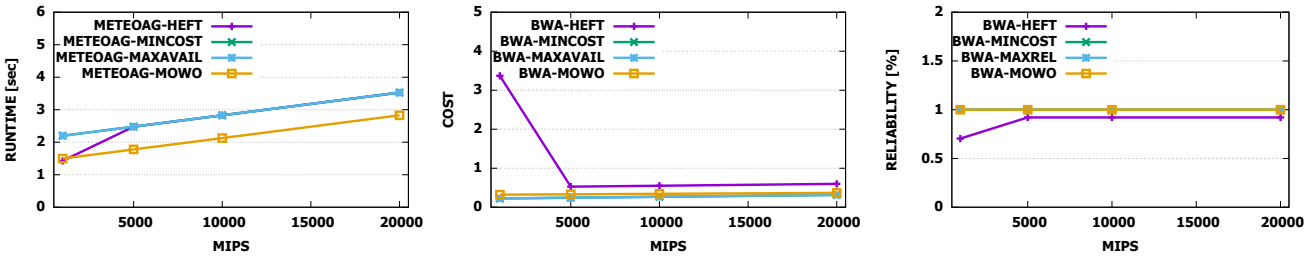
[8] X. Wang, Z. Ning, and L. Wang. Offloading in internet of vehicles: A fog-enabled real-time traffic management system. *IEEE Transactions on Industrial Informatics*, 14(10):4568–4578, Oct 2018.

[9] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue. On reducing iot service delay via fog offloading. *IEEE Internet of Things Journal*, 5(2):998–1010, April 2018.

[10] Vincenzo De Maio and Ivona Brandic. First hop mobile offloading of dag computations. In *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '18, pages 83–92. IEEE Press, 2018.

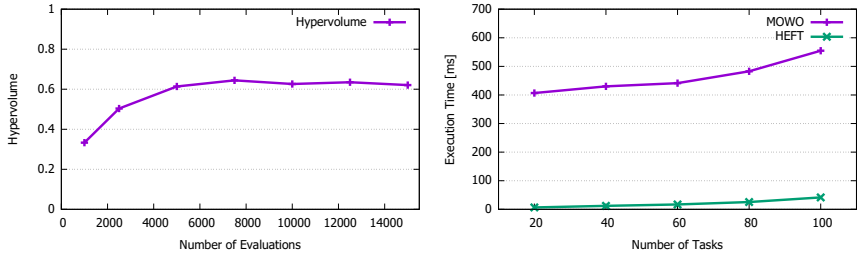
[11] Rafael Ferreira da Silva, Rosa Filgueira, Ilia Pietri, Ming Jiang, Rizos Sakellariou, and Ewa Deelman. A characterization of workflow management systems for extreme-scale applications. *Future Generation Computer Systems*, 75:228 – 238, 2017.

[12] Jinjun Chen and Yun Yang. Temporal dependency-based checkpoint selection for dynamic verification of temporal constraints in scientific workflow systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(3):9, 2011.



(a) Response time *(b) Cost* *(c) Reliability*

Figure 13: Results for the METEO-AG Workflow, MOWO vs HEFT, MINCOST and MAXREL



(a) Solutions quality *(b) Scalability*

Figure 14: Evaluation of the multi-objective characteristics of the MOWO approach

[13] Mahadev Satyanarayanan. The emergence of edge computing. *IEEE Computer*, 50(1):30–39, 2017.

[14] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16. ACM, 2012.

[15] Juan J Durillo, Hamid Mohammadi Fard, and Radu Prodan. Moheft: A multi-objective list-based method for workflow scheduling. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 185–192. IEEE, 2012.

[16] Guoquan Liu, Yifeng Zeng, Dong Li, and Yingke Chen. Schedule length and reliability-oriented multi-objective scheduling for distributed computing. *Soft Computing*, 19(6):1727–1737, Jun 2015.

[17] A. Ahmed and E. Ahmed. A survey on mobile edge computing. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–8, Jan 2016.

[18] May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. Cloud Computing Pricing Models: A Survey. *International Journal of Grid and Distributed Computing*, 6(5):93–106, 2013.

[19] Tianchu Zhao, Sheng Zhou, Xueying Guo, Yun Zhao, and Zhisheng Niu. Pricing policy and computational resource provisioning for delay-aware mobile edge computing. *2016 IEEE/CIC International Conference on Communications in China, ICCC 2016*, 2016.

[20] Jeffrey D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[21] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International conference on parallel problem solving from nature*, pages 849–858. Springer, 2000.

[22] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: the montage example. In *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12. Ieee, 2008.

[23] William R Cotton, RA Pielke Sr, RL Walko, GE Liston, CJ Tremback, H Jiang, RL McAnelly, JY Harrington, ME Nicholls, GG Carrio, et al. Rams 2001: Current status and future directions. *Meteorology and Atmospheric Physics*, 82(1-4):5–29, 2003.

[24] Felix Schüller, Jun Qin, Farrukh Nadeem, Radu Prodan, Thomas Fahringer, and Georg Mayr. Performance, scalability and quality of the meteorological grid workflow meteoag. In *Proceedings of 2nd Austrian Grid Symposium*, pages 21–23, 2006.

[25] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling. Scientific workflow applications on amazon ec2. In *2009 5th IEEE International Conference on E-Science Workshops*, pages 59–66, Dec 2009.

[26] José M Abuín, Juan C Pichel, Tomás F Pena, and Jorge Amigo. Bigbwa: approaching the burrows–wheeler aligner to big data technologies. *Bioinformatics*, 31(24):4003–4005, 2015.

[27] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *CoRR*, abs/1606.02007, 2016.

[28] C. Sonmez, A. Ozgovde, and C. Ersoy. Edgecloudsim: An environment for performance evaluation of edge computing systems. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 39–44.

[29] A. Brogi, S. Forti, and A. Ibrahim. How to best deploy your fog applications, probably. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 105–114.

[30] W. Nowak. Introduction to stochastic search and optimization. estimation, simulation, and control. *IEEE Transactions on Neural Networks*, 18(3):964–965, 2007.

[31] W. Zheng and R. Sakellariou. A monte-carlo approach for full-ahead stochastic dag scheduling. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pages 99–112.

[32] Abhishek Mishra and Anil Kumar Tripathi. A monte carlo algorithm for real time task scheduling on multi-core processors with software controlled dynamic voltage scaling. *Applied Mathematical Modelling*, 38(7):1929 – 1947, 2014.

[33] Antonio J. Nebro, Juan J. Durillo, and Matthieu Vergne. Redesigning the jmetal multi-objective optimization framework. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1093–1100. ACM, 2015.

[34] M. DeVirgilio, W. David Pan, L. L. Joiner, and D. Wu. Internet delay statistics: Measuring internet feel using a dichotomous hurst parameter. In *2013 Proceedings of IEEE Southeastcon*, pages 1–6.

[35] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[36] H. Arabnejad and J. G. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):682–694, March 2014.

[37] Hamid Mohammadi Fard, Radu Prodan, and Thomas Fahringer. Multi-objective list scheduling of workflow applications in distributed computing infrastructures. *Journal of Parallel and Distributed Computing*, 74(3):2152 – 2165, 2014.

[38] Lyndon While, Philip Hingston, Luigi Barone, and Simon Huband. A faster algorithm for calculating hypervolume. *IEEE transactions on evolutionary computation*, 10(1):29–38, 2006.

[39] Frank Wilcoxon, SK Katti, and Roberta A Wilcox. Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test. *Selected tables in mathematical statistics*, 1:171–259, 1970.

[40] Mourad Hakem and Franck Butelle. Reliability and scheduling on systems subject to failures. In *2007 International Conference on Parallel Processing (ICPP 2007)*, pages 38–38. IEEE, 2007.

[41] Saurabh Kumar Garg, Rajkumar Buyya, and Howard Jay Siegel. Time and cost trade-off management for scheduling parallel applications on utility grids. *Future Generation Computer Systems*, 26(8):1344–1355, 2010.

[42] Ruxia Li, Chase Q Wu, Aiqin Hou, Yongqiang Wang, Tianyu Gao, and Mingrui Xu. On scheduling of high-throughput scientific workflows under budget constraints in multi-cloud environments. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pages 1087–1094. IEEE, 2018.

[43] Hamid Arabnejad, Jorge G Barbosa, and Radu Prodan. Low-time complexity budget–deadline constrained workflow scheduling on heterogeneous resources. *Future Generation Computer Systems*, 55:29–40, 2016.

[44] Jia Yu, Michael Kirley, and Rajkumar Buyya. Multi-objective planning for workflow execution on grids. In *Proceedings of the 8th IEEE/ACM International conference on Grid Computing*, pages 10–17. IEEE Computer Society, 2007.

[45] Attiqa Rehman, Syed S Hussain, Zia ur Rehman, Seemal Zia, and Shahaboddin Shamshirband. Multi-objective approach of energy efficient workflow scheduling in cloud environments. *Concurrency and Computation: Practice and Experience*, page e4949, 2018.

[46] Ahmad M Manasrah and Hanan Ba Ali. Workflow scheduling using hybrid ga-pso algorithm in cloud computing. *Wireless Communications and Mobile Computing*, 2018, 2018.

[47] Ying Xie, Yuanwei Zhu, Yeguo Wang, Yongliang Cheng, Rongbin Xu, Abubakar Sadiq Sani, Dong Yuan, and Yun Yang. A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment. *Future Generation Computer Systems*, 97:361–378, 2019.

[48] Anubhav Choudhary, Indrajeet Gupta, Vishakha Singh, and Prasanta K Jana. A gsa based hybrid algorithm for bi-objective workflow scheduling in cloud computing. *Future Generation Computer Systems*, 83:14–26, 2018.

[49] Jorge Pereira, Leandro Ricardo, Miguel Luís, Carlos Senna, and Susana Sargento. Assessing the reliability of fog computing for smart mobility applications in vanets. *Future Generation Computer Systems*, 94:317 – 332, 2019.

[50] Qiang Wu, Jun Shen, Binbin Yong, Jianqing Wu, Fucun Li, Jinqiang Wang, and Qingguo Zhou. Smart fog based workflow for traffic control networks. *Future Generation Computer Systems*, 2019.

[51] Warley Junior, Eduardo Oliveira, Albertinin Santos, and Kelvin Lopes Dias. A context-sensitive offloading system using machine-learning classification algorithms for mobile cloud environment. *Future Generation Comp. Syst.*, 90:503–520, 2019.

[52] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi. Multiobjective optimization for computation offloading in fog computing. *IEEE Internet of Things Journal*, 5(1):283–294, Feb 2018.

[53] Yan Sun, Fuhong Lin, and Haitao Xu. Multi-objective optimization of resource scheduling in fog computing using an improved nsga-ii. *Wireless Personal Communications*, 102, 01 2018.

[54] S. Ramakrishnan, R. Reutiman, A. Chandra, and J. Weissman. Accelerating distributed workflows with edge resources. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pages 2129–2138, 2013.

[55] Gideon Juve and Ewa Deelman. *Scientific Workflows in the Cloud*, pages 71–91. Springer London, London, 2011.

[56] Xuan-Qui Pham and Eui-Nam Huh. Towards task scheduling in a cloud-fog computing system. In *Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific*, pages 1–4. IEEE, 2016.

[57] VB Souza, Xavier Masip-Bruin, Eva Marín-Tordera, Sergio Sànchez-López, Jordi Garcia, Guang-Jie Ren, Admela Jukan, and A Juan Ferrer. Towards a proper service placement in combined fog-to-cloud (f2c) architectures. *Future Generation Computer Systems*, 87:1–15, 2018.

[58] Vincenzo De Maio and Ivona Brandic. Multi-objective mobile edge provisioning in small cell clouds. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ICPE '19, pages 127–138, New York, NY, USA, 2019. ACM.