The final publication is available at Springer via https://doi.org/10.1007/s10586-013-0325-0

Multi-Objective Workflow Scheduling in Amazon EC2

Juan J. Durillo · Radu Prodan

the date of receipt and acceptance should be inserted later

Abstract Nowadays, scientists and companies are confronted with multiple competing goals such as makespan in high-performance computing and economic cost in Clouds that have to be simultaneously optimised. Multi-objective scheduling of scientific applications in these systems is therefore receiving increasing research attention. Most existing approaches typically aggregate all objectives in a single function, defined a-priori without any knowledge about the problem being solved, which negatively impacts the quality of the solutions. In contrast, Pareto-based approaches having as outcome a set of (nearly) optimal solutions that represent a tradeoff among the different objectives, have been scarcely studied. In this paper, we analyse MOHEFT, a Pareto-based list scheduling heuristic that provides the user with a set of tradeoff optimal solutions from which the one that better suits the user requirements can be manually selected. We demonstrate the potential of our method for multi-objective workflow scheduling on the commercial Amazon EC2 Cloud. We compare the quality of the MOHEFT tradeoff solutions with two state-of-the-art approaches using different synthetic and real-world workflows: the classical HEFT algorithm for single-objective scheduling and the SPEA2* genetic algorithm used in multi-objective optimisation problems. The results demonstrate that our approach is able to compute solutions of higher quality than SPEA2*. In addition, we show that MOHEFT is more suitable than SPEA2* for workflow scheduling in the context of commercial Clouds, since the genetic-based approach is unable of dealing with some of the constraints imposed by these systems.

Keywords Workflow scheduling · Cloud · Multi-objective optimisation · List-based heuristics

1 Introduction

Many scientists and researches are moving today towards Cloud computing for achieving high performance computing (HPC). The Cloud paradigm brings a new operational model in which resources are managed by specialised data centres and rented on-demand and only for the period they need to be used. Cost-wise, this new model is becoming very attractive for companies and institutions because it frees them from permanent hardware overprovisining, maintenance, and depreciation costs.

In previous e-science research [21], many scientists and researches have found in scientific workflows, from now on simply referred to as workflows, an attractive model of building large scale applications for heterogeneous wide-area parallel and distributed computing systems such as Grids. Typically, a scientific workflow application [19] consists of several (legacy) programs (referred from now on as tasks or activities) in the form of a dependency graph, where the input of some of these programs may depend on the output of the others. Once the application is composed as a workflow, its performance depends on how the individual tasks are mapped (scheduled) on to the available parallel and distributed resources. Traditionally in the HPC field, finding an optimal schedule of the tasks minimising the *makespan* or completion time of the whole workflow is the main objective. This represents a major challenge, since the problem of scheduling a set of different tasks onto a set of heterogeneous resources belongs to the class of NP-complete problems [23], and therefore, no polynomial algorithm for computing the optimal solution exists. As a consequence, many heuristics and meta-heuristics [4] for approximating a solution to this problem have been proposed [14, 22].

In the context of Cloud computing, the computed mapping must additionally optimise the *economic cost* incurred by renting resources. Today, most commercial Clouds offer heterogeneous types of resources at different prices and with different performance. For example, in Amazon EC2 (`http://aws.amazon.com/ec2/pricing/`) a user can choose among different types of instances, where the fastest resource is about eight times more expensive than the slowest one[1]. In these circumstances, the workflow scheduling problem has to be formulated as a *multi-objective optimisation problem (MOP)* which aims at optimising at least two conflicting criteria: makespan and economic cost of workflow's execution. The main characteristic of MOPs is that no single solution exists that is optimal with respect to all objectives, but a set of tradeoff solutions known as *Pareto front* [6]. Solutions within this set cannot be further improved in any of the considered objectives without causing the degradation of at least another objective. Most related work [12, 13, 2] simplifies workflow scheduling optimising several competing objectives to a single-objective problem by aggregating all the objectives in one analytical function. The main drawback of these approaches is that the aggregation of the different objectives is made a priori, with any knowledge about the workflow, infrastructure,

[1] These prices only refer to the Standard On-Demand Instances (September 2013)

and in general about the problem being solved. Therefore, the computed solution may not properly capture the user preferences. On the other hand, few approaches computing the tradeoff solutions have been proposed. Their main advantage over the aggregative ones is that the user is provided with a set of optimal solutions from which the one that better suits the requirement or preferences can be manually selected.

To address this gap, we introduce in this paper a new multi-objective workflow scheduling method called *Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT)* as an extension to the well-known HEFT [22] mono-objective workflow scheduling algorithm. Our proposal is a new heuristic-based method that computes a set of tradeoff solutions with a small additional overhead compared to the traditional single objective methods. In doing this, MOHEFT builds several intermediate workflow schedules in parallel in each step instead of a single one. To ensure the quality of the tradeoff solutions, MOHEFT uses dominance relationships and a metric called crowding distance to guarantee their diversity. MOHEFT is generic in the number and type of objectives, applied in this paper for optimising the makespan and economic cost of running workflow applications in an Amazon-based commercial Cloud.

Although early results [8] demonstrated that MOHEFT can deliver good results, these experiments were conducted on synthetic simulated resources and workflows instead of real-world ones. Moreover, they assumed that the number and type of resources were known beforehand which may be true in distributed systems in general, but not necessarily in Clouds. While in theory a Cloud user can access an infinite pool of resources, in practice most providers restrict this number to a maximum of N instances that can be simultaneously acquired. For example, in Amazon this maximum number is limited to $N = 20$ and could be enlarged through offline communication. Within this maximum number N, the user flexibly can choose between the different types of instances offered by the Cloud provider (e.g. `m1.small`, `m1.large`, `m1.xlarge`, `c1.medium`, `c1.xlarge` for Amazon EC2) with different performance and prices. The question which instances should compose the set of maximum size N for running the workflow becomes critical and has no single answer, since different combinations produce different tradeoff schedules. Moreover, the set of N instances does not need to be invariant during the whole workflow execution. For example, it may occur that one type of instance is particularly good at the beginning of the workflow execution, and a different type of instance is mostly beneficial towards the end. These additional constraints imposed by commercial Cloud systems require modifications to the proposed algorithms originally designed for heterogeneous distributed computing systems. Additionally, we also aim at highlighting the potential of the Pareto front as a tool for decision support, analysing how the user can exploit this information for improving the workflow schedule.

The contributions of this paper can be summarised as follows:

- We introduce a new Pareto-based multi-objective workflow scheduling algorithm for heterogeneous distributed computing systems called MOHEFT;

- We extend MOHEFT for dealing with commercial Cloud computing systems offering a limited amount of instances and a flexible combination of instance types;
- We compare the results of MOHEFT with two state-of-the-art algorithms adapted to the new scenario of commercial Clouds in the context of the Amazon EC2:
  - HEFT [22], a state-of-the-art workflow scheduling algorithm; our goal is to ensure that our method computes makespan solutions of the same quality or superior;
  - SPEA2* [25], a genetic metaheuristic for computing tradeoff solutions in the multi-objective optimisation theory.
- We analyse the tradeoff solutions computed by MOHEFT for different synthetic and real-world workflow applications.
- We analyse the impact of using a different number of resources on the computed tradeoff solutions.

The paper is organised as follows. The next section describes the related work, followed by a brief description of Amazon EC2 in Section 3. Section 4 defines the abstract workflow, resource, and problem definition underneath our approach. Section 5 gives a short background on multi-criteria optimisation required for a better understanding. In Section 6, we present our multi-objective workflow scheduling algorithm, adapted to the case of commercial Clouds. SPEA2*, and the modifications applied for dealing with the Cloud case are described in Section 7. We present in Section 8 the experimental setup for evaluating our technique on several synthetic and real-world workflows on Amazon EC2 (Section 9). Finally, we summarise the conclusions and the future work in Section 10.

2 Related Work

Many existing approaches to multi-objective workflow scheduling reduce the problem to mono-objective optimisation and compute a single solution only. These approaches differ in the considered objectives and how they are combined into a single one. Moreover, all works assume a heterogeneous computing system when the number and type of each resource in known beforehand. In contrast, we address a different constraint raised by existing commercial Clouds that offer a limited number of resources that can be flexibly instantiated by the offered instance types.

For example, [2,12,13] combine reliability (in terms of resource failures) and makespan using a weight vector aimed at expressing the user preferences over these two criteria. The main disadvantage of this method is that the computed solution depends on the selected weights, which is usually decided a-priori and without any information about the problem being solved. As only one solution is computed by this kind of approaches, it may not be satisfactory for the solved problem if the weights do not capture the user preferences in an accurate way. In addition, all objective functions have to be normalised to

the same interval to properly capture the user preferences, which requires the optimal solutions be known for each objective.

Other approaches are based on sorting and constraining the different criteria. The idea is to optimise for different objectives in a sequential fashion. Once an objective has been optimised and no further improvements are possible, the next objective in the list is considered. The optimisation of this new objective is carried out so that none of the imposed constrains over the previous criteria are violated. An example such as approach was proposed in [17] for optimising makespan and economic cost in utility Grids. The main weakness of this kind of approaches is that the number of objectives is limited to a few. Furthermore, the order in which the objectives are optimised requires some sort of preferential information, which may be difficult to derive. These drawbacks are overcome in [10], where a constraint-based list heuristic is proposed. This approach is targeted to optimise makespan, economic cost, energy consumption, and reliability and seeks for an optimal schedule by minimizing the distance to an ideal point and maximizing the distance to the constraints. A common problem to all these approaches is, however, that reasonable a-priori values for the constraints are often unknown until the first schedule is computed.

To the best of our knowledge, most of the approaches computing the whole Pareto front are based on the use of genetic algorithms. In [25], several state-of-the-art multi-objective genetic algorithms are analysed. The results show that these algorithms are able to compute high quality solutions, but require a high computation time. To overcome this drawback, the authors show that initialising these algorithms with high quality solutions for each criterion significantly reduces the time required to compute the set of tradeoff solutions. The computation of these initial solutions is performed using list-based heuristics such as HEFT for makespan, or customised versions of it for the other criteria. A bi-objective genetic algorithm is described [15]. This proposal optimises for makespan and energy consumption in Cloud systems. In [26], an enhanced version of the clasical multi-objective genetic algorithm, NSGA-II, is presented. This proposal includes the use of Quality of Service (QoS) constraints and is targeted to optimize five different QoS parameters. Multi-objective workflow scheduling attending to QoS parameters was also addressed with a differential evolution algorithm in [20]. Another version of NSGA-II, called R-NSGA-II, is proposed in [11] for optimising makespan, economic cost, and reliability. The main problem of R-NSGA-II is that it requires the user to specify a reference point, which may not be easy to derive before the optimisation is done. NSGA-II is in addition used for optimising makespan, cost, and reliability under budget constraints in [18].

In [5], a multi-objective list-based heuristic was proposed and applied to optimising makespan and reliability. In every iteration of the algorithm, a new set X of solutions is computed. As this set can get quite large, the number of solutions in every iteration is pruned in order to avoid an exhaustive search. This pruning is performed using the hypervolume metric indicating the quality of a set of tradeoff solutions (see Section 5). One problem with this approach is that the hypervolume is computationally expensive and exponential with the

number of objectives. Another drawback is that the hypervolume needs to be computed several times in every iteration, thus hiding the main computational advantage of list-based heuristic methods.

The technique we propose in this paper is a Pareto-based multi-objective list scheduling heuristic that extends HEFT for computing a set of tradeoff solutions. The method assures that at least one tradeoff solution is as good as the one computed by HEFT in terms of makespan. In contrast to [5], we employ a metric called crowding distance [7] of polynomial complexity with the number of solutions and the number of objectives to avoid the exhaustive search and the computationally expensive hypervolume method for pruning the set of tradeoff solutions. To the best of our knowledge, this is the first multi-objective optimisation proposal that extends a list-based heuristic for workflow scheduling in a Cloud computing environment.

3 Amazon EC2 Cloud

Amazon Elastic Computing Cloud (EC2) is an Infrastructure as a Service (IaaS) environment that opens Amazon's large computing infrastructure to its users. The term elastic references the fact that users can expand or shrink their infrastructure by acquiring/releasing resources or instances. In the following, we use the terms resource and instance interchangeably.

Currently, Amazon EC2 offers its users three type of instances:

- *Reserved Instances*, which allow users make a long-term reservation of several machines;
- *On-Demand Instances*, which allow users pay only for the resources they need and for the time required. Amazon EC2 charges its users per every hour of computation;
- *Spot Instances*, which allow users bid for unused Amazon EC2 capacity. In this mode, users specify the maximum price they are willing to pay for a resource. If this price is higher than the current spot price, which depends on the demand for resources, the user acquires the instance. If the spot price gets higher than the user bid, the instance is reclaimed by Amazon.

Reserved instances are only offered for periods between one and three years and therefore, can be considered in some sense as machines owned by the user. In this case, there is little or simply no room for optimising for cost. On the other hand, spot instances are an interesting mechanism for lowering the computing cost of many applications, but there is no guarantee of acquiring an instance if the bid is insufficient. Spot instances introduce therefore reliability as a third optimisation criterion, defined as the probability of acquiring the set of instances for the desired duration. In this paper, we focus only on on-demand instances, leaving the analysis of spot instances for future work.

Amazon EC2 offers fourteen different types of on-demand instances with different performance and price. The company describes the performance of these machines in terms of an abstract unit called *Elastic Compute Unit*

Table 1: Performance and price of various Amazon EC2 instances.

Instance	Mean performance [GFLOPS]	Price [$/h]	GFLOPS/$
`m1.small`	2.0	0.1	19.6
`m1.large`	7.1	0.4	17.9
`m1.xlarge`	11.4	0.8	14.2
`c1.medium`	3.9	0.2	19.6
`c1.xlarge`	50.0	0.8	62.5

(ECU), equivalent with a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor. In [1], Iosup *et al.* evaluated the Amazon EC2 resources for scientific computing and reported the average performance in millions of floating point operations per second (GFLOPs) of five different instance types thorough extensive benchmark experimentation. Table 1 summarises the average performance, the price per hour of computation, and the ratio GFLOPs per invested dollar of these resources. In this work, we will evaluate our multi-criteria workflow scheduling method on these five instance types.

4 Model

This section formally describes the workflow, resource, and problem definition underneath our approach.

4.1 Workflow Model

We model a *workflow application* as a directed acyclic graph: $W = (A, D)$ consisting of n tasks (also referred in the remaining of this paper as activities) $A = \bigcup_{i=1}^{n} \{A_i\}$, interconnected through control flow and data flow dependencies; $D = \{(A_i, A_j, Data_{ij}) \,|\, (A_i, A_j) \in A \times A\}$, where $Data_{ij}$ represents the size of the data which needs to be transferred from activity A_i to activity A_j. We use $pred(A_i) = \{A_k | \, (A_k, A_i, Data_{ki}) \in D\}$ to denote the *predecessor* set of activity A_i, (i.e. activities to be completed before starting A_i). Finally, we assume that the computational workload of every activity A_i is known and is given by the number of machine instructions required to be executed.

4.2 Resource Model

We assume that our hardware platform consists of a set of m heterogeneous resources $R = \cup_{j=1}^{m} R_j$, which can be of any type as provided by Amazon EC2 (e.g. `m1.small`, `m1.medium`, `m1.large`, `m1.xlarge`, `c1.medium`, `c1.xlarge`). For a given resource R_j of a certain type, we know its average performance measured in GFLOPs. In our workflow model we assume that an activity that is executed in any of these resource can benefit from a parallel execution using all the virtual cores exposed by the instance, achieving the performance

indicated in the last column of Table 1. The use of any of these resources is charged per every hour of computation following the Amazon prices indicated in the third column of that table. The final price is based not only on the resources' usage, but also in the data stored and transferred among different instances which depends on four components:

1. price per hours of resource's usage PE_{R_i};
2. price per MB of data storage PS_{R_i};
3. price per MB of data received PI_{R_i};
4. price per MB of data sent PO_{R_i}.

The prices of these components depend on the Cloud provider. Currently, Amazon EC2 does not charge for internal data transfers among EC2 instances which do not require a public IP address, i.e. which do not require to be publicly reachable from Internet. Amazon EC2 also does not charge for incoming data from Internet to EC2 instances. In the case of outgoing data, the first GB transferred each month is free, and up to 10TB of information can be transferred at a relative low price of 0.120$ per GB. For the data storage, the price charged by Amazon EC2 is 0.10$ per stored GB.

Finally, commercial Clouds such as Amazon EC2 introduce new constraints that must be included in the resource model. While in theory a user can access an infinite pool of resources, in practice most providers restrict this number to a maximum of N instances that can be simultaneously acquired. For example, in case of Amazon this maximum number is limited to 20 and can be enlarged through offline communication. Within this maximum number N, the user can flexibly choose between the different types of instances with different performance and prices. The question which instances to compose the set of maximum size N for running the workflow becomes critical and has no single answer since different systems of maximum size N will produce different tradeoff schedules. Moreover, the set of N instances does not have to be invariant during the whole workflow execution. For example, it may occur that one type of instance is particularly good at the beginning of the workflow execution, and a different type of instance the most beneficial at the end.

4.3 Problem Definition

Our problem consists in scheduling the execution of the workflow tasks on the Amazon resources such that the makespan and the economic costs are minimised. In the rest of this paper, we will use $sched(A_i)$ to denote the resource on which the task A_i is scheduled to be executed. We describe in the following how the two objectives of interest are computed.

4.3.1 Makespan

For computing the workflow makespan, it is first necessary to define the *execution time* $t_{(A_i,R_j)}$ of an activity A_i on a resource $R_j = sched\,(A_i)$ as the

sum of the time required for transferring the biggest input data from any $A_p \in pred\,(A_p)$ and the time required to complete A_i in R_j:

$$t_{(A_i,R_j)} = \max_{A_p \in pred(A_i)} \left\{ \frac{Data_{pi}}{b_{pj}} \right\} + \frac{workload\,(A_i)}{s_j}, \quad (1)$$

where $Data_{pi}$ is the size of the data to be transferred between A_p and A_i, b_{pj} is the bandwidth of one TCP stream between the resource where task A_p was executed and the resource R_j, $workload(A_i)$ the length of the task A_i in machine instructions, and s_j the speed of the resource R_j in number of machine instructions per second. Next, we can compute the *completion time* T_{A_i} of activity A_i considering the execution time of itself and its predecessors:

$$T_{A_i} = \begin{cases} t_{(A_i,sched(A_i))}, & pred\,(A_i) = \emptyset; \\ \max\limits_{A_p \in pred(A_i)} \left\{ T_{A_p} + t_{(A_i,sched(A_i))} \right\}, & pred\,(A_i) \neq \emptyset. \end{cases} \quad (2)$$

The workflow makespan is finally defined as the maximum completion time of all the activities in the workflow:

$$T_W = \max_{i \in [1,n]} \left\{ T_{(A_i,sched(A_i))} \right\}. \quad (3)$$

4.3.2 Economic Cost

The economic cost depends on two terms: the computation cost $C^{(comp)}$ and the cost of data transfer and storage $C^{(data)}$.

We define $C^{(data)}_{(A_i,R_j)}$ as the cost of the data transfers $In(A_i)$ and $Out\,(A_i)$ and storage $Data\,(A_i)$ resulting from executing activity A_i on resource R_j:

$$C^{(data)}_{(A_i,R_j)} = Data\,(A_i) \cdot t_{(A_i,R_j)} \cdot PS_{R_i} + In(A_i) \cdot PI_{R_i} + Out\,(A_i) \cdot PO_{R_i}, \quad (4)$$

In defining the cost $C^{(comp)}_{R_j}$ of using a resource R_j, we assume that for each task A_i executed on R_j we record two timestamps: $t^{(start)}_{A_i}$ when the activity starts and $t^{(end)}_{A_i}$ when the activity finishes its execution. The value $t^{(end)}_{A_i}$ can be computed as $t^{(start)}_{A_i} + t_{(A_i,R_j)} + \max_{A_i \in pred(A_p)} \left\{ \frac{Data_{ip}}{b_{jp}} \right\}$. We therefore consider that the times for transferring the input $In\,(A_i)$ and the output data $Out\,(A_i)$ are included in the interval between $t^{(start)}_{A_i}$ and $t^{(end)}_{A_i}$. In other words, these time stamps indicate the period of time on which the resource R_j needs to be active due to the execution of the activity A_i.

Let us consider now the set of all p activities scheduled on resource R_j denoted as $\{J_1, \ldots, J_p\}$, where $p < n$ and $sched\,(J_i) = R_j, i \in [1,p]$, sorted based on their start timestamp: $t^{(start)}_{J_1} < \ldots < t^{(start)}_{J_p}$. Based on this ordering, we cluster these activities in $q \leq p$ different *groups* $G^{(j)}_k$, $1 \leq k \leq q$, so that all activities in one group are executed consecutively without releasing the resource. After the activity with the largest start timestamp in the group completes, the resource is released.

We construct the first group $G_1^{(j)} = \{J_1, \ldots, J_r\}, r \leq p$, based on the following three rules:

1. The first activity J_1 belongs to the first group: $J_1 \in G_1^{(j)}$;
2. Every activity $J_i \in G_1^{(j)}, 2 \leq i \leq r$ completes before the resource is released. This means that J_i starts when the resource is still leased because of the execution of J_{i-1}:

$$t_{J_i}^{(start)} < t_{J_1}^{(start)} + \left\lceil \frac{t_{J_{i-1}}^{(end)} - t_{J_1}^{(start)}}{3600} \right\rceil \cdot 3600. \tag{5}$$

   We divide the total time in seconds of using a resource by 3600 in order to convert it to hours, and use the ceiling operator to round this value to complete hours of computation. Obviously, the resource will be rented for as many hours as required for finishing all the activities within this group.
3. The next activity (not part of the previous group) $J_{r+1} \notin G_1^{(j)}, r+1 \leq p$ starts in an instant of time $t_{J_{r+1}}^{start}$ when the resource has been already released, i.e., task J_r has finished its execution, the last rented period of one hour for executing J_r has expired, and the resource R_j was not needed in the period of time elapsed between $t_{J_r}^{end}$ and $t_{J_{r+1}}^{start}$. Mathematically, it can be expressed as:

$$t_{J_1}^{(start)} + \left\lceil \frac{t_{J_r}^{(end)} - t_{J_1}^{(start)}}{3600} \right\rceil \cdot 3600 < t_{J_{r+1}}^{(start)}. \tag{6}$$

Successive groups are built until the last activity J_p has been assigned to one group. The second group $G_2^{(j)}$ is constructed in the same way starting from the task J_{r+1} instead of J_1. The same strategy is used for the rest of the groups. Once all the groups have been created, we define the cost $C_{R_j}^{(comp)}$ of using the resource R_j as the number hours required for executing all groups multiplied by the cost per hour:

$$C_{R_j}^{(comp)} = PE_{R_j} \cdot \sum_{k=1}^{q} \left\lceil \frac{\sum_{A_i \in G_{R_j}^{(k)}} t_{(A_i,R_j)}}{3600} \right\rceil. \tag{7}$$

We compute the economic cost of executing the entire workflow $W = (A, D)$ as the computation cost on all m resources plus the cost for transferring and storing the data:

$$C_W = \sum_{j=1}^{m} C_{R_j}^{(comp)} + \sum_{(A_i,A_j,Data_{ij}) \in D} C_{(A_i,R_j)}^{(data)}. \tag{8}$$

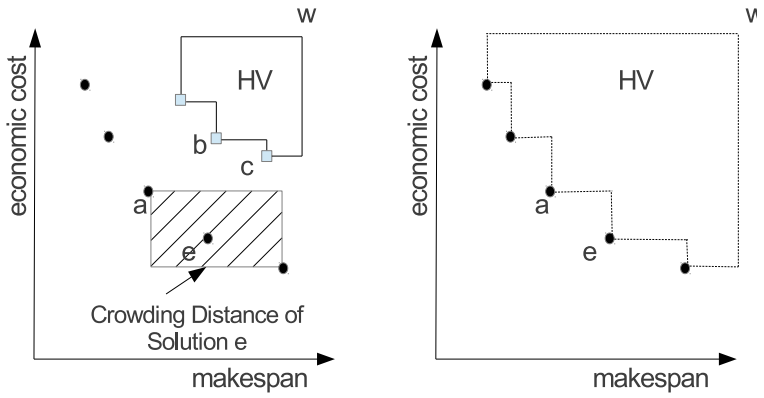


Fig. 1: Comparison of multi-objective tradeoff solutions.

5 Multi-Objective Optimisation Background

In this section, we introduce a few concepts from the *multi-objective optimisation* theory for a better understanding of this work. We assume without loss of generality that minimisation is the goal for all objectives, as any maximisation problem can be defined in terms of a minimisation too.

A general multi-objective optimisation problem can be formally defined as follows: find all the solutions x which minimise the vector function $f(x) = [f_1(x), f_2(x), \ldots, f_o(x)]$, where o represents the number of objectives to be optimised (two in our case, makespan and cost). Every solution x in our problem consists of two vectors: a vector of n components $(x_1, \ldots, x_n)$, being n the number of activities in the workflow ($n = |A|$) and $x_i = sched\,(A_i)$ the resource where the activity $A_i \in A$ will be executed; and a permutation p of size n representing the order in which those activities will execute.

As it is not possible to find a solution that minimises both makespan and economic cost simultaneously, we introduce the concept of *dominance*. A solution y dominates a solution z, if the makespan and economic cost of y are smaller than those of z. Conversely, two solutions are said to be non-dominated whenever none of them dominates the other (i.e. one is better in makespan and the other is better in cost). In Fig. 1 (left) for example, the solution labelled a dominates the one labelled b because it has better makespan and economic cost. Similarly, a dominates c. Meanwhile, a and d are non-dominated because a is better in makespan, but d is better in economic cost. The set of optimal non-dominated solutions is called Pareto front (the trend line containing the a, d, and e solutions) and represents a set of tradeoff solutions among the different objectives. Every solution in this set represents a different mapping of the workflow tasks with different makespans and costs.

A Pareto front can be seen as a tool for decision support and preference discovery. Its shape can provide insight to researches or scientists (from now decision makers), allowing them in many cases to explore the possible space of non-dominated solutions with certain properties, and possibly revealing regions of particular interest which cannot be seen until the Pareto front is computed and visualised. This way, the users do not have to set their preferences before finding a solution, instead the preferences are discovered afterwards. Nevertheless, not all Pareto fronts are valid. A good Pareto front is one which provides accuracy (solutions close the optimal ones) and diversity (uniformly cover all the possible ranges of optimal solutions). A way of measuring the quality of a set of tradeoff solutions is the *hypervolume*. Given a set of tradeoff solutions X, the hypervolume $HV(X)$ measures the area enclosed between the points in X and a reference point W (see Fig. 1), usually selected as the maximum objective value (e.g. highest makespan and economic cost). Therefore, the better and the more diverse the points contained in X are, the higher $HV(X)$. In Fig. 1, the set containing the solid round points is better than the set containing the solutions presented by squared shapes; therefore, the hypervolume of the set containing all the solid round points is higher than the set containing the solutions presented by squared shapes.

6 MOHEFT: Multi-Objective Heterogeneous Earliest Finish Time Algorithm

In this section we describe the MOHEFT algorithm as an extension to the HEFT list-based scheduling algorithm for optimising workflow makespan. For a better understanding, we start by describing HEFT and extend it afterwards for dealing with multiple objectives. After presenting both techniques, we describe how they can be extended for commercial Clouds.

6.1 HEFT: Heterogeneous Earliest Finish Time Algorithm

The Heterogeneous Earliest Finish Time Algorithm (HEFT) is a popular list-based heuristic scheduling algorithm for optimising the makespan [22] in workflow applications, described in pseudo-code in Algorithm 1. The method consists of two phases: ranking and mapping. In the ranking phase (line 2), the order in which the activities are being mapped is computed using the B-rank metric (distance of the activity to the end of the workflow). The idea of this ranking is to execute before those tasks having more dependent tasks than others. Further details about how to sort the tasks can be found in [22]. Once the execution order is determined, the second phase consists in assigning each task to the resources (lines 4–14) following the order computed in the first phase. For each task (line 4) and for each resource (line 6), the completion time of that task on that resource is computed (line 7). Finally, the task is mapped onto the resource where it is finished earlier (line 13). After all tasks have been mapped, the workflow schedule is returned (line 15).

Algorithm 1 HEFT algorithm.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: $R = \bigcup_{j=1}^{m} R_j$ ▷ Set of resources
Ensure: $sched_W = \{(A_i, sched\,(A_i))\,|\forall A_i \in A\}$ ▷ Workflow schedule
1: **function** HEFT(W,R)
2: $Rank \leftarrow$ B-RANK(A) ▷ Order the tasks according to B-rank
3: $sched_W \leftarrow \emptyset$ ▷ Initialize workflow schedule with empty set
4: **for** $i \leftarrow 1, n$ **do** ▷ Iterate over the ranked tasks
5: $T_{\min} \leftarrow \infty$
6: **for** $j \leftarrow 1, m$ **do** ▷ Iterate over all resources
7: $T_{Rank_i} \leftarrow \max_{A_p \in pred(Rank_i)} \left\{T_{A_p} + t_{(Rank_i, R_j)}\right\}$ ▷ Compute completion of $Rank_i$
8: **if** $T_{Rank_i} < T_{\min}$ **then** ▷ Save the minimum completion time
9: $T_{\min} \leftarrow T_{Rank_i}$
10: $R_{\min} \leftarrow R_j$
11: **end if**
12: **end for**
13: $sched_W \leftarrow sched_W \cup (Rank_i, R_{\min})$ ▷ Schedule the task
14: **end for**
15: **return** $sched_W$
16: **end function**

6.2 MOHEFT: Multi-Objective Heterogeneous Earliest Finish Time Algorithm

As described in the previous section, HEFT builds a solution by iteratively mapping the workflow tasks onto resources. That mapping is aimed at minimising the completion time of every single task, so in every iteration only the resource which minimises this goal is considered. When multiple objectives are considered, the goal is to compute a set of tradeoff solutions. To this end, we must allow the creation of several solutions at the same time instead of building a single solution. Additionally, instead of mapping every task onto the resource where it is finished earlier, we should allow mapping of tasks also to resources that provide a tradeoff between the considered objectives.

The MOHEFT algorithm extends HEFT with these ideas, as depicted in pseudocode in Algorithm 2. The only additional input parameter of MOHEFT is the size of the set of tradeoff solutions K. Similar to HEFT, our new method ranks first the tasks using the B-rank metric (line 2). However, instead of creating an empty solution as in HEFT, it creates a set S of K empty solutions (lines 3–5). Afterwards, the mapping phase of MOHEFT begins (lines 6–16). MOHEFT iterates first over the list of tasks (line 6) sorted by the execution order. The idea is to extend every solution in S by mapping the next task to be executed onto all m possible resources and store them in a temporal set S' which is initially empty (line 7). For creating these new solutions, we iterate over the set of resources (line 8) and the set S (line 9), and add the new extended intermediate schedules to the new set S' (line 11). This strategy results in an exhaustive search if we do not include any restrictions. Therefore, we save only the best K tradeoffs solutions from the temporary set S' to the set S (lines 14–15). We consider that a solution belongs to the best tradeoff if it is not dominated by any other solution and if it contributes to the diversity of the set. For this last criterion, we employ the crowding distance defined

Algorithm 2 MOHEFT algorithm.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: $R = \bigcup_{j=1}^{m} R_j$ ▷ Set of resources
Require: K ▷ Number of tradeoff schedules
Ensure: $S = \bigcup_{i=1}^{K} sched_W, sched_W = \{(A_i, sched\,(A_i))\,|\forall A_i \in A\}$ ▷ Set of K tradeoff schedules
1: **function** MOHEFT(W,R,K)
2: $Rank \leftarrow$ B-RANK(A) ▷ Order the tasks according to B-rank
3: **for** $k \leftarrow 1, K$ **do** ▷ Create K empty workflow schedules
4: $S_k \leftarrow \emptyset$
5: **end for**
6: **for** $i \leftarrow 1, n$ **do** ▷ Iterate over the ranked tasks
7: $S' \leftarrow \emptyset$
8: **for** $j \leftarrow 1, m$ **do** ▷ Iterate over all resources
9: **for** $k \leftarrow 1, K$ **do** ▷ Iterate over all tradeoff schedules
10: $s \leftarrow S_k \cup (Rank_i, R_j)$ ▷ Extend all intermediate schedules
11: $S' \leftarrow S' \cup \{s\}$ ▷ Add new mapping to all intermediate schedules
12: **end for**
13: **end for**
14: $S' \leftarrow$ SORTCROWDDIST(S', K) ▷ Sort according to crowding distance
15: $S \leftarrow$ FIRST(S', K) ▷ Choose K schedules with highest crowding distance
16: **end for**
17: **return** S
18: **end function**

in [7] and graphically depicted in Fig. 1, which gives a measure of the area surrounding a solution where no other tradeoff solution is placed. Our criterion is to prefer solutions with a higher crowding distance, since this means that the set represents a wider area of different tradeoff solutions. After assigning all the tasks (line 17), the algorithm returns the set of K best tradeoff solutions.

6.3 Cloud-aware Extensions

The HEFT algorithm was originally intended for workflow scheduling in heterogeneous computing systems in which the number and type of resources are known beforehand. MOHEFT works in the same way, requiring the number and type of each resource before its execution. In this section, we extend these two algorithms to deal with the new resource model encountered in commercial such as Amazon EC2 which constrain the total amount of resources to a certain number, and leave their type of resources to be flexibly chosen depending on demand (see Section 4.2).

To deal with this new scenario, we extended HEFT and MOHEFT to consider the input number of resources as $m = N \cdot I$, where I is the number of instance types offered by the provider. By doing so, we ensure that all combinations of resources of maximum size N are possible. Additionally, we must extend the algorithms to discard any non-valid schedule that use more than N simultaneous resources by setting the corresponding objective value for makespan and cost to infinite.

Algorithm 3 contains the pseudocode of the Cloud-aware HEFT algorithm. The only difference to the original HEFT is in lines 7–11. While the original version always computed the makespan of every new partial schedule, the Cloud-aware version checks if the partial schedule requires more than N re-

Algorithm 3 Cloud-aware HEFT algorithm.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: N ▷ Maximum simultaneous instances
Require: I ▷ Number of different instance types
Require: $R = \bigcup_{j=1}^{m} R_j$ ▷ Set of resources, where $m = N \cdot I$
Ensure: $sched_W = \{(A_i, sched\,(A_i))\,|\forall A_i \in A\}$ ▷ Workflow schedule
1: **function** HEFT(W,R)
2: $Rank \leftarrow$ B-RANK(A) ▷ Order the tasks according to B-rank
3: $sched_W \leftarrow \emptyset$ ▷ Initialize workflow schedule with empty set
4: **for** $i \leftarrow 1, n$ **do** ▷ Iterate over the ranked tasks
5: $T_{\min} \leftarrow \infty$
6: **for** $j \leftarrow 1, m$ **do** ▷ Iterate over all resources
7: **if** COUNTRESOURCES($sched_W, m$) $\leq N$ **then** ▷ Less than N instances used
8: $T_{Rank_i} \leftarrow \max_{A_p \in pred(Rank_i)} \left\{T_{A_p} + t_{(Rank_i, R_j)}\right\}$ ▷ Compute completion of $Rank_i$
9: **else**
10: $T_{Rank_i} \leftarrow \infty$ ▷ Mark schedule as non-valid
11: **end if**
12: **if** $T_{Rank_i} < T_{\min}$ **then** ▷ Save the minimum completion time
13: $T_{\min} \leftarrow T_{Rank_i}$
14: $R_{\min} \leftarrow R_j$
15: **end if**
16: **end for**
17: $sched_W \leftarrow sched_W \cup (Rank_i, R_{\min})$ ▷ Schedule the task
18: **end for**
19: **return** $sched_W$
20: **end function**

sources (line 7). If not, its makespan is computed as in HEFT (line 8), otherwise the makespan of that solution is set to infinite (line 10). This way, non-valid schedules will never verify the condition in line 12 and will be discarded in favour of the ones requiring less than N simultaneous instances.

The pseudocode of the Cloud-aware version of MOHEFT is described in Algorithm 4. The only difference to the original version is in lines 11–14. As in the Cloud-aware HEFT, the constraint on the number of resources is checked in line 11. If the constraint is not violated, the makespan and cost are computed as before, otherwise they are set to infinite. This will cause the algorithm to discard that solutions later on in line 18, producing only tradeoff solutions which consider at most N instances simultaneously.

6.4 MOHEFT Complexity

Given a set of n activities and m resources, the computational complexity of HEFT is $O(n \cdot m)$. MOHEFT only introduces two main differences with respect to HEFT: the creation of several solutions in each iteration of the algorithm, and the possibility of considering resources providing a tradeoff solution. These two modifications only require an additional loop in MOHEFT (see Algorithm 2, lines $9-11$). Considering that the set of tradeoff solutions is K, the extra loop in MOHEFT performs only K iterations, rendering a complexity of $O(n \cdot m \cdot K)$. Usually, the number of tradeoff solutions is a constant much lower than n and m. For example, a workflow can be composed of thousands of tasks and the set of tradeoff solutions can be accurately represented

Algorithm 4 Cloud-aware MOHEFT algorithm.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: N ▷ Maximum simultaneous instances
Require: I ▷ Number of different instance types
Require: $R = \bigcup_{j=1}^{m} R_j$ ▷ Set of resources, where $m = N \cdot I$
Require: K ▷ Number of tradeoff solutions
Ensure: $S = \bigcup_{i=1}^{K} sched_W, sched_W = \{(A_i, sched\,(A_i))\,|\forall A_i \in A\}$ ▷ Set of K tradeoff schedules
1: **function** MOHEFT(W,R,K)
2: $Rank \leftarrow$ B-RANK(A) ▷ Order the tasks according to B-rank
3: **for** $k \leftarrow 1, K$ **do** ▷ Create K empty workflow schedules
4: $S_k \leftarrow \emptyset$
5: **end for**
6: **for** $i \leftarrow 1, n$ **do** ▷ Iterate over the ranked tasks
7: $S' \leftarrow \emptyset$
8: **for** $j \leftarrow 1, m$ **do** ▷ Iterate over all resources
9: **for** $k \leftarrow 1, K$ **do** ▷ Iterate over all tradeoff schedules
10: $s \leftarrow S_k \cup (Rank_i, R_j)$ ▷ Extend all intermediate schedules
11: **if** COUNTRESOURCES($sched_W, m$) $> N$ **then** ▷ More than N instances used
12: $T_s \leftarrow \infty$ ▷ Mark schedule as non-valid
13: $C_s \leftarrow \infty$
14: **end if**
15: $S' \leftarrow S' \cup \{s\}$ ▷ Add new mapping to all intermediate schedules
16: **end for**
17: **end for**
18: $S' \leftarrow$ SORTCROWDDIST(S', K) ▷ Sort according to crowding distance
19: $S \leftarrow$ FIRST(S', K) ▷ Choose K schedules with highest crowding distance
20: **end for**
21: **return** S
22: **end function**

with tens of solutions. Thus, the complexity can be approximated as almost $O(n \cdot m)$, as in HEFT. In the case of the Cloud extensions, the only difference is on the number of resources; this way, the complexity stays the same as in the original algorithms.

7 The SPEA2* Algorithm

In this section we introduce SPEA2*, the multi-objective workflow scheduler algorithm MOHEFT and HEFT are compared with in this work.

SPEA2* [25] is a version of SPEA2, a genetic algorithm proposed in [27]. SPEA2* was shown to outperform NSGA-II and PAES for multi-objective workflow scheduling in [25]. This algorithm works with a population (set) P of candidate solutions which are iteratively recombined with the aim of evolving them towards the optima. The pseudo-code of SPEA2* is presented in Algorithm 5.

The algorithm requires the same input parameters as the cloud version of MOHEFT (see Section 6.3). The number K of desired tradeoff solutions is used by SPEA2* as the population size. At the beginning, the method is initialised with a nearly-optimal solution in terms of makespan (computed using the cloud version of HEFT) and in terms of economic cost (computed with a heuristic aimed at optimising cost) in lines 1 and 2. The rest of the population, up to K solutions, is filled with randomly generated schedules (lines 3 to 5). After that, the main loop of the algorithm is run until a

Algorithm 5 Pseudocode of SPEA2.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: N ▷ Maximum simultaneous instances
Require: I ▷ Number of different instance types
Require: $R = \bigcup_{j=1}^{m} R_j$ ▷ Set of resources, where $m = N \cdot I$
Require: K ▷ Number of tradeoff solutions
1: $P \leftarrow$ heft(W,N,I,R)
2: $P \leftarrow P; \cup$ **heftc**(W,N,I,R) ▷ heftc is a list-heuristic for optimising cost. See [25]
3: **while** $|P| < K$ **do**
4: $P \leftarrow P \cup$ **generateRandomSchedule**() ▷ Includes a random scheduler in the population P
5: **end while**
6: **while not terminationCondition**() **do**
7: $parents \leftarrow$ **binaryTournament**(P)
8: $P' \leftarrow$ **crossover**($parents$)
9: **mutate**(P')
10: **evaluate**(P')
11: **evaluateConstraints**(P')
12: $R \leftarrow P \cup P'$
13: $P \leftarrow$ truncate(R,K)
14: **end while return** P

termination condition is met (lines 6 to 14). Here, we consider as termination condition to execute this loop for a maximum of 1000 iterations (ten times more than the iterations done in [25]).

The work carried out within the main loop is the same as in the original SPEA2 evolutionary algorithm. Firstly, a set of good solutions are selected from P for generating new solutions (line 7); this set is usually referred in the context of evolutionary algorithms as *parents*. Immediately after, the solutions within the parents set are recombined using a stochastic recombination operator (line 8), which depends on how the solutions to the problem are codified. In this work, each schedule is codified with two strings: the task-assignment string, which indicates on which resource is executed each tasks; and, the scheduling-order string, which indicates the order in which tasks are executed. As recombination operator we use a two-points crossover, which combines parts of the tasks-assignment strings of two parents. Further details about the codification and operator are included in [25] and have been omitted here for brevity. As a result of applying recombination to the parents, a new set of solutions P' is generated. All the solutions in this set are later on modified by means of another operator, called mutation (line 9); the goal of this operator is to promote the exploration of a wider area of the search space. In particular, SPEA2* uses two types of mutation operators: reordering mutation, and replacing mutation; the former is aimed at changing the execution order of the tasks and the latter to re-allocate tasks into different resources (see [25]). The schedules in P' are then evaluated in order to compute their makespan and cost (line 10). Once evaluated, P' and P are merged (line 12), and the best K schedules out of $P' \cup P$ become the population P for the next iteration of the loop 13. This selection is called in SPEA2 truncation and the idea is to prioritize non-dominated schedules and schedules that increase the diversity of the Pareto set (see [27] for additional details).

We slightly modified SPEA2* in order to deal with the limitation imposed by commercial Clouds on the maximum number of simultaneous resources. As with HEFT and MOHEFT, we consider that the set of resources consists of $m = N \cdot I$ instances ($20 \cdot 5$ in our particular example). We enhanced the algorithm with a mechanism for dealing with constrains, similar to the one proposed in [7]. This mechanism always compares first two solutions on the basis of their constraint violation. Solutions which do not violate the constraint are preferred to solutions which violate it (i.e. schedules using 20 or less instances simultaneously are preferred to schedules using more than 20 instances). If both solutions violate the constraint, the one violating it in a lesser extent is preferred (i.e. a solution using 23 machines simultaneously is preferred to a schedule that uses 30). Solutions which do not violate the constraint or violate it to the same extent are compared using the Pareto dominance relationship described in Section 5. In Algorithm 5, constraints are computed in line 11. By using this mechanism, only the solutions which less violate the constraints survive to the next generation. It is worth mentioning that a scheme as the one used by MOHEFT is not possible for SPEA2*. Evolutionary algorithms work with a population of fixed size (in our case K). Discarding solutions which violate the constraints would mean that the population may decrease in size, thus breaking the evolutionary cycle of the algorithm.

We implemented SPEA2* using the jMetal framework [9]. In our experiments, we used $K = 10$. The recombination operator is applied with a probability of 0.9 and the mutation with 0.5. This configuration is the same one used in the original paper where SPEA2* is described.

According to [27], SPEA2 has a complexity of at least $O(2K^2 log 2K)$. SPEA2* requires to execute HEFT in the first place in order to compute a good solution in terms of makespan. As HEFT has the same complexity as MOHEFT (see Section 6.4), we can affirm that SPEA2* has at least the same complexity as MOHEFT in the best case.

8 Experimental Setup

We describe in this section the experiments carried out for validating the Cloud-aware MOHEFT algorithm. First, we summarise the comparison criteria for assessing the quality of our approach. Finally, we describe the different types of workflows and the Amazon EC2 infrastructure considered in the experiments.

8.1 Evaluation Metrics

We consider three criteria for comparing the quality of solutions computed by MOHEFT, HEFT and SPEA2*. First, we consider the shortest makespan of the schedules computed by the three analysed techniques. Second, we focus on the economic aspect of the schedules, analysing the cheapest solution reported

by each technique. The idea of these two indicators is to assess the behaviour of the different approaches optimising each individual criterion. Finally, we consider the hypervolume indicator described in Section 5 for assessing the quality of computed tradeoff solutions.

We also analyse the tradeoff solutions computed by MOHEFT and SPEA2* for different workflow types. The idea is to study the balance between the two conflicting objectives, makespan and cost, and how much can be gained in one objective by deteriorating the other. For this analysis we rely on a graphical representation of the solutions computed by the two algorithms. The presented graphs start with the most makespan-efficient schedule and continue along the Pareto-front towards the cheapest solution. Therefore, although we compute the tradeoff between cost and makespan, for the sake of highlighting the potential of the obtained results, we will plot the cost savings versus the makespan deterioration, as percentages relative to the most makespan-efficient solution, computed by HEFT.

Finally, we pay attention at the number and the type of instances selected by the different scheduling solutions computed by the three approaches.

It is worth mentioning that MOHEFT and HEFT are non-stochastic algorithms, and thus, the same solution is computed in different runs. This claim is not valid for SPEA2*, which may compute different fronts in different runs of the algorithm. In order to avoid our conclusions be biased by any hazard effect of this stochastic behaviour, we run SPEA2* for five times and always consider the run producing the front with the largest hypervolume.

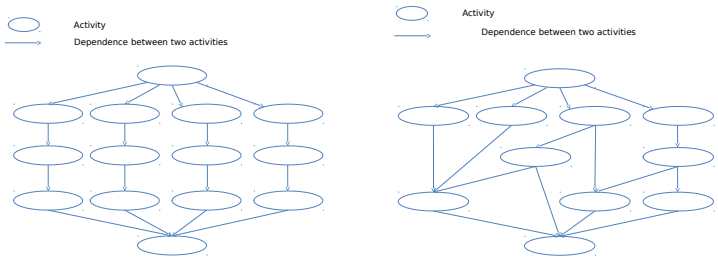
8.2 Workflow Applications

We consider in our evaluation two kinds of workflows: *synthetic* with different types of artificially-generated properties, and *real-world* coming from our real-world collaborations with domain scientists in the Austrian Grid.

8.2.1 Synthetic Workflows

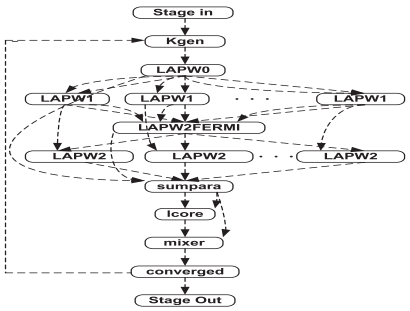
We generated three types of synthetic workflows using a random workflow generator as the one described in [24] (see Fig. 2a and Fig. 2b). Our interest is to analyse how the number of independent activities influences the scheduling results. Therefore, the defined types are intended to cover a wide spectrum of workflow structures from this point of view:

- *Type-1* where the number of tasks that can be executed in parallel ranges between one and two;
- *Type-2* where the number of tasks that can be executed in parallel is high, and the workflow is balanced (same number of tasks in every level);
- *Type-3* where the number of tasks that can be executed in parallel is high, but the workflow is unbalanced (different number of tasks in every level).

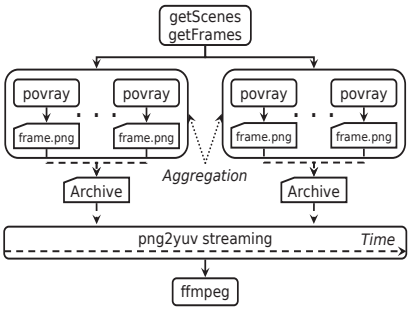


(a) Synthetic workflow (balanced).

(b) Synthetic workflow (unbalanced).



(c) WIEN2k.



(d) POV-Ray.

Fig. 2: Evaluated workflows applications.

We generated the length of every task and the data produced using a Gaussian distribution. For every type, we considered 100 different instances having between 100 and 1000 different tasks.

8.2.2 Real-world Workflow Applications

Alongside the synthetic workflows, we consider two real-world applications for validating our work: WIEN2k and POV-Ray.

WIEN2k [3] is a material science workflow for performing electronic structure calculations of solids using density functional theory based on the full-

potential (linearized) augmented plane-wave ((L)APW) and local orbital (lo) method. WIEN2k is a workflow of *Type-2* consisting of two parallel sections with sequential synchronisation activities in between (see Fig. 2c).

Persistence Of Vision Raytracer (POV-Ray) [16] is a free tool for creating three-dimensional graphics, which is known to be a time consuming process used not only by hobbyists and artists, but also in biochemistry research, medicine, architecture and mathematical visualisation. We modelled a POV-Ray rendering scenario as a *Type-2* workflow depicted in Fig. 2d, where the description of a movie can be separated in several scenes, each scene being composed of several frames which can be rendered as parallel activities. Finally, all frames are merged into a `mpeg` movie.

8.3 Resource Infrastructure

We consider that the user has access to the default maximum number of $N = 20$ Amazon instances which can be of any of the five types summarised in Table 1 (i.e. $I = 5$ and $m = N \cdot I = 20 \cdot 5 = 100$). We assume that no public IP addresses are required for running the experiments on the Amazon EC2 infrastructure. Additionally, the output data transfers from Amazon to the outside Internet are constant, take place only at the end of the workflow execution and thus, do not influence the scheduling results. In this situation, we assume in our experiments that the prices for data sent and received are zero: $PI_{R_i} = 0$ and $PO_{R_i} = 0$.

9 Evaluation

We present in this section the evaluation results for the synthetic workflow first, then for the real-world ones. Finally, we analyse how the solutions computed by the algorithms change when the constraint of simultaneously using 20 resources is relaxed.

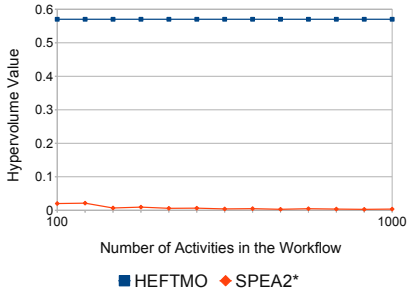
9.1 Synthetic Workflows

We analyse in this section the results for the three types of synthetic workflows.

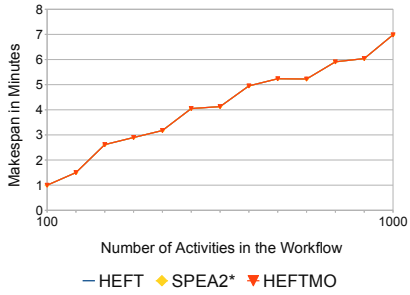
9.1.1 Type-1 *Workflows*

We start by analysing the results for the three comparison metrics, makespan, economic cost, and hypervolume, summarised in Fig. 3.

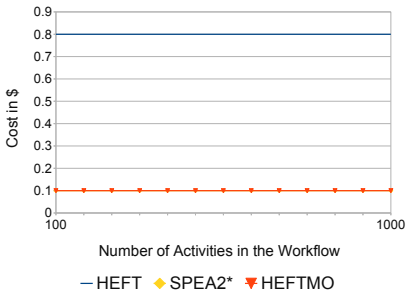
First, Fig. 3a shows that MOHEFT outperformed SPEA2 in terms of hypervolume for all evaluated instances. We did not include HEFT in this comparison because it only delivers a single solution with the optimal makespan. It is remarkable that, for this workflow type, MOHEFT always computed solutions with the same hypervolume value, meaning that the shape of the optimal



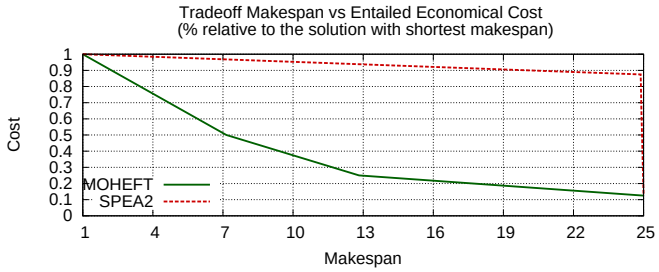
(a) Hypervolume.



(b) Makespan.



(c) Economic cost.



(d) Tradeoff analysis cost vs makespan.

Fig. 3: Evaluation results for synthetic workflows of *Type-1*.

set of tradeoff solutions in this case does not vary with the workflow size. In terms of makespan (see Fig. 3b), all the three methods computed the same solution, which confirms that the performance of MOHEFT does not degrade compared to HEFT. In case of SPEA2*, the results are not surprising since the algorithm is initialised with the solution computed by HEFT. Both MOHEFT and SPEA2 computed the same cheapest schedule illustrated in Fig. 3c, which considers the cheapest instance (`m1.small`) for the entire workflow. This fact is a consequence of the low degree of parallelism of this workflow. The solution computed by HEFT is always the most expensive one.

Fig. 3d shows an example of tradeoff solutions computed by MOHEFT and SPEA2*. The higher quality of the solutions computed by MOHEFT can be easily visualised in this chart. In particular, we observe that our method computed a schedule which halves the price of the solution with the optimal makespan by only introducing a 7% of time overhead. In case of SPEA2*, a solution with the same cost would have required an increase of 25% in makespan. These results highlight the importance of the Pareto front as a decision support tool, since computing a single schedule at a time would have hidden this information.

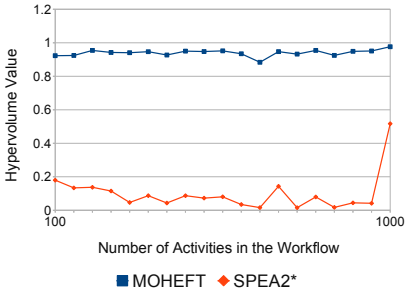
9.1.2 Type-2 *Workflows*

The hypervolume, shortest makespan and cheapest schedule computed for this workflow type are summarised in Fig. 4.

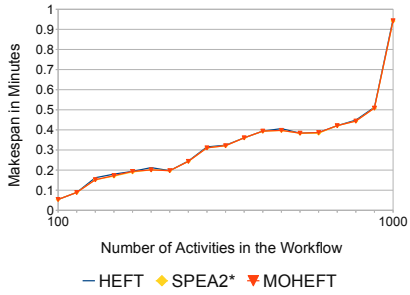
In terms of the quality of the set of tradeoff solutions, MOHEFT has again outperformed SPEA2* for all workflow sizes, as indicated by the hypervolume indicator in Fig. 4a. In this case, different workflow sizes result in Pareto fronts with different hypervolumes, meaning that the shape of the Pareto front for this problem depends on the number of tasks that can be executed in parallel. If we focus on the makespan (see Fig. 4b), it is worth mentioning that MOHEFT and SPEA2* were able in some cases to compute solutions with better makespans than HEFT. The explanation for this behaviour is that, due to its greedy nature, HEFT easily converges towards a local optimum, situation which is overcome by MOHEFT and SPEA2* due to a larger exploration of the search space. In terms of economic cost (see Fig. 4c), MOHEFT and SPEA2* computed the same solution which is a lot cheaper than the solution with the best makespan.

Fig. 4e shows a comparison of the tradeoff solutions computed by MOHEFT and SPEA2*. The differences between both algorithms are even more noticeable than for workflows of *Type-1*. In this case, MOHEFT computed a schedule which reduced the cost by 30% incurring only a 1.4% increase in makespan. Computing a solution of similar price for SPEA2* would have required increasing the makespan by more than 450%. This huge difference between our approach and SPEA2* clearly points out the potential of MOHEFT for multi-objective workflow scheduling in terms of the quality of the computed solutions.

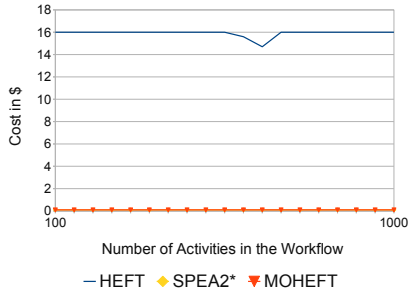
For this workflow type, many solutions computed by SPEA2* required more than 20 resources, thus invalidating its adoption for workflow scheduling in the context of commercial Clouds with limitations on the maximum number of instances that can be simultaneously rented. In particular, 66% of the computed schedules required more than the 20 resource limit imposed by Amazon EC2 (see Fig. 4e). This behaviour does not appear in the solutions computed by MOHEFT or HEFT, which always provided schedules with at most 20 resources.



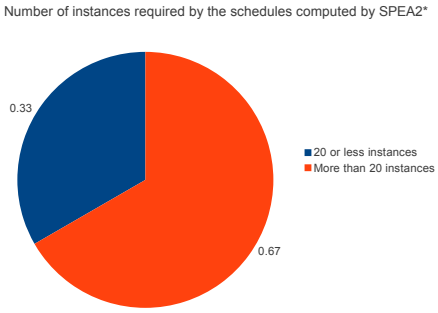
(a) Hypervolume.



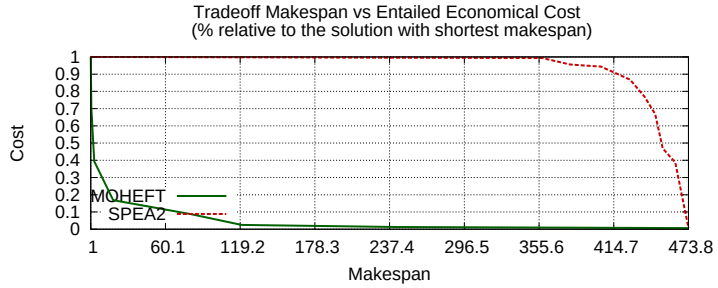
(b) Makespan.



(c) Economic cost.



(d) SPEA2* solutions with more than 20 instances.



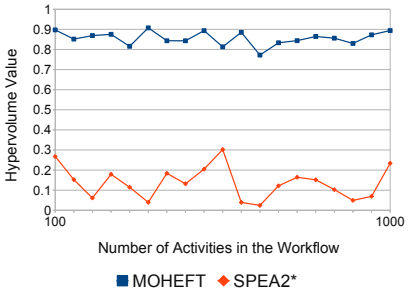
(e) Tradeoff analysis cost vs makespan.

Fig. 4: Evaluation results for synthetic workflows of *Type-2*.

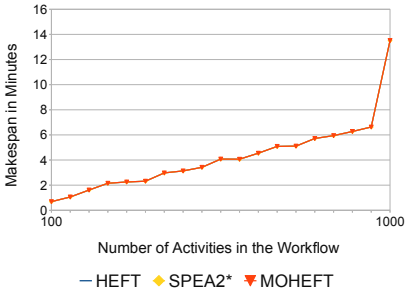
9.1.3 Type-3 *Workflows*

The results for this workflow type, summarised in Fig. 5, confirm the findings of the previous two types in terms of the hypervolume, makespan and cost.

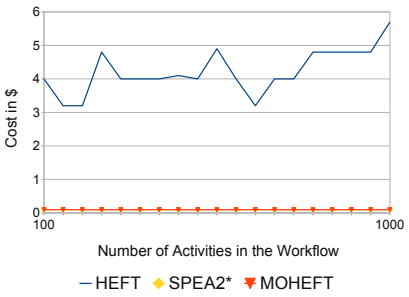
The hypervolume of the tradeoff sets for this workflow type (see Fig. 5a) shows that MOHEFT outperforms SPEA2* also in this case. As in the previous case, the hypervolume reflects a different shape of the Pareto front for workflows with different number of activities. This fact validates the hypothesis that the shape of the tradeoff solutions depends on the number of activities



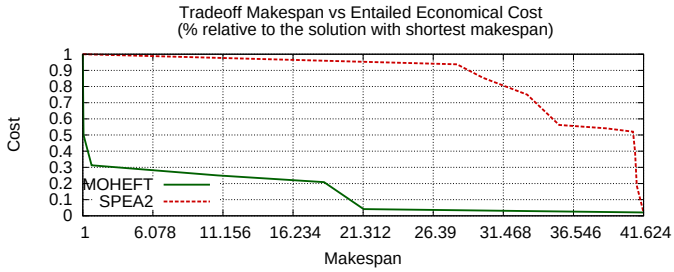
(a) Hypervolume.



(b) Makespan.



(c) Economic cost.



(d) Tradeoff analysis cost vs makespan.

Fig. 5: Evaluation results for synthetic workflows of *Type-3*.

of the workflow that can be executed in parallel. All three techniques computed the schedule which minimises the makespan, confirming again the suitability of this method for workflow scheduling if the user is only interested in optimising this goal. Similar to the previous case, MOHEFT and SPEA2* computed the best solutions in terms of economic cost. The difference between HEFT and the other two methods tends to increase with the number of activities composing the workflow.

An example of the tradeoff solutions computed by MOHEFT and SPEA2* is shown in Fig. 5d. For this type of workflow, MOHEFT was able to compute solutions halving the maximum price with only 1% increase in makespan,

while SPEA2* required at least a 40% of extra time for a solution of the same cost. These results point out once more the better suitability of MOHEFT for multi-objective workflow scheduling on the Amazon EC2 Cloud. In this case, the three techniques always computed schedules meeting the restriction of using at most 20 on-demand instances.

9.2 Real-World Workflows

In this section we complete our evaluation by applying our method on two real-world workflows: WIEN2k and POV-Ray. In both cases, the number of tasks ranges between 100 and 1000.

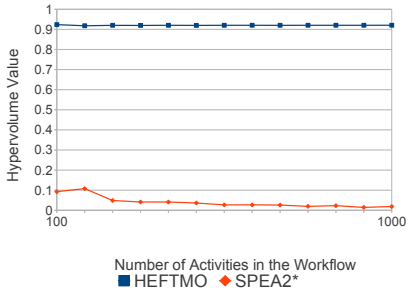
9.2.1 WIEN2k

The results for the WIEN2k workflow are depicted in Fig. 6.

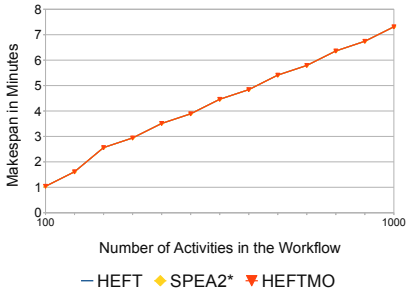
First, Fig. 6a shows that, similar as for the synthetic workflows, MOHEFT always outperformed SPEA2* in terms of the hypervolume. Moreover, the difference in this metric is even higher than in any of the previously evaluated cases. One possible explanation for this behaviour is related to the higher difficulty of solving this workflow application. Our hypothesis is that SPEA2* got stuck in some areas of the search space, thus requiring a prohibitively large number of evaluations for increasing the quality of the computed results. The results for makespan and economic cost displayed in Fig. 6b and 6c verify the results obtained for the synthetic workflows. In terms of makespan, all three algorithms performed equally well and computed the solution with the lowest makespan. In terms of economic cost, MOHEFT and SPEA2* always computed the solution with the lowest cost, while HEFT delivered solutions three to six times more expensive.

An example comparing the tradeoff solutions delivered by MOHEFT and SPEA2* is illustrated in Fig. 6e. The results are similar to the ones obtained for the *Type-2* workflows, where huge economic savings are obtained by MOHEFT with only a small increase in makespan. In this case, SPEA2* requires a huge loss in performance for obtaining similar cost results. For example, MOHEFT computed a solution which halves the price of the schedule with the shortest makespan and experienced only a 5% of time deterioration. Meanwhile, a 1% of cost saving in SPEA2* would have required an increase of 250% in makespan. In 23.3% of the cases, SPEA2* computed schedules requiring more than 20 resources, while all solutions computed by MOHEFT met this constraint.

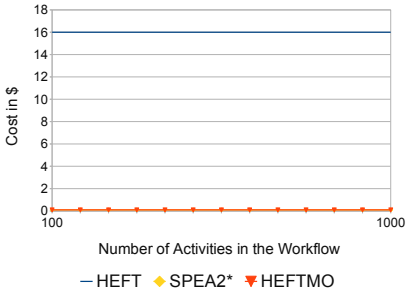
Analysing the solutions computed by MOHEFT, we observe in Fig. 6d that the mostly preferred instance was `c1.large` offering the highest performance per $ ratio (see Table 1). This instance was selected for scheduling a task in 40% of the cases. The second most preferred machine is `m1.small`. Although the performance per $ of this instance is similar to `c1.medium`, MOHEFT preferred it as the cheapest instance. The instances with the poorest ratio,



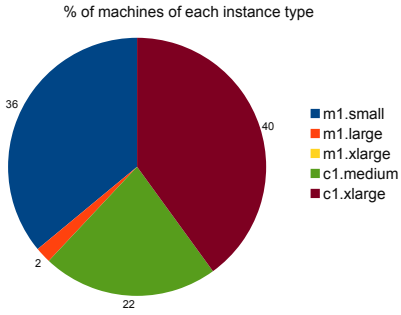
(a) Hypervolume.



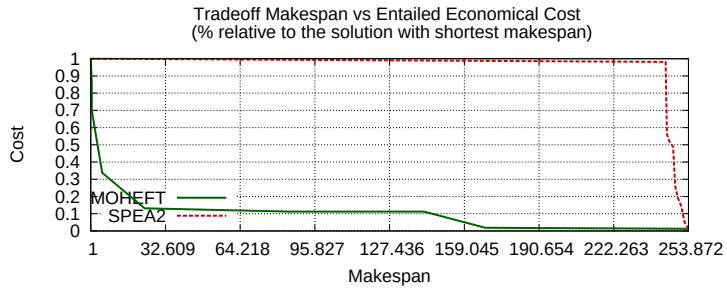
(b) Makespan.



(c) Economic cost.



(d) Used instances.



(e) Tradeoff analysis cost vs makespan.

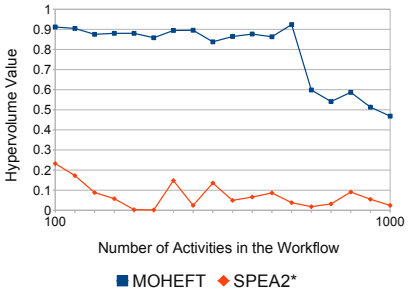
Fig. 6: Evaluation results for the WIEN2k workflow.

`m1.large` and `c1.medium`, were barely considered by MOHEFT and do not even appear in the graph (1% percent and 0%, respectively).

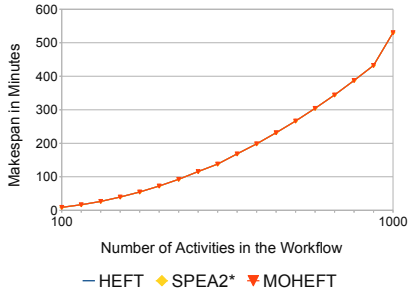
9.2.2 POV-Ray

The results for the POV-Ray workflow summarised in Fig. 7 are similar to the WIEN2k experiment.

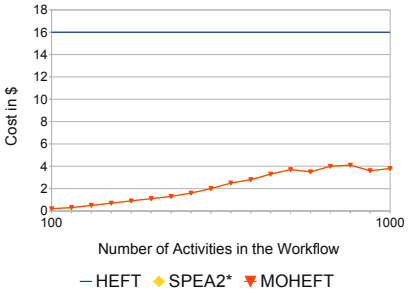
Fig. 7a shows the hypervolume of the sets of tradeoff solutions. Also in this case MOHEFT outperformed SPEA2* for all the evaluated workflow sizes. For this application, the higher the number of tasks in the workflow is, the harder



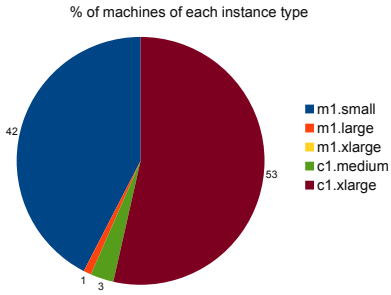
(a) Hypervolume.



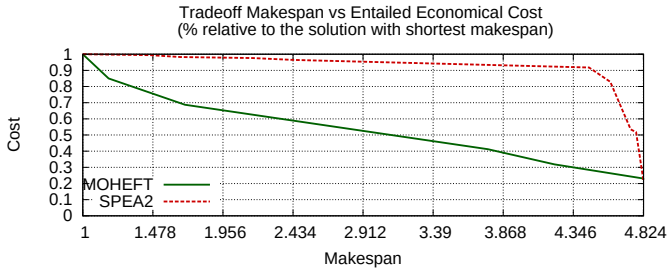
(b) Makespan.



(c) Economic cost.



(d) Used instances.



(e) Tradeoff analysis cost vs makespan.

Fig. 7: Evaluation results for the POV-Ray workflow.

is for MOHEFT to compute a set of tradeoff solutions with high quality. This result can be visualized in the hypervolume that decreases with the number of tasks. This behavior is not that obvious for SPEA2*, however, the quality of the computed fronts are of poorer quality than the ones of MOHEFT, as reflected by the low values of the indicator. As in the previous experiments, all the algorithms computed the same solution with the shortest makespan (see Fig. 7b), while MOHEFT and SPEA2* computed the same cheapest schedule too (see Fig. 7c).

An example of the tradeoff solutions computed by MOHEFT and SPEA2* for this application is shown in Fig. 7e. We observe a small difference in

makespan of only 4.8% between the cheapest and more expensive solutions. In this case, while MOHEFT found a solution halving the price for the sake of only 3% increase in makespan, SPEA2* required a 4.7% increase. Nevertheless, the small difference between both techniques has carefully interpreted, since in 40% of the cases SPEA2* has been unable to produce a workflow schedule using 20 or less resources.

Finally, we observe in Fig. 7d that MOHEFT mainly considered two instances with the best performance per $ ratio: `m1.small` and `c1.xlarge`, while the other types of instances were selected only in a marginal number of schedules. Similar to WIEN2k, the `m1.xlarge` instance was not considered again in any schedule, which demonstrates its unsuitability for running our computational-intensive scientific workflows.

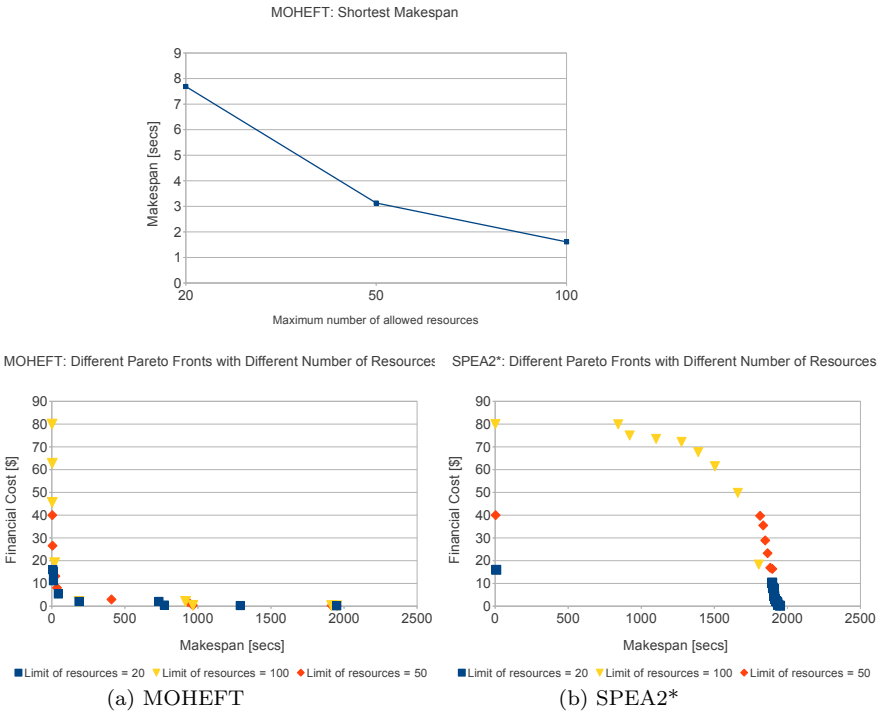
9.3 Analysis of the Impact of the Number of Resources

In this section we analyse the implications of relaxing the constraint of simultaneously using a maximum of $N = 20$ resources. Our idea is to study whether or not the algorithms scale with the number of resources and how these extra resources affect the computed tradeoff solutions. In this experiment we focused on the instance of the WIEN2k workflow with the largest number of activities evaluated in Section 9.2.1.

Figures 8 and 9 summarise the obtained results when the limitation on the number of simultaneous resources is set to 20, 50, and 100. More specifically, Fig. 8 shows how the shortest computed makespan is affected by considering these number of resources. It is worth mentioning that in this case the three algorithms always compute the same solution with the shortest makespan. We see in this figure that the evaluated algorithms scale with the number of resources, computing schedules with shorter makespan as long as the number of resources increases. In all the cases, these computed schedules make use of the maximum allowed number of resources.

Fig. 9 shows the Pareto fronts computed by MOHEFT and SPEA2*. The idea of this figure is to show how the different tradeoff solutions are affected by having different number of resources. As HEFT only computes the solution with the shortest makespan, we do not consider it in this comparison. In the case of MOHEFT (Fig. 9a) the computed Pareto fronts are similar in all the cases and the only differences appear regarding to the solutions optimizing makespan; in these cases, having a higher number of resources implies more expensive but shorter schedules (left part of the showed fronts). In the case of schedules minimising the financial cost, the computed solutions are not affected by having the possibility of using simultaneously a higher number of resources. The tradeoff solutions computed by SPEA2* are plotted in Fig. 9b. It is possible to see that the solutions computed by SPEA2* are still dominated by the solutions computed by MOHEFT. Increasing the number of resources means for SPEA2* to compute always shorter but more expensive schedules. A deeper analysis of the compute solutions reveals that some of the schedules

Fig. 8: Shortest makespan computed with different constraints in the number of resources.



(a) MOHEFT (b) SPEA2*

Fig. 9: Evaluation of the Pareto fronts computed with different constraints on the number of resources.

computed by SPEA2* still violate the constraint regarding to the maximum number of allowed simultaneous resources; however, the number of schedules doing so decreases as long as the constraint is relaxed.

10 Conclusions and Future Work

In this paper we proposed and analysed a truly multi-objective workflow scheduler for commercial IaaS Clouds such as Amazon EC2. Our new method called MOHEFT extends a well-known list scheduling heuristic to provide scientists with a set of optimal solutions offering different tradeoffs between makespan and economic cost for workflow executions. We extended the method to deal with the realistic constraints imposed today by commercial Clouds that restrict the total number of resources that can be simultaneously acquired, but keep their type flexible depending on the temporal needs.

We showed the potential of the Pareto front as a tool for decision support in selecting the most appropriate tradeoff solutions. In particular, the visuali-

sation of the Pareto front for some workflow types revealed that one can obtain solutions with a marginal 5% makespan increase by investing half of the money in renting Cloud instances. We validated and compared MOHEFT with HEFT, the most popular workflow scheduling algorithm, and SPEA2*, an extension of the state-of-the-art multi-objective optimisation algorithm SPEA2, using a combination of synthetic and real-world workflows. In all experiments, MOHEFT computed schedules with the same makespan as the HEFT but with better economic cost. MOHEFT also outperformed SPEA2* in terms of hypervolume used as an indicator of the quality of the set of tradeoff solutions. A visual analysis of the tradeoff solutions revealed that SPEA2* computed in many cases solutions involving a higher economic cost than MOHEFT for the same makespan. Finally, our experiments revealed that MOHEFT was able to meet the constraints imposed by current commercial Clouds in terms of the maximum amount of instances, while SPEA2* failed on this issue.

In future work we intend to evaluate MOHEFT for other objective functions such as security issues, reliability of spot instances or energy consumption. We will consider increasing the number of objectives to three and more, and extend the resource model to comprise federated Cloud infrastructures with different limitations on the number of resources that can be simultaneously rented.

References

1. Alexandru, I., Ostermann, S., Yigitbasi, M., Prodan, R., Fahringer, T., Epema, D.: Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Transactions onf Parallel and Distributed Systems pp. 1–16 (2010)
2. Assayad, I., Girault, A., Kalla, H.: A bi-criteria scheduling heuristics for distributed embedded systems under reliability and real-time constraints. In: International Conference on Dependable Systems and Networks, DSN'04. IEEE, Firenze, Italy (2003)
3. Blaha, P., Schwarz, K., Madsen, G., Kvasnicka, D., Luitz, J.: Wien2k: An augmented plane wave plus local orbitals program for calculating crystal properties. Tech. rep., Institute of Physical and Theoretical Chemistry, TU Vienna (2001)
4. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and c onceptual comparison. ACM Computing Surveys **35**(3), 268 – 308 (2003)
5. Canon, L.C., Emmanuel: Mo-greedy: an extended beam-search approach for solving a multi-criteria scheduling problem on heterogeneous machines. International Heterogeneity in Computing (2011)
6. Coello, C.A.C., Lamont, G.B., Van Veldhuisen, D.A.: Evolutionary algorithms for solving multi-objective problems. Springer (2007)
7. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast elitist multi-objective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**, 182–197 (2000)
8. Durillo, J., Fard, H., Prodan, R.: Moheft: A multi-objective lilst-based method for workflow scheduling. In: 4th IEEE International Conference on Cloud Computing Technology and Science (2012)
9. Durillo, J.J., Nebro, A.J.: jmetal: A java framework for multi-objective optimization. Advances in Engineering Software **42**, 760–771 (2011)
10. Fard, H., Prodan, R., Barrionuevo, J., Fahringer, T.: A multi-objective approach for workflow scheduling in heterogeneous environments. In: Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, pp. 300–309 (2012). DOI 10.1109/CCGrid.2012.114
11. Garg, R., Singh, A.K.: Reference point based multi-objective optimization to workflow grid scheduling. Int. J. Appl. Evol. Comput. **3**(1), 80–99 (2012)

12. Garg, S.K., Buyya, R., Siegel, H.J.: Scheduling parallel applications on utility grids: time and cost trade-off management. In: Proceedings of the Thirty-Second Australasian Conference on Computer Science - Volume 91, ACSC '09, pp. 151–160. Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2009)
13. Hakem, M., Butelle, F.: Reliability and scheduling on systems subject to failures. In: Proceedings of the 2007 International Conference on Parallel Processing, ICPP '07, pp. 38–. IEEE Computer Society, Washington, DC, USA (2007)
14. Ilavarsan, E., Thambidurai, P.: Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. Journal of Computer Science **3**(2), 94–103 (2007)
15. Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y., albi, E.G.T., A.Y.Zomaya, Tuyttens, D.: A parallel bi-objective hybrid metaheuristic for energy-aware s cheduling for cloud computing systems. Journal of Parallel and Distributed Computing (71), 1497–1508 (2011)
16. Plachetka, T.: POVRAY – Persistence of Vision Parallel Raytracer. In: Proceedings of Computer Graphics International '98, pp. 123–129 (1998)
17. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.D.: Scheduling workflows with budget constraints. In: in Integrated Research in Grid Computing, S. Gorlatch and M. Danelutto, Eds.: CoreGrid series. Springer-Verlag (2007)
18. Singh, D., Garg, R.: A robust multi-objective optimization to workflow scheduling for dynamic grid. In: Proceedings of the International Conference on Advances in Computing and Artificial Intelligence, ACAI '11, pp. 183–188. ACM, New York, NY, USA (2011)
19. Singh, M.P., Vouk, M.A.: Scientific Workflows: Scientific Computing Meets Transactional Workflows (1996)
20. Talukder, A.K.M.K.A., Kirley, M., Buyya, R.: Multiobjective differential evolution for scheduling workflow applications on global grids. Evolution **21**(13), 1742–1756 (2009)
21. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M.: Workflows for e-Science: Scientific Workflows for Grids. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
22. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. Parallel and Distributed Systems, IEEE Transactions on **13**(3), 260 –274 (2002)
23. Ullman, J.: Np-complete scheduling problems. Journal of Computer and System sciences **10**(3), 384–393 (1975)
24. Yu, J., Buyya, R., Ramamohanarao, K.: Workflow scheduling algorithms for grid computing. In: F. Xhafa, A. Abraham (eds.) Metaheuristics for Scheduling in Distributed Computing Environments, pp. 109–153. Springer Berlin (2008)
25. Yu, J., Kirley, M., Buyya, R.: Multi-objective planning for workflow execution on grids. In: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, GRID '07, pp. 10–17. IEEE Computer Society, Washington, DC, USA (2007)
26. Zeng, J.t., Xia, J.w., Li, J.z., Li, M.h.: Multi-objective optimal grid workflow scheduling with qos constraints. In: B. Cao, T.F. Li, C.Y. Zhang (eds.) Fuzzy Information and Engineering Volume 2, *Advances in Intelligent and Soft Computing*, vol. 62, pp. 839–847. Springer Berlin Heidelberg (2009)
27. Zitzler, E., Laumanns, M., Thiele, L.: Spea2: Improving the strength pareto evolutionary algorithm. Tech. Rep. 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland (2001)