DOI: https://doi.org/10.1016/j.simpat.2015.07.001

Pareto Tradeoff Scheduling of Workflows on Federated Commercial Clouds

Juan J. Durillo and Radu Prodan[a], Jorge G. Barbosa[b]

[a]*Institute of Computer Science, University of Innsbruck, Austria*
[b]*Departamento de Engenharia Informática, Universidade do Porto, Portugal*

Abstract

As distributed computing infrastructures become nowadays ever more complex and heterogeneous, scientists are confronted with multiple competing goals such as makespan in high-performance computing and economic cost in Clouds. Existing approaches typically aim at finding a single tradeoff solution by aggregating or constraining the objectives in an a-priory fashion, which negatively impacts the quality of the solutions. In contrast, Pareto-based approaches aiming to approximate the complete set of (nearly-) optimal tradeoff solutions have been scarcely studied. In this paper, we extend the popular Heterogeneous Earliest Finish Time (HEFT) workflow scheduling heuristic for dealing with multiple conflicting objectives and approximating the Pareto frontier optimal schedules. We evaluate our new algorithm for performance and cost tradeoff optimisation of synthetic and real-world applications in Distributed Computing Infrastructures (DCIs) and federated Clouds and compare it with a state-of-the-art meta-heuristic from the multi-objective optimisation community.

Keywords: Scheduling, Cloud computing, scientific workflows, multi-objective optimisation, Pareto frontier, HEFT

1. Introduction

Scientific workflows are a popular way of modelling applications to be executed on distributed computing infrastructures (DCI), such as Clouds. In this context, one of the most challenging tasks in the workflow lifecycle is how to schedule its tasks onto the heterogeneous available resources. Traditionally

in parallel and distributed systems, workflow scheduling has been targeted to optimise the performance, measured in terms of makespan or the completion time of all tasks [1, 2, 3], which has been shown to be NP-complete.

In the context of Cloud computing, the user needs to care about an additional metric represented by the cost incurred by renting resources. Most of the commercial Clouds offer different types of resources at different prices. For example, in Amazon EC2[1] a user can choose among four different types of resources, where the fastest is eight times more expensive than the slowest. Federated Clouds bring new opportunities when using commercial Clouds, as different providers may offer resources with different performance and pricing models. In such situations, a customer may be interested in scheduling low-priority tasks on the slow resources offered by a cheap provider, and high-priority critical tasks on the expensive fast resources offered by a high-performance provider. Scheduling a workflow application becomes therefore a *multi-objective optimisation* problem with at least two in-conflict criteria, *makespan* and *financial cost*, to which no single solution exists, but a set of tradeoff solutions called *Pareto front*. Moreover, federated Clouds pose additional drawbacks limiting the range of applications that can benefit from them. For example, resources belonging to different providers may be located in different areas connected via best-effort Internet, which is particularly problematic in the context of data-intensive applications. In addition, companies may impose restrictions over sensitive data that must stay within the frontiers of a single institution, limiting optimisation opportunities. In this situations, it is not clear whether federated Clouds are an appealing alternative for workflow applications.

In most related works, workflow scheduling optimising several competing objectives has been simplified either to a single-objective problem either by aggregating all the objectives in a single analytical function or by constraining the others. The main drawback of these approaches is that the aggregation and constraining are applied a-priori, with a complete knowledge about the workflow, infrastructure, 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 that approximate the Pareto set of tradeoff solutions have been proposed. Computing the complete Pareto front provides the user with a set of optimal solutions to select

[1] `http://aws.amazon.com/ec2/pricing/`

based on personal preferences and requirements, and may reveal solutions impossible to see beforehand.

In this paper, we extend the popular Heterogeneous Earliest Finish Time (HEFT) workflow scheduling algorithm for dealing with multiple conflicting objectives and approximating the Pareto frontier of tradeoff schedules on DCIs. Our new algorithm called *Multi-Objective HEFT (MOHEFT)* is able to deal with an arbitrary number of objectives, instantiated in this paper by makespan and financial cost. We tackle the problem from the point of view of a broker coordinating the services of two federated Clouds with different performance and pricing models: Amazon EC2 and GoGrid. For this purpose, we customise the algorithm to deal with the peculiarities encountered today in public commercial Clouds, such as limited number of simultaneous resources and hourly billing intervals. We analyse the performance and cost tradeoffs for scheduling workflows with different shapes (degree of parallelism), number of activities, and computation and communication requirements on DCIs and federated Clouds by comparing MOHEFT with a genetic state-of-the-art meta-heuristic from the multi-objective optimisation theory called SPEA2* [4].

The paper is organised as follows. In the next section we describe the related work. Section 3 gives a short background on multi-criteria optimisation required for a better understanding. Section 4 defines the abstract workflow, resource, and scheduling models underneath our approach. Section 5 presents the MOHEFT algorithm as an extension to the original HEFT heuristic, including a customisation for commercial Cloud infrastructures. We describe the experimental setup for validating our algorithm in Section 6, followed by the results in Section 7. Section 8 concludes the paper.

2. Related Work

The most common technique combines the multiple objectives in a single objective, for example by assigning different weights to the different objectives and optimising the resulting aggregation function. The main differences in these approaches is the way in which preferences are expressed. For example, [5] combines reliability (in terms of resource failures) and makespan using a weight vector. A similar approach has been used in [6] and [7], also for optimising reliability and time. A general disadvantage of these approaches is that the computed solution depends on the combination of the multiple objectives, which is made a-priori and without a complete information about

the problem being solved. This fact implies that, if the tradeoff or aggregation function does not capture the user preferences in an accurate way, the computed solution may not be satisfactory for the solved problem. Additionally, the objective functions require to be normalised to the same interval in order to properly capture these preferences.

Another way of combining the different objectives in a single objective function is by imposing user-defined constraints to each objective function (i.e. a desired value for each function to be optimised). Once the constraints have been set, the idea is to compute a solution optimised for one objective. After computing this solution, several modifications are made with the aim of improving according to another objective, as long as the constrains are not violated. An example work using this approach has been proposed in [8, 9] for optimising makespan and economical cost in utility Grids. The main weakness of this approach is that the number of objectives that can be optimised is limited to a few. Furthermore, it is required to established an order on which objective to optimise first, hence including some sort of preference information. Finally, reasonable a-priori values for the constraints are often unknown until the first schedule is computed.

More recently, several approaches computing the complete set of tradeoff solutions emerged grouped in two main lines: (1) genetic algorithms-based techniques for optimising makespan and cost [4], makespan and energy consumption [10], or makespan, cost and reliability [11]; and (2) list-based heuristics for optimising makespan and cost or makespan [12, 13] and energy consumption [14]. Only few of these works targeted workflwow scheduling on Clouds and none of them considers Cloud federations.

While most Cloud-related research deals with the placement of virtual machines onto physical machines [15, 16, 17, 18], scheduling of tasks has been scarcely studied on the context of a Cloud federation. In this sense, [19] analyses and proposes several heuristics for task scheduling onto resources belonging to the same provider but geographically located in different areas. Though not purely a federation of Clouds, the authors address a similar problem to the one arising in a federated system. The authors in [20] present a semantic architecture to build schedulers for federated Clouds with emphasis on the system architecture with no special focus to the optimality of the scheduler. None of these approaches consider a purely multi-objective formulation of the problem.

3. Multi-Objective Optimisation Background

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

A general multi-objective optimisation problem aims to find a vector $\vec{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]$ which satisfies the m inequality constraints $g_i(\vec{x}) \geq 0, i = 1, 2, \ldots, m$, the p equality constraints $h_i(\vec{x}) = 0, i = 1, 2, \ldots, p$, and minimises the vector function $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T$, where $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables.

Taking into account this definition, a solution $x^1 = [x_1^1, x_2^1, \ldots, x_n^1]$ is said to *dominate* a solution $x^2 = [x_1^2, x_2^2, \ldots, x_n^2]$ if and only if $f_i(x^1) \leq f_i(x^2)$ for $i = 1, 2, \ldots, m$, and there exist at least one j $(1 \leq j \leq m)$ such that $f_i(x^1) < f_i(x^2)$. Conversely, two points are said to be non-dominated whenever none of them dominates the other. Figure 1 depicts some examples of dominated and non-dominated solutions. Assuming that f_1 is makespan and f_2, the cost, a dominates b because $f_1(a) < f_1(b)$ and $f_2(a) < f_2(b)$. Similarly, a dominates c too. Meanwhile, a and d are non-dominated because a is better than d in makespan, but d is better in financial cost. A set of non-dominated solutions is called *Pareto set* of tradeoff solutions. The value of the solutions in the Pareto set is known as *Pareto frontier*.

A Pareto frontier represents a powerful tool for decision support and preferences discovery. For example, the shape of the frontier can be used to provide insight to decision makers such as scientists or simple operators, allowing them 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 frontier is known. This way, the users does not have to set their preferences before finding a solution, instead the preferences are discovered afterwards. A high quality Pareto frontier needs to fulfill two properties, accuracy and diversity, by uniformly covering all the possible ranges of optimal solutions, as close as possible to the optimal ones. A metric of measuring the quality of a set of tradeoff solutions X is the *hypervolume* $HV(X)$ representing the area enclosed between the points in X and a reference point W (see Figure 1), usually selected as the nadir or anti-utopia point (e.g. with the maximum makespan and cost). This way, the better and the more diverse the points in X are, the higher $HV(X)$ gets. In

Figure 1 for example, the set containing the solid round points is better than the set containing the squared solutions because the area enclosed within the dashed lines is larger than the one represented by the sold lines. A method to ensure the diversity of the solutions is to maximise the *crowding distance* defined in [21] and graphically depicted in Figure 1, which gives a measure of the area surrounding a solution where no other tradeoff solution is placed.

4. Model

In this section, we formally describe the workflow scheduling problem on a DCI and the two metrics of interest.

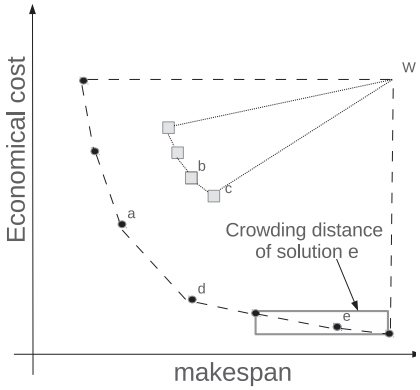


Figure 1: Pareto frontier, HV and crowding distance.

4.1. Workflow model

A *workflow application* is usually modelled as a directed acyclic graph (DAG), $W = (A, D)$ consisting of n activities (or tasks) $A = \cup_{i=1}^{n} \{A_i\}$, interconnected through a set of control flow and data flow dependencies $D = \{Ai, Aj, Data_{ij}\} \,|\, (A_i, A_j) \in A \times A$, where $Data_{ij}$ represents the size of the data which needs to be transferred from activity A_i to activity A_j. We use $pred(A_j)$ to denote the *predecessor* set of activity A_j, (i.e. those activities which should be finished before executing A_j) where $A_i \in pred\,(A_j)$. 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

Let us assume that our hardware platform consists of a set of m heterogeneous resources $R = \cup_{j=1}^{m} R_j$. For every resource R_i we assume that we know its speed s measured in the number of instructions per second. Additionally, every resource has a price model consisting of four components:

1. pe_{R_i} is the price of using the resource per second;
2. ps_{R_i} is the price for storing the data on that resource in MB per second;

3. pi_{R_i} is the price of receiving data in terms of incoming MB;
4. po_{R_i} is the price of sending data in terms of outgoing MB.

Without loss of generality, this pricing model can be adapted to match any of the existing commercial Clouds.

4.3. Problem Definition

The problem here consists in scheduling the execution of the workflow activities in the resources in such a way that the makespan and economical costs are minimised. In the rest of this work, we use $sched(A_i)$ to denote the resource in which activity A_i is plan to be executed.

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 $A_p \in pred(A_i)$ and the time required to execute A_i in R_j:

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

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 has been 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. We can compute the completion time T_{A_i} of activity A_i considering the execution time of itself and its predecessors as:

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

The workflow makespan is the maximum completion time of all its activities.

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

4.3.2. Financial Cost

We define the financial cost $cost_{(A_i,R_j)}$ of executing an activity A_i on a resource R_j as the sum of the four cost components introduced in Section 4.2:

$$\begin{aligned} C_{(A_i,R_j)} = \; & t_{(A_i,R_j)} \cdot pe_{R_j} + Data\,(A_i) \cdot t_{(A_i,R_j)} \cdot ps_{R_i} + \\ & In(A_i) \cdot pi_{R_i} + Out\,(A_i) \cdot po_{R_i}, \end{aligned} \tag{4}$$

where $Data\,(A_i)$ represents the total data storage required for executing the activity A_i (including input, output, and temporary data) and $In\,(A_i)$ / $Out\,(A_i)$ is the sum of the data sizes transferred to / from A_i from / to activities executed on resource other than R_j. The financial cost of executing the entire workflow W is the simple sum of the costs of all its activities:

$$C_W = \sum_{i=1}^{n} C_{(A_i, sched(A_i))} \tag{5}$$

5. MOHEFT: Multi-Objective Heterogenous Earliest Finish Time

In this section we describe our proposed multi-objective scheduling algorithm based on extending an existing list scheduling algorithm for computing a set of tradeoff solutions instead of a single one. For a better understanding, we start by describing the mono-objective version of the algorithm and extend it afterwards for dealing with multiples objectives.

5.1. HEFT: Heterogeneous Earliest Finish Time Algorithm

The Heterogeneous Earliest Finish Time Algorithm (HEFT) is a popular list-based heuristic scheduling algorithm for optimizing the makespan [1] 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 schedule first the critical path tasks from the set of ready tasks. Further details about how to sort the tasks can be found in [1]. 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).

5.2. MOHEFT: Multi-Objective Heterogenous Earliest Finish Time

As described before, HEFT builds a solution by sequentially mapping 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

Algorithm 1 HEFT algorithm.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: $R = \bigcup_{i=1}^{m} R_i$ ▷ Set of resources
Ensure: $sched_W = \{(A_i, sched\,(A_i)) \,|\forall A_i \in A\}$ ▷ Workflow schedule

```
1: function HEFT(W,R)
2:     Ranking ← B-RANK(A)                          ▷ Order the tasks according to B-rank
3:     sched_W ← ∅                                  ▷ Initialize workflow schedule with empty set
4:     for i ← 1, n do                              ▷ Iterate over the ranked tasks
5:         T_min ← ∞
6:         for j ← 1, m do                          ▷ Iterate over all resources
7:             T_Ranking_i ← max_{A_p ∈ pred(Ranking_i)} { T_{A_p} + t_(Ranking_i, R_j) }   ▷ Compute completion of Ranking_i
8:             if T_Ranking_i < T_min then          ▷ Save the minimum completion time
9:                 T_min ← T_Ranking_i
10:                R_min ← R_j
11:            end if
12:        end for
13:        sched_W ← sched_W ∪ (Ranking_i, R_min)    ▷ Schedule the task
14:    end for
15: return sched_W
16: end function
```

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 desired size of the set of tradeoff solutions K. Our method also ranks the tasks using the B-rank (line 2). After that, 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 6–15). MOHEFT iterates first over the list of tasks (line 6) sorted by execution order. The idea is to extend every solution in S by mapping the next task to be executed onto all possible resources creating m new solutions. We store all the created solutions 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 10). This strategy results in an exhaustive search if we do not include any restrictions, therefore MOHEFT only saves the best K tradeoffs solutions from the temporary set S' to the set S (lines 13–14). We consider that a solution belongs to the best tradeoff if it is not dominated

Algorithm 2 MOHEFT algorithm.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: $R = \bigcup_{i=1}^{m} R_i$ ▷ Set of resources
Require: K ▷ Number of workflow tradeoff schedules
Ensure: $S = \bigcup_{i=1}^{K} sched_W, sched_W = \{(A_i, sched\,(A_i))\,|\forall A_i \in A\}$ ▷ Pareto set of tradeoff schedules

```
1: function MOHEFT(W,R,K)
2:     Ranking ← B-RANK(A)                          ▷ Order the tasks according to B-rank
3:     for k ← 1, K do                              ▷ Create K empty workflow schedules
4:         S_k ← ∅
5:     end for
6:     for i ← 1, n do                              ▷ Iterate over the ranked tasks
7:         S' ← ∅
8:         for j ← 1, m do                          ▷ Iterate over all resources
9:             for k ← 1, K do                      ▷ Iterate over all tradeoff schedules
10:                S' ← S' ∪ {S_k ∪ (Ranking_i, R_j)}   ▷ Add new mapping to all intermediate schedules
11:            end for
12:        end for
13:        S' ← SORTCROWDDIST(S', K)                ▷ Sort according to crowding distance
14:        S ← FIRST(S', K)                         ▷ Choose K schedules with highest crowding distance
15:    end for
16: return S
17: end function
```

by any other solution and if it contributes to the diversity of the set. Our criterion is to prefer solutions with a higher crowding distance (defined in Section 3), which ensures that the selected Pareto set represents a wider area of different tradeoff solutions. After assigning all the tasks (line 16), the algorithm returns the set of K best tradeoff solutions.

Complexity. Given a set of n activities and m resources, the computational complexity of HEFT's mapping phase is $O(n \cdot m)$. MOHEFT introduces two extensions with respect to HEFT: the creation of several solutions in parallel, and the possibility of considering resources providing a tradeoff solution. We implemented these capabilities by introducing a new loop that iterates over the set tradeoff solutions (line 9). Thus, considering that the set of tradeoff solutions is M, the complexity of MOHEFT becomes $O(n \cdot m \cdot M)$. 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 a set of tradeoff solutions can be accurately represented with tens of solutions. Thus, the complexity could be considered as almost $O(n \cdot m)$, as in HEFT.

5.3. Multi-Objective Scheduling on Federated Clouds

In this section, we extend our multi-objective scheduling approach to deal with the characteristics of federated Cloud environments. For this purpose,

we first redefine the financial cost metric to fit the cost models applied today by commercial Clouds, and then customise the MOHEFT algorithm to deal with their resource provisioning peculiarities.

5.4. Cloud Financial Cost

The financial cost for executing an activity in a commercial Cloud depends on two terms: the computation cost $C^{(comp)}$ and the cost of data transfer and storage $C^{(data)}$. We assume that all providers charge the customers using an hourly based model (i.e. per hour of computation). 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:

$$C^{(data)}_{(A_i,R_j)} = Data\,(A_i) \cdot t_{(A_i,R_j)} \cdot ps_{R_i} + In(A_i) \cdot pi_{R_i} + Out\,(A_i) \cdot po_{R_i}, \quad (6)$$

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. We consider without loss of generality 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 $t^{(end)}_{A_i}$.

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* $G^{(j)}_k$, $1 \leq k \leq q$. 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 $G^{(j)}_1 = \{J_1, \ldots, J_r\}, r \leq p$ following three rules:

1. The first activity J_1 belongs to the first group: $J_1 \in G^{(j)}_1$;
2. Every activity $J_i \in G^{(j)}_1, 2 \leq i \leq r$, starts before the current leased hour expires and before the machine is released:

$$t^{(start)}_{J_i} < t^{(start)}_{J_1} + \left\lceil \frac{t^{(end)}_{J_{i-1}} - t^{(start)}_{J_1}}{3600} \right\rceil \cdot 3600. \quad (7)$$

   We convert the total time of using a resource to hours by dividing it by 3600 and using the ceiling operator. This equation guarantees a contiguous resource allocation of activities within one hour slot;

3. The next activity not in the first group $J_{r+1} \notin G_1^{(j)}, r+1 \leq p$, starts after the last hour of computation elapses and the resource is released:

$$t_{J_1}^{(start)} + \left\lceil \frac{t_{J_r}^{(end)} - t_{J_1}^{(start)}}{3600} \right\rceil \cdot 3600 < t_{J_{r+1}}^{(start)}. \tag{8}$$

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 J_{r+1} instead of J_1, and similarly for the rest of the groups. Once all groups have been created, we define the cost $C_{R_j}^{(comp)}$ of using the resource R_j as the time in hours for executing all groups multiplied by the hourly cost:

$$C_{R_j}^{(comp)} = pe_{R_j} \cdot \sum_{k=1}^{q} \left\lceil \frac{\sum_{A_i \in G_k^{(j)}} t_{(A_i,R_j)}}{3600} \right\rceil. \tag{9}$$

The cost of executing the workflow $W = (A, D)$ is the sum of the cost of all m the used resources and the cost for transferring and storing the data:

$$C_W = \sum_{j=1}^{m} C_{R_j}^{(comp)} + \sum_{(A_i,A_j,Data_{ij}) \in D} C_{(A_i,R_j)}^{(data)}. \tag{10}$$

5.5. Federated MOHEFT Cloud Algorithm

Algorithm 3 requires as input the instance types offered by all Cloud providers, the maximum number of resources that can be simultaneously rented from each provider $\vec{N}$, and the number of tradeoff solutions K.

Firstly, the algorithm ranks the tasks in the workflow using the B-rank metric and creates a set S of K empty schedules (lines 2 and 3). Afterwards, it iterates over the list of tasks and extends every solution in S by mapping the next task onto different possible instances. For every task, the algorithm builds a list of possible resources where the task can be executed, either by reusing an instance already assigned to a previous task, or by acquiring a new instance (lines 8 and 12, respectively). This list is used for building new schedules that also consider the current task (line 16). The newly produced schedules are stored in a temporary set S', initially empty (line 21). After each iteration, S' replaces S before the next task in the list is considered (line 24). Obviously, this strategy results in an exhaustive search if we do not include any restrictions. To avoid this, we save only the best K tradeoff solutions from the temporary set S' to the set S, selected based on the

Algorithm 3 Federated MOHEFT Cloud algorithm.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: $\vec{N} = (N_1, ..., N_c)$ ▷ Maximum instances allowed by each of the c Cloud providers
Require: $I = \bigcup_{i=1}^{m} I_i$ ▷ Instance types offered by all c Cloud providers
Require: K ▷ Number of workflow tradeoff schedules
Ensure: $S = \bigcup_{i=1}^{K} sched_W, sched_W = \{(A_i, sched\,(A_i))\,|\forall A_i \in A\}$ ▷ Pareto set of tradeoff schedules

1: **function** MOHEFT($W, \vec{N}, I, K$)
2: $Ranking \leftarrow$ B-RANK(A) ▷ Order tasks according to the B-rank
3: $S \leftarrow \{S_1, \ldots, S_K\}$, where $S_k = \emptyset, \forall k \in [1, K]$ ▷ Create K empty workflow schedules
4: **for** $i \leftarrow 1, n$ **do** ▷ Iterate over the ordered tasks in the *Ranking* list
5: $S' \leftarrow \emptyset$
6: **for** $k \leftarrow 1, K$ **do** ▷ Iterate over all tradeoff schedules
7: $R \leftarrow \emptyset$ ▷ Build a set of possible instances for executing next task
8: **for** $r \leftarrow 1, |S_k|$ **do** ▷ Reuse instance where tasks in S_k are executing
9: $(A', R') = S_{kr}$
10: $R \leftarrow R \cup R'$
11: **end for**
12: **for** $r \leftarrow 1, m$ **do** ▷ Consider a new instance of each type
13: $R \leftarrow R \cup I_r$
14: **end for**
15: **for** $j \leftarrow 1, |R|$ **do** ▷ Iterate over all resources
16: $s \leftarrow S_k \cup (Ranking_i, R_j)$ ▷ Extend all intermediate schedules
17: **if** VIOLATIONCONSTRAINTS($s, \vec{N}$) **then** ▷ Check if too many instances from a provider
18: $T_s \leftarrow \infty$ ▷ Mark schedule as non-valid
19: $C_s \leftarrow \infty$
20: **end if**
21: $S' \leftarrow S' \cup \{s\}$ ▷ Add new mapping to intermediate schedules
22: **end for**
23: **end for**
24: $S' \leftarrow$ SORTCROWDDIST(S', K) ▷ Sort according to the crowding distance
25: $S \leftarrow$ FIRST(S', K) ▷ Choose K schedules with the highest crowding distance
26: **end for**
27: **return** S
28: **end function**

objective functions and the diversity of the set, i.e., how different these solutions are (see [13]). To deal with the restriction on the maximum number of instances that can be simultaneously rented from a provider, we discard any schedule that violates this constraint (line 17) by setting its financial costs and makespan to infinite. This way, the partial solution will be always worse (dominated) than any other schedule and will be discarded in line 24.

6. Experimental Setup

We evaluated the MOHEFT algorithm in different configurations involving synthetic and real-world workflows with different properties on synthetic DCIs and federated commercial public Clouds.

6.1. Synthetic Workflows

We started our evaluation on synthetic workflows of three types, generated as described in [4, 22] (see Figure 2):

- *Type-1* with maximum two parallel activities;
- *Type-2* where the number of parallel activities is high and the workflow is balanced (same number of activities in every level), with many independent activities sharing one successor and one predecessor;
- *Type-3* where the number of parallel activities is high, but the workflow is unbalanced (the number of activities in every level is different) and most tasks have different successors and predecessors.

We generated workflows with a number of activities between 100 and 1000. We generated the length of each activity using a Gaussian distribution with the mean execution time of 10 seconds on an average single core instance. For the experiments on federated Clouds, we considered three activity classes with different data requirements: (1) *low* where each activity produces/consumes around 10MB of data, (2) *medium* producing/consuming around 100MB, and (3) *high* producing/consuming around 1GB.

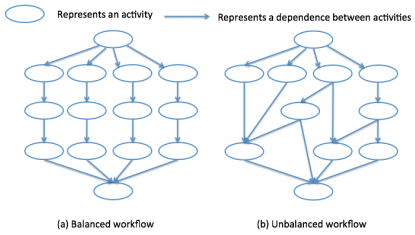


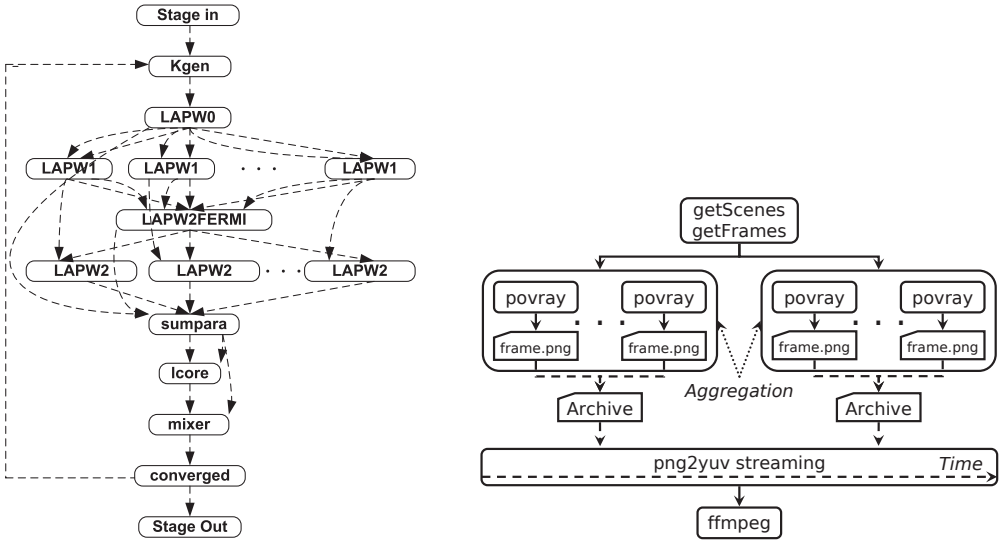
Figure 2: Balanced and unbalanced workflow examples.

6.2. Synthetic DCIs

We simulated two synthetic resource infrastructures: (1) *DCI-1*, where the number of resources ranges between 10 and 100; (2) *DCI-2*, where the number of resources are similar to the number of tasks in the workflow. We generated the speed and price of each resource using a random uniform distribution limited within a maximum and minimum speed and cost limits.

6.3. Real-World Applications

We complement our evaluation on synthetic workflows with two real applications. Both applications can be modelled with different parallelization sizes involving different number of activities. In this work we considered 100 different instances, each of them having between 100 and 1500 activities.



(a) WIEN2k. (b) POV-Ray.

Figure 3: Real-world workflow applications.

WIEN2k [23] is a material science workflow for performing electronic structure calculations of solids using density functional theory based on the full-potential (linearised) 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 Figure 3a).

The Persistence Of Vision Raytracer (POV-Ray) (`http://www.povray.org`) is a free tool for creating time consuming three-dimensional graphics 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 Figure 3b, where the description of a movie is separated in several scenes, each scene being composed of several frames which can be rendered as parallel activities. Finally, all frames are merged into an `mpeg` movie. The data volume consumed by this application also depends on the number of frames which have to be transferred to the merger activity after being rendered. We consider here three instances of this application rendering 512, 1024, and 4096 frames.

6.4. Federated Cloud

With respect to the real federated infrastructure, we simulated two commercial Cloud providers: Amazon EC2 and GoGrid. We use for modelling

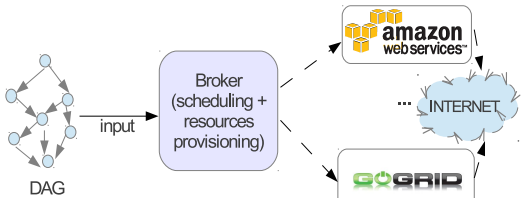


Figure 4: Federated Cloud infrastructure.

Table 1: Amazon EC2 and GoGrid instances.

Instance	*Speed [GFLOPS]*	*Price [$/h]*	*GFLOPS/ $*
Amazon EC2			
`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
GoGrid			
`GG.large`	8.8	0.16	46.4
`GG.xlarge`	28.1	0.76	37.0

the workflows' makespan the average performance in millions of floating point operations per second (GFLOPs) of five different instance types of Amazon EC2 and two instances of GoGrid, as reported in [24] (see Table 1). We assume that the resources of the same provider are connected using a local gigabit network and the different providers are connected using a 150 MB per second wide area network. We consider that a user can simultaneously rent 20 instances from both Amazon EC2 and GoGrid, which can be of any of the types summarised in Table 1. Although these limits can be usually extended upon request, we intend in this paper to evaluate how MOHEFT can deal with the default configurations.

6.5. Evaluation metrics

We evaluated the results of the MOHEFT algorithm in terms of three different metrics: shortest makespan, lowest financial cost, and hypervolume as a measure for the quality of tradeoff solutions (see Section 3). We compared our results MOHEFT with the original HEFT and SPEA2, a genetic algorithm from the multi-objective optimisation theory proposed by Zitzler et al. [25]. SPEA2 starts with a population of candidate solutions and iteratively recombines them with the aim of producing better ones. We used a custom version of the algorithm called SPEA2* [4] initialised with a nearly-optimal solution in terms of makespan (computed using HEFT) and financial cost (by using the cheapest resources). In our work, We implemented SPEA2* in the jMetal framework [26], initialised it with a population size equal to the number of tradeoff solutions M, and run it for 1000 generations. In all the cases, we set the size of the Pareto set of tradeoff solutions to $M = 10$.

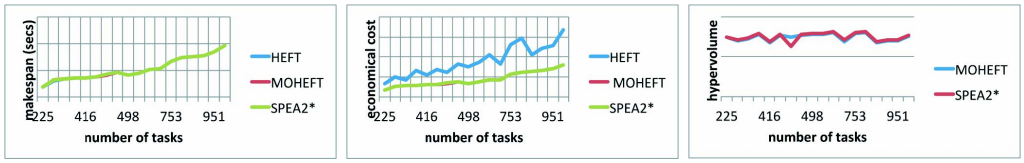


Figure 5: Synthetic *Type-1* workflows with low parallelism.

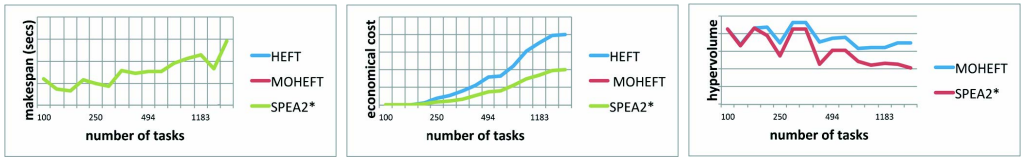


Figure 6: Synthetic *Type-2* workflows with high parallelism and balanced structure.

7. Experimental Results

We present the experimental results organised in four categories covering the four combinations of synthetic and real-world workflows and DCIs.

7.0.1. Synthetic Workflows on Synthetic DCIs

We used for these experiments the *DCI-2* resource infrastructure.

Type-1 *workflows.* For *Type-1* workflows, we observe in Figure 5 that all three methods computed solutions with the same makespan, while HEFT delivered worse financial costs than MOHEFT and SPEA2*. While the result for SPEA2* is not surprising since the algorithm is initialised with HEFT schedules in terms of makespan and the cheapest in terms of cost, it demonstrate that MOHEFT does not degrade its performance either. We further observe that MOHEFT and SPEA2* computed solutions with similar HV which indicates a similar quality of solutions.

Type-2 *workflows.* The makespan and financial costs for *Type-2* workflows are similar to the previous case as displayed in Figure 6. In terms of quality of the Pareto set, MOHEFT outperforms SPEA2* in all the evaluated vases. In particular, the higher the number of activities in the workflow is, the higher is the difference in HV between the two techniques. This means that for a high number of parallel activities, MOHEFT is able of producing better tradeoff solutions than SPEA2* genetic operators.

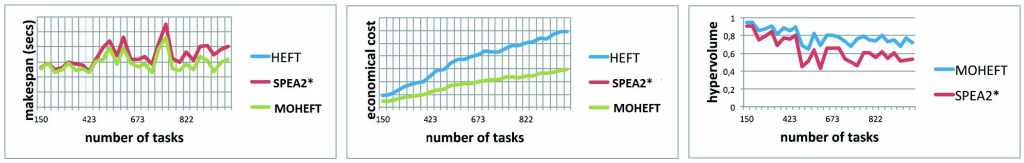
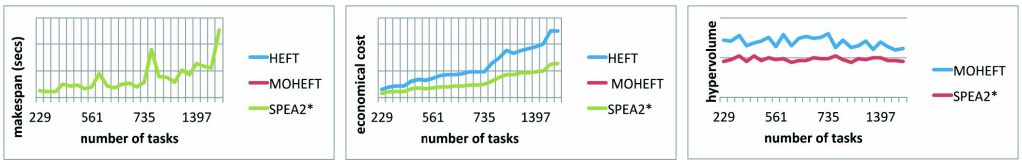
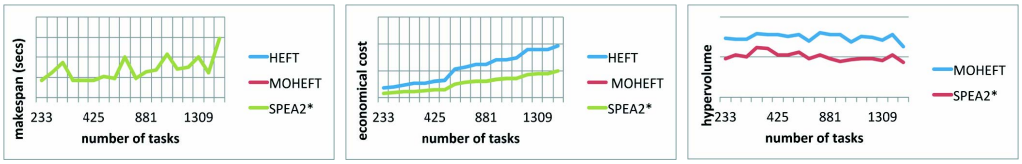


Figure 7: Synthetic *Type-3* workflows with high parallelism and unbalanced structure.



(a) *DCI-1* infrastructure.



(b) *DCI-2* infrastructure.

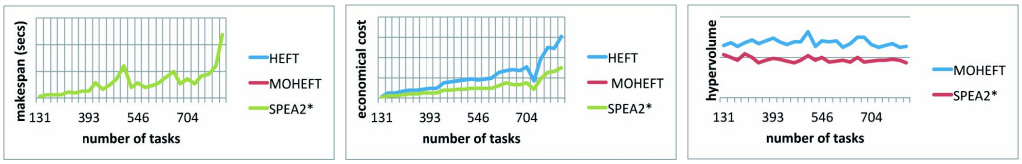
Figure 8: WIEN2k workflow results.

Type-3 *workflows.* For the *Type-3* workflows, MOHEFT outperformed the results of HEFT and SPEA2* in terms of makespan, as observed in Figure 7. The explanation is that HEFT is a greedy heuristic and, hence, selects in every iteration the resource that minimises the makespan of the next task in the list. On the contrary, MOHEFT builds several solutions in parallel achieving a better exploration of the search space. In terms of cost, MOHEFT and SPEA2* computed the best solutions as in the previous case, and the improvements to HEFT are higher when the number of activities increases. Finally, the HV metric indicates that MOHEFT outperforms again SPEA2* in terms of quality of solutions. Similar to *Type-2* workflows, the difference between the two methods increases with the number of activities.

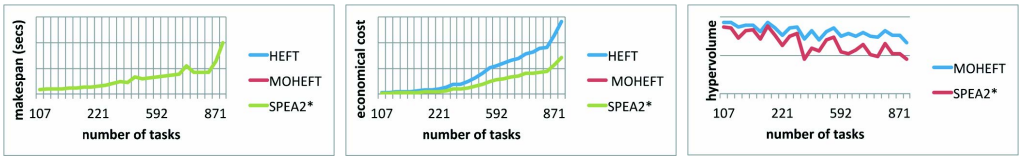
7.1. Real-World Applications on Synthetic DCIs

In this section, we extend our validation with two real-world workflow applications: WIEN2k and POV-Ray.

WIEN2k. The results for the WIEN2k workflow are depicted in Figure 8 for *DCI-1* and *DCI-2* resource configurations. In both cases, MOHEFT



(a) *DCI-1* infrastructure.



(b) *DCI-2* infrastructure.

Figure 9: POV-Ray workflow results.

and SPEA* were able to compute a solution with the same makespan as HEFT. While this the result is not surprising for SPEA* since the algorithm is initialised in part with solutions delivered by HEFT, it demonstrates that MOHEFT does not degrade the HEFT performance either. In terms of financial cost, MOHEFT and SPEA2* clearly outperform HEFT. The differences cost are also small for small workflows, but it increased with the number of activities. The results of SPEA2* are similar to MOHEFT, which is again no surprise since SPEA2* is initialised in part with the cheapest scheduled. Finally, we observe that the quality of the Pareto set computed by MOHEFT in terms of HV is always better than the one computed by SPEA*.

POV-Ray. The results for the POV-Ray workflow summarised in Figure 9 are similar to the ones reported for WIEN2k. First, the three methods compute again the same solution for the best makespan. Second, MOHEFT and SPEA2* provided the best results in terms of the financial cost, HEFT being again the worst alternative. As before, the difference in cost between MOHEFT and HEFT increases with the number of activities. Finally, the analysis of the *HV* verifies that MOHEFT outperformed SPEA2* also in this case. In the *DCI-2* configuration, we further observe that the difference in the quality of the Pareto

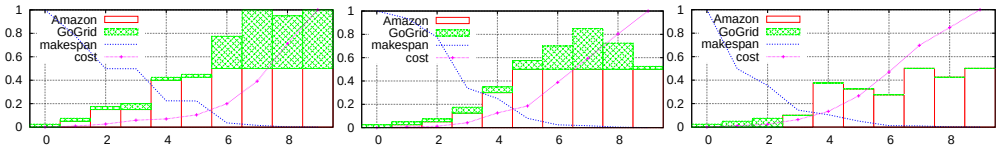
Figure 10: POV-Ray Pareto sets.

set increases with the number of activities. An example of the set of tradeoff solutions computed by the two methods for the same workflow instance is depicted in Figure 10, which demonstrates that the scheduled computed by MOHEFT dominate those computed by SPEA2*.

7.2. Synthetic Workflows on Federated Clouds

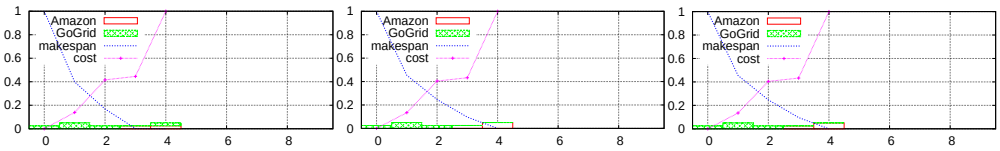
In this section, we analyse the schedules computed by MOHEFT for the *Type-2* and *Type-3* workflows using a federation of EC2 and GoGrid Cloud. We do not consider *Type-1* workflows here that obviously do not benefit from a Cloud federation. We display the tradeoff solutions in Figures 11, 12, and 13, showing for each solution (numbered $0-9$ on the horizontal axis) the makespan and cost normalised in the interval $[0, 1]$. We performed this normalisation using the maximum and minimum values within the Pareto frontier for both the makespan and financial cost. These graphs also show the percentage of simultaneously used instances from the maximum allowed by each provider in form of bar charts, where the height of each bar indicates the percentage normalised in the [0,0.5] interval. As we use a total of 40 resources (20 from each provider), a bar with a height of 0.5 indicates that 20 instances have been used. We use two different bar textures to differentiate between the two considered providers. All experiments in this section report results for workflows with 1000 activities. We omitted those with a different number of activities showing the same behaviour due to space limitations.

Type-2 *workflows.* For *Type-2* workflows, we focus first our analysis on the advantages of using the Cloud federation and comment afterwards on the makespan and cost of the computed solutions. The three graphs in Figure 11 show that the benefits of using a federation of Clouds decreases with the volume of the data managed by the application. When the data volume is low, the workflow makespan can be further decreased by considering the joined use of services from the federation. We observe this behaviour in Figure 11a, where the three tradeoff schedules with smallest makespan (labelled 7, 8, 9) uses all available machines provided by the federation As long as the data volume increases, aggregating resources from different providers does not reduce the makespan, despite the fact that the number of independent activities in the workflow is high. In particular, when the data volume is medium, the percentage of machines used from the federation decreased and never receased the maximum of 40. Finally, when the data volume is high,



(a) *Low* data class. (b) *Medium* data class. (c) *High* data class.

Figure 11: Normalised *Type-2* workflow tradeoff schedules.

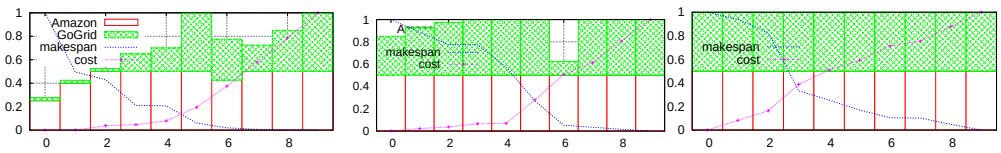


(a) *Low* data class. (b) *Medium* data class. (c) *High* data class.

Figure 12: Normalised *Type-3* workflow tradeoff schedules.

the federated Cloud does not bring any advantage and the computed schedules only use machines of one or the other provider. Regarding the makespan and financial cost, we observe in all cases that a schedule with an overhead of less than 10% with respect to the shortest computed makespan dramatically increases the costs. For example, the difference between the makespan of the schedules labelled 6 and 9 is smaller than 5%, while the difference in financial costs are up to 80% (see Figure 11a). A deeper analysis of the results also reveals that cheap solutions rarely consider federated resources and rather use resources from a single provider. A possible explanation for this behaviour is the hourly based price model offered by the providers, cheap solutions trying to increase resource utilisation instead of launching simultaneous instances.

Type-3 *workflows.* For *Type-3* workflows, the results in Figure 12 show that MOHEFT produced less than $K = 10$ tradeoff schedules which indicates that the tradeoff between makespan and financial cost is lower for this kind of workflows than for the *Type-2*. With respect to the used resources, only few instances are required to execute this kind of workflows as a consequence of the lower number of activities which can be executed in parallel. Therefore, the use of a Cloud federation does not bring any benefit in this situation. Similar to the previous case, the tradeoff analysis between makespan and cost shows that reducing the makespan overhead to less than 10% over the shortest one implies a strong economical investment, while schedules with



(a) 512 frames. (b) 1024 frames. (c) 4096 frames.

Figure 13: Normalised POV-Ray workflow tradeoff schedules.

more than 10% overhead imply a small price fraction.

7.3. Real-World Workflows on Federated Clouds

As explained before, each activity of the POV-Ray workflow renders a frame and transfers it to a final activity which merges all the frames and stores them into a file. Each frame is of around 1 MB in size. Therefore, the POV-Ray application can be considered as a *Type-2* workflow and expects to achieve benefits from using a federation of Clouds. As summarised in Figure 13, it is obvious that the federation helps in reducing the makespan of this application. The three evaluated cases show that the higher the number of activities, the higher the benefit from the federation. This result is a consequence of the high degree of parallelism showed by this workflow application and the low volume of data required.

8. Conclusions and Future Work

We described a new multi-objective list-based scheduling algorithm for scientific workflows called MOHEFT. In contrast to related work based on a-priori aggregative functions and constrained objectives, MOHEFT is a truly multi-criteria optimisation that approximates the Pareto frontier using a number well-distributed tradeoff solutions selected by the crowding distance and evaluated using the HV metric.

We compared our method with the HEFT algorithm and another state-of-the-art heuristic from the multi-criteria optimisation theory (SPEA2*). We considered in our evaluation both synthetic workflows with different characteristics, as well as two real-world ones. In all experiments, MOHEFT obtained equal or even better makespan as the mono-objective HEFT algorithm and better economical cost. For workflows with a few parallel activities, MOHEFT and SPEA2* computed solutions of the same quality having the same makespan as HEFT and better financial cost. For workflows with

a high number of parallel activities and a balanced structure, MOHEFT outperformed SPEA2*, the difference increasing with the number of workflow activities. For workflows with a high number of independent activities and unbalanced structure, MOHEFT delivered better makespan results than HEFT and SPEA2* and higher quality tradeoff solutions than SPEA2*. Experimental results using Amazon EC2 and GoGrid as independent providers illustrated that federated Clouds can help in shortening the makespan for workflow applications which do not require transferring large amounts of data among activities. In situations where data transfers dominate the computation time, the workflows do not benefit from a federation of Clouds and performs better in a single provider configuration.

In the future we plan to extend our analysis for other real-world applications and a wider set of Cloud providers. We will also analyse extensions to MOHEFT to better deal with federated Clouds and big data problems.

Acknowledgement

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, Network for Sustainable Ultrascale Computing (NESUS).

[1] H. Topcuoglu, S. Hariri, W. Min-You, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 260–274.

[2] E. Ilavarsan, P. Thambidurai, Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments, Journal of Computer Science 3 (2) (2007) 94–103.

[3] H. Arabnejad, J. G. Barbosa, List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table, IEEE Transactions on Parallel and Distributed Systems 25 (3) (2014) 682–694.

[4] J. Yu, M. Kirley, R. Buyya, Multi-objective planning for workflow execution on Grids, in: 8th International Conference on Grid Computing, GRID, IEEE Computer Society, ISBN 978-1-4244-1559-5, 10–17, 2007.

[5] S. K. Garg, R. Buyya, H. J. Siegel, Scheduling parallel applications on utility grids: time and cost trade-off management, in: 32nd Australasian

Conference on Computer Science, vol. 91 of *ACSC*, Australian Computer Society, 151–160, 2009.

[6] I. Assayad, A. Girault, H. Kalla, A Bi-Criteria Scheduling Heuristics for Distributed Embedded Systems Under Reliability and Real-Time Constraints, in: International Conference on Dependable Systems and Networks, DSN, IEEE, 347–356, 2004.

[7] M. Hakem, F. Butelle, Reliability and Scheduling on Systems Subject to Failures, in: International Conference on Parallel Processing, IEEE Computer Society, 38, 2007.

[8] R. Sakellariou, H. Zhao, E. Tsiakkouri, M. D. Dikaiakos, Scheduling Workflows with Budget Constraints, in: S. Gorlatch, M. Danelutto (Eds.), Integrated Research in Grid Computing, CoreGRID, Springer, 189–202, 2007.

[9] H. Fard, R. Prodan, J. Barrionuevo, T. Fahringer, A Multi-objective Approach for Workflow Scheduling in Heterogeneous Environments, in: 12th International Symposium on Cluster, Cloud and Grid Computing, IEEE, 2012.

[10] M. Mezmaz, N. Melab, Y. Kessaci, Y. Lee, E.-G. T. albi, A.Y.Zomaya, D. Tuyttens, A Parallel bi-objective hybrid metaheuristic for energy-aware s cheduling for cloud computing systems, Journal of Parallel and Distributed Computing 71 (11) (2011) 14971508.

[11] A. K. M. K. A. Talukder, M. Kirley, R. Buyya, Multiobjective differential evolution for scheduling workflow applications on global Grids, Concurrency and Computation: Practice and Experience 21 (13) (2009) 1742–1756.

[12] L.-C. Canon, E. Jeanot, MO-Greedy: an extended beam-search approach for solving a multi-criteria scheduling problem on heterogeneous machines, in: International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, International Heterogeneity in Computing, IEEE, 2011.

[13] J. Durillo, H. Fard, R. Prodan, MOHEFT: A Multi-Objective List-based Method for Workflow Scheduling, in: 4th IEEE International Conference on Cloud Computing Technology and Science, IEEE, 2012.

[14] J. J. Durillo, V. Nae, R. Prodan, Multi-objective Workflow Scheduling: An Analysis of the Energy Efficiency and Makespan Tradeoff, in: 13th International Symposium on Cluster, Cloud and Grid Computing, IEEE, 203–210, 2013.

[15] L. Rao, X. Liu, W. Liu, Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment, in: INFOCOM, IEEE, 1–9, 2010.

[16] S. Ren, Y. He, F. Xu, Provably-Efficient Job Scheduling for Energy and Fairness in Geographically Distributed Data Centers, in: 32nd International Conference on Distributed Computing Systems, ICDCS, IEEE Computer Society, ISBN 978-0-7695-4685-8, 22–31, 2012.

[17] R. Urgaonkar, U. C. Kozat, K. Igarashi, M. J. Neely, Dynamic resource allocation and power management in virtualized data centers., in: Network Operations and Management Symposium, IEEE, 479–486, 2010.

[18] Y. Yao, L. Huang, A. Sharma, L. Golubchik, M. Neely, Power Cost Reduction in Distributed Data Centers: A two Time Scale Approach for Delay Tolerant Workloads, IEEE Transactions on Parallel and Distributed Systems 25 (1) (2014) 200–211, ISSN 1045-9219.

[19] S. K. Garg, C. S. Yeo, A. Anandasivam, R. Buyya, Environment-conscious scheduling of HPC applications on distributed Cloud-oriented data centers, Journal of Parallel and Distributed Computing 71 (6) (2011) 732–749, ISSN 0743-7315.

[20] I. Santana-Perez, M. S. Perez-Hernndez, A Semantic Scheduler Architecture for Federated Hybrid Clouds, in: 5th International Conference on Cloud Computing, IEEE Computer Society, ISSN 2159-6182, 384–391, 2012.

[21] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2000) 182–197.

[22] J. Yu, R. Buyya, K. Ramamohanarao, Workflow Scheduling Algorithms for Grid Computing, in: F. Xhafa, A. Abraham (Eds.), Metaheuristics for Scheduling in Distributed Computing Environments, vol. 146 of *Studies in Computational Intelligence*, Springer, 173–214, 2008.

[23] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, J. Luitz, WIEN2k: An Augmented Plane Wave plus Local Orbitals Program for Calculating Crystal Properties, Tech. Rep. ISBN: 3-9501031-1-2., Institute of Physical and Theoretical Chemistry, TU Vienna, 2001.

[24] I. Alexandru, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, D. H. J. Epema, Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing, IEEE Transactions onf Parallel and Distributed Systems 22 (6) (2011) 931–945.

[25] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the Strength Pareto Evolutionary Algorithm, TIK-Report 103, Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, URL `http://www.kddresearch.org/Courses/Spring-2007/CIS830/Handouts/P8.pdf`, 2001.

[26] J. J. Durillo, A. J. Nebro, jMetal: A Java framework for multi-objective optimization, Advances in Engineering Software 42 (10) (2011) 760–771.