Handling the Uncertainty in Resource Performance for Executing Workflow Applications in Clouds

Hamid Mohammadi Fard
hamid@dps.uibk.ac.at

Sasko Ristov
sashko@dps.uibk.ac.at

Radu Prodan
radu@dps.uibk.ac.at

Institute of Computer Science, University of Innsbruck
Technikerstr. 21a, A-6020 Innsbruck, Austria

ABSTRACT

Execution of workflow applications in Cloud environments involves many uncertainties because of elastic resource provisioning and unstable performance of multitenant virtual machines (VM) instances over time. These uncertainties are usually either neglected by existing researches, or modeled with some probability distribution function. To address this gap, we extend a multi-objective workflow scheduling algorithm (MOHEFT) in two directions: (1) to deal with the dynamic nature of Cloud environments offering a potentially infinite amount of on-demand resources, and (2) to consider robustness as an objective that mitigates the variability in VM performance over time. Our new robust model, called R-MOHEFT, considers uncertainty in processing times of workflow activities without a precise estimation or known distribution function within an uncertainty interval. We approach this scheduling problem as a three-objective optimisation that considers makespan, monetary cost, and robustness as simultaneous objectives of a commercial Cloud environment. Our new algorithm is able to estimate the Pareto optimal set of scheduling solutions that resist against fluctuations in processing times three times better than its MOHEFT predecessor, with a tradeoff of only 15% worse Pareto frontier. R-MOHEFT's hypervolume suffers by only 5% to 16%, compared to the MOHEFT's drawback of 38% to surprisingly 87%, when the processing time fluctuates up to its double value.

CCS Concepts

•**Theory of computation → Self-organization;** Distributed computing models; •**Computing methodologies →** *Self-organization;* •**Computer systems organization →** *Cloud computing;*

Keywords

Cloud environment; multi-objective optimisation; robust scheduling; uncertain processing time; workflow application

DOI: http://dx.doi.org/10.1145/2822332.2822333

1. INTRODUCTION

Several technological advantages such as elasticity, scalability, accessibility, and reliability, have made of Infrastructure as a Service (IaaS) Clouds a popular alternative in many domains to replace in-house IT infrastructures. In the same way that Clouds have resolved many limitations and concerns within private clusters (e.g. related to on-demand scalability), Cloud computing can be efficiently applied for solving complex scientific problems. An important class of applications that benefit from being executed in Cloud infrastructures are *scientific workflows* [9], as a set of loosely coupled components or *activities* interconnected through control flow and data flow dependencies. *Scheduling* workflow applications in heterogeneous environments including Clouds is an important NP-complete problem for which a large number of heuristic algorithms have been proposed [10, 27]. However, Cloud infrastructures introduce three new challenges in workflow scheduling compared to other distributed systems, such as clusters: the pay-as-you-go pricing model, dynamic on-demand resource provisioning, and instance performance instability over time [28].

Existing theoretical models for scheduling workflows in Clouds assume that the processing time and the performance of Cloud resources can be precisely estimated. Real-world scenarios, however, differ from these mathematical models and are confronted with many *uncertainties*. A realistic Cloud environment is subject to many sources of uncertainties, such as the breakdown of virtual and physical machines, variant setup time [12], and unexpected performance of Cloud instances [2]. The experiments in [1,7] demonstrate how variant the performance of identical Cloud instances can be in different repetitions and during the execution of the experiments, making it hard to predict [23]. Because of this uncertainty in the performance of Cloud resources, we cannot always perfectly estimate the processing times of workflow activities, as often assumed in the literature.

This paper proposes a new and more realistic (robust) model for resource provisioning and scheduling of workflow applications in Cloud infrastructures. Our model assumes that the processing time of a workflow activity is unknown and bound within a certain *uncertainty interval.* The probability function of the processing time distribution in each uncertainty interval is also unknown. The only deterministic information being its lower and upper bound, which can be taken as the minimal and maximal values of execution times. Using this model, we propose a *multi-objective optimisation* approach that approximates the Pareto optimal set of workflow scheduling solutions with respect to makespan, monetary cost, and *robustness* (an objective that shows how much a schedule is resistant to the uncertainty). We solve this three-objective scheduling problem by proposing a new algorithm called *Robust MOHEFT (R-MOHEFT)* that extends the *Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT)* algorithm researched in previous work [8] to deal with the uncertainty in resource performance [4]. Experimental results show that scheduling solutions produced by R-MOHEFT resist aro-und three times better against the fluctuation in processing time than its MOHEFT predecessor. The drawback of this improvement is an only 15% worse Pareto frontier of optimal solutions.

The paper is organised as follows. Section 2 studies the related work, followed by Section 3 that introduces a few background notions on multi-objective scheduling required for understanding our work. Section 4 formally models the application, Cloud, and three-criteria scheduling problem underneath our work. We describe the new R-MOHEFT scheduling algorithm in Section 5, followed by its evaluation in Section 6. Section 7 discusses the importance of R-MOHEFT and Section 8 concludes the paper.

2. RELATED WORK

We classify the related work based on the awareness on the activities processing times into: (1) precisely known, (2) random with known probability distribution function, (3) unknown, and (4) uncertain processing time. We investigate these classes in detail next.

2.1 Precisely known processing time

Many heuristics algorithms have been proposed to approximate a nearly optimal scheduling solution. Given precise estimations of the activity processing times, a full-ahead offline scheduling is the best approach for which a broad set of related works exist [10,27]. The only considered uncertainty is the start time of an activity, which depends on the critical path of activities in the workflow graph.

2.2 Random processing time with known probability distribution function

In the recent years, probability theory and statistical techniques have been incorporated into the scheduling field to treat uncertainties originating from different sources [21]. This has been considered as stochastic scheduling applied for problems in which the activity specifications (e.g. processing time, due date, arrival time) are modeled as random variables with known probability distribution function. For example, Chaves et al. [5] described a TVM-Fuzzy scheduler for minimising the makespan of Cloud applications under uncertain available bandwidth. Poola et al. [17] proposed two metrics to measure robustness of the workflow execution considering historical execution data. Considering the time and cost constraints, they propose a robust workflow scheduling algorithm and resource allocation on heterogeneous Cloud resources assuming the known probability distribution functions for all the data. Malawski et al. [11] discuss the uncertainty in IaaS Clouds considering both time and cost constraints in a set of ensembles of inter-related workflows using a uniform distribution.

2.3 Unknown processing time

Online scheduling cannot be improved when there is no information about the processing time. Online scheduling is characterised by no knowledge about the future activity arrival, and the decisions are being made each time an activity arrives. In such a case, the scheduler usually operates in a just-in-time fashion by submitting each upcoming activity on one available resource.

2.4 Uncertain processing time

Whenever random variables with specific probability distributions are available, stochastic approaches are the usual techniques for scheduling. However, many real-life situations do not give enough information to characterise the probability distribution function of each random parameter, thus requiring other approaches.

In robust scheduling, a decision maker prefers a schedule that hedges against the worst-case scenario [13,19]. Considering stability analysis, Sotskov et al. [21] discuss a multi-stage decision framework consisting of offline planning and online scheduling stages, and a solution based on the minimal dominant set of activity permutations.

Tchernykh et al. [23] discuss the impact of uncertainty for resource provisioning in Clouds. They briefly indicate the main sources of uncertainty and general scheduling approaches such as reactive, stochastic, fuzzy and robust, but do not propose any model or approach for the problem.

Bittencourt et al. [3] suggest a scheduling algorithm based on HEFT [24] to overcome the wrong estimations by considering relative instead of absolute costs for processing and communication times.

Canon and Jeannot [4] proposed several heuristics (including evolutionary algorithms) for simultaneously minimising the makespan and maximising the robustness, including a discussion on metrics to measure the robustness.

Tchernykh et al. [22] addressed non-preemptive scheduling problems on heterogeneous peer-to-peer Grids, where resources are changing over time and scheduling decisions miss information of application characteristics. The authors consider a scheduling algorithm with task replications to overcome possible unappropriate resource allocations in the presence of uncertainty and to ensure better performance. However, the use of replication in Clouds causes a dramatic increase in cost for this scheduling algorithm.

Vredeveld [25] discussed a model for scheduling under uncertainty that combines online and stochastic scheduling. Activities arrive in an online manner and, as soon as an activity becomes known, the scheduler only learns the probability distribution function of the processing time without its actual value. The problem is theoretically discussed in single- and multiple-machines environments.

Werne [26] researched a multiple phase scheduling method in case of uncertainties, when only the lower and the upper bounds for the activity durations are known. In a first offline phase, it constructs a set of potentially optimal schedules considering the uncertainty of the numerical input data. In the second phase, it uses stability and sensitivity analysis when a solution of an optimisation problem has been found, and performs additional calculations to investigate how that solution depends on the numerical input data. Similar to this, we assume a known interval for the processing times of activities executed on a specific Cloud instance type, with neither known probability function, nor fuzzy model available. Such uncertainty has been recently theoretically investigated for general scheduling of problems like open and flow shop [21], but has not been considered for workflow

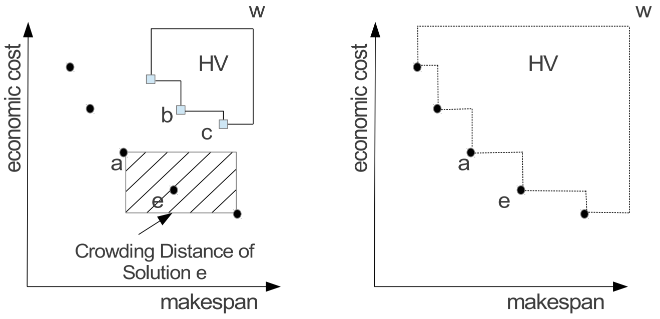


Figure 1: Pareto frontier, hypervolume, crowding distance.

scheduling in Clouds yet.

None of these works approach workflow scheduling in dynamic Cloud environments as a multi-objective optimisation problem able to estimate the Pareto frontier of scheduling solutions that are robust to unpredictable fluctuations within the uncertainty processing time intervals. The Pareto-based approach provides the user with a set of (nearly) optimal solutions, such that each of which represents a tradeoff between the objectives.

3. BACKGROUND

In this section, we introduce the important concepts of the multi-objective optimisation theory and the MOHEFT scheduling algorithm for a better understanding of the paper.

3.1 Multi-objective optimisation

A *multi-objective optimisation* considering more than one objective distinguishes between two spaces: (1) a *solution space* X comprising all feasible solutions, for example the complete set of possible schedules of a workflow application, and (2) an *objective space* O comprising an image of every element of X mapped onto the objective values.

A point $o \in O$ *dominates* $o' \in O$ if o is not worse than o' with respect to all objectives and o is better for at least one of them. A point $o' \in O$ is said to be *non-dominated* if there is no $o \in O$ that dominates o'. A solution $x \in X$ is *Pareto optimal* (efficient) if its image in the objective space is non-dominated. The set of all Pareto optimal solutions is called *Pareto optimal set.* The image of all members of a Pareto optimal set in the objective space is called *Pareto frontier.* A vector comprising the best possible values for all objectives is called *Utopia point.* Such a vector typically dominates the entire Pareto frontier and therefore it is impossible to realistically achieve. A vector comprising the worst possible values for all existing objectives is called *Nadir point* (or anti-utopia). The entire Pareto frontier dominates this point.

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, or how close the tradeoff solutions are to the corresponding optimal solutions, 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 point. This way, more accurate and more diverse the points in X are, the higher hypervolume is achieved. In Figure 1, for example, the set containing the solid round points (right) is better than the set containing the squared solutions (left) because the area enclosed within the dashed lines is larger than the one represented by the solid lines. A method to ensure the diversity of the solutions is to maximise the *crowding distance* [6] graphically depicted as a rectangle around the solution e in Figure 1 (left), which gives a measure of the area surrounding a solution where no other tradeoff solution is placed.

3.2 MOHEFT algorithm

The *multi-objective heterogeneous earliest finish time (MOHEFT)* algorithm is an extension of the HEFT scheduling algorithm [24] for estimating the Pareto optimal set of scheduling solutions for a workflow application. We describe the MOHEFT in pseudocode in Algorithm 1. Apart of the workflow application W and the Cloud resource set, the MOHEFT algorithm additionally receives the number np of Pareto optimal solutions to be computed (the size of the $paretoSet$), which is the output of the algorithm. In line 3, we sort the workflow activities in ascending order according to their bottom level (B-rank), defined in the graph theory for each activity A_i as the longest path to the exit activity (A_{exit}), including the activity A_i itself [24]. The outer loop in lines 4 – 16 iterates over all ranked activities of the workflow. For each activity, it generates all possible sub-workflow schedules by separately mapping the activity onto each resource and inserting it into all partial Pareto optimal solutions (lines 9 – 12). From the large set of partial *solutions*, the algorithm selects the best np schedules based on the crowding distance metric in lines 14 – 15. The criterion is to prefer the solutions with a higher crowding distance, since the Pareto set represents a wider area of different tradeoff solutions. The result of each iteration is utmost np Pareto optimal solutions for the sub-workflows traversed so far. The algorithm returns the computed Pareto optimal set of the scheduling solutions in line 17 after all activities have been traversed.

4. THE MODEL

This section formally models the workflow, Cloud and three-criteria scheduling problem underneath our work.

4.1 Workflow application model

We model a *workflow application* as a precedence constraint graph (A, D) consisting of a set $A = \bigcup_{i=1}^{n} \{A_i\}$ of n *activities* interconnected through a set of dependencies $D = \{(A_i, A_j, D_{ij}) \mid (A_i, A_j) \in A \times A\}$, where (A_i, A_j, D_{ij}) implies that A_i needs to be executed before A_j, and the size of data to be transferred from A_i to A_j is D_{ij} bytes. The activities are assumed to be non-preemptive, so is not allowed to suspend one and resume it later on.

The function $pred : A \rightarrow \mathcal{P}(A)$, where $\mathcal{P}$ denotes the power set, returns the set of immediate *predecessors* of each activity $A_i \in A$ (i.e. $A_j \in pred\,(A_i) \iff (A_j, A_i, D_{ji}) \in D$), while the function $succ : A \rightarrow \mathcal{P}(A)$ returns the set of immediate *successors* of the activity A_i (i.e. $A_j \in succ\,(A_i) \iff (A_i, A_j, D_{ij}) \in D$). Each workflow has an *entry activity* A_{entry} with no predecessors (i.e. $A_{entry} \in A : pred\,(A_{entry}) = \emptyset$) and an *exit activity* A_{exit} with no successors (i.e. $A_{exit} \in A : succ\,(A_{exit}) = \emptyset$).

Algorithm 1: MOHEFT algorithm.

```
Input: Workflow application: $W = (A, D)$; Resource set: $resourcePool$; Number of Pareto optimal solutions: $np$.
Output: Pareto optimal set of workflow schedules: $paretoSet$.
begin
    $paretoSet[0 : np - 1] \leftarrow \emptyset$                                  /* Create Pareto set of $np$ empty workflow schedules */
    $rankedActivities \leftarrow sortBrank(A)$                                    /* Sort activities based on B-rank */
    foreach $A_i \in rankedActivities$                                            /* Iterate over all ranked activities */
    do
        $solutions \leftarrow \emptyset$                                          /* All sub-workflow schedules in each iteration */
        foreach $R_j \in resourcePool$                                            /* Iterate over all possible resources */
        do
            foreach $paretoSolution \in paretoSet$                                /* Iterate over the current Pareto solutions */
            do
                $solutions \leftarrow solutions \cup \{paretoSolution \cup (A_i, R_j)\}$   /* Save the new sub-workflow schedule */
            end
        end
        $solutions \leftarrow sortCrowdingDistance(solutions)$                    /* Sort the solutions based on crowding distance */
        $paretoSet \leftarrow tail(solutions, np)$                                /* Select the best $np$ solutions based on crowding distance */
    end
    return $paretoSet$
end
```

Each activity A_i has a *requirement vector* R_i, which defines its hardware or software requirements such as the minimum value of memory or storage needed for execution. We express the *computational complexity* w_i (i.e. work) of each activity A_i in million of instructions (MI).

4.2 Cloud infrastructure model

A Cloud provider offers a set of r virtual machine (VM) *instance types* $T = \bigcup_{k=1}^{r} \{T_k\}$. We assume without loss of the generality an hourly-based pricing model, which means that the leasing duration of an instance is rounded to the next full hour, as offered by most commercial Cloud providers. Each instance type T_k is characterised by three parameters: *computational speed* s_k in million instructions per second (MIPS), *cost per time unit* c_k, and *boot delay* b_k.

We denote the set of available *VM instances* as: $I = \bigcup_{j=1}^{m} \{I_j\}$, whose number m may dynamically change since instances can be started and terminated on-demand any during the workflow execution. Each instance I_j has an associated instance type T_k defined as a function: $type : I \rightarrow T$.

We model the *expected processing time* t_i^j of an activity A_i as the ratio between its computational complexity and the speed of the instance I_j on which it is executed, plus a boot delay b_j of the instance I_j if it is not available:

$$t_i^j = \begin{cases} \frac{w_i}{s_k}, & I_j \in I \wedge T_k = type\left(I_j\right); \\ b_k + \frac{w_i}{s_k}, & I_j \notin I \wedge T_k = type\left(I_j\right). \end{cases}$$

Since the performance of Cloud resources (s_k) and the boot delays (b_k) are not precisely predictable, we assume that only the lower and the upper bounds of the expected t_i^j processing time are known within the *uncertainty interval* $t_i^j \in \left[lower\left(t_i^j\right), upper\left(t_i^j\right)\right]$ with an unknown probability distribution function. Shrinking this interval reduces the uncertainty of t_i^j.

4.3 Three-objective scheduling model

We first define a *workflow schedule* as a function $S : A \rightarrow I$ that maps each activity $A_i \in A$ to one Cloud instance $I_j \in I$. We consider three simultaneous optimisation objectives in our scheduling problem: makespan, monetary cost for execution in a commercial Cloud environment, and robustness to fluctuations in instance performance and boot delay. The objective is to minimise all three objectives.

4.3.1 Makespan

The *completion time* of an activity A_i executed on an instance I_j is the latest completion time of all its predecessors plus its expected processing time:

$$end\left(A_i\right) = \begin{cases} t_i^j, & A_i = A_{entry}; \\ \max\limits_{A_p \in pred(A_i)} \left\{end\left(A_p\right) + t_i^j\right\}, & A_i \neq A_{entry}. \end{cases}$$

The first optimisation objective, the workflow *makespan* M, is given by the completion time of the exit activity:

$$M = end\left(A_{exit}\right).$$

4.3.2 Cost

We calculate the *cost* c_i^j of executing the activity A_i on the instance I_j as follows:

$$c_i^j = \begin{cases} 0, & I_j \in I \wedge t_i^j \leq u_j; \\ \left\lceil \frac{t_i^j - u_j}{h} \right\rceil \cdot c_k, & I_j \in I \wedge t_i^j > u_j; T_k = type(I_j); \\ \left\lceil \frac{t_i^j}{h} \right\rceil \cdot c_k, & I_j \notin I; \quad T_k = type(I_j). \end{cases}$$

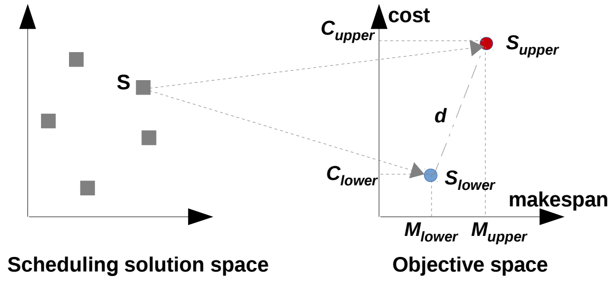
where u_j is unused time of the instance I_j and h is the time unit (i.e one unit per hour) charged by the cloud provider. Executing an activity within the unused time of an existing instance has no cost as the resource is already leased; otherwise the leased full time unit must be payed.

The second optimisation objective, the *cost* C of the workflow execution, is calculated as:

$$C = \sum_{\forall A_i \in A \wedge S(A_i) = I_j} c_i^j.$$

4.3.3 Robustness

Dynamic models can be usually solved using different methods, such as certainty analysis, sensitivity analysis and stability analysis for theoretic models, and simulation for more complex ones that do not fit within a theoretic model [26]. In our workflow scheduling model, the theoretical approach is not applicable because the model is too complex. Because of the uncertainty in the processing time of activities, an optimal offline scheduling solution is usually non-optimal. Thus, we need to find a heuristic approach that will handle these uncertainties.



Scheduling solution space

Objective space

Figure 2: Mapping a scheduling solution S onto the objective space.

Depending on the variant processing time of activities, each scheduling solution S may map onto different points in the objective space with a different makespan and cost. We denote by S_{lower} the point in the objective space representing the M_{lower} makespan when the lower processing time *lower* $\left(t_i^j\right)$ is applied, with a corresponding C_{lower} cost. Similarly, we use the notation S_{upper} for the point *upper* $\left(t_i^j\right)$ in the objective space representing the M_{upper} makespan and the C_{upper} cost.

In order to take the uncertainty of processing time into account, we define the *robustness* R of a scheduling solution S based on the Euclidean distance $d\left(S_{lower}, S_{upper}\right)$ between points S_{lower} and S_{upper} in the objective space:

$$d\left(S_{lower}, S_{upper}\right) = \sqrt{\left(M_{upper} - M_{lower}\right)^2 + \left(C_{upper} - C_{lower}\right)^2}.$$

For a better understanding, Figure 2 represents the mapping of a scheduling solution S onto the S_{lower} and S_{upper} points in objective space, including its robustness.

The motivation for choosing the three objectives is that we need to find a scheduling solution S that minimises the makespan (M_{lower}) and cost (C_{lower}) by applying the lower bounds of the processing times, while at the same time achieves a minimal robustness, measured as the distance between S_{upper} and S_{lower}.

5. ROBUST MOHEFT

This section describes our new scheduling algorithm called *Robust MOHEFT (R-MOHEFT)* in two steps. First, we present how to tailor an offline full-ahead scheduling algorithm like HEFT for a Cloud environment. Then, we extend the MOHEFT algorithm to solve this three-objective scheduling problem.

5.1 Tailoring scheduling algorithms for Clouds

In a Cloud environment, customers can access a very large amount of resources. In such an environment, instead of having a static amount of resources as in clusters, the scheduler needs to deal with a resource pool that may dynamically be changed over the time. Therefore, the scheduling approaches need to be correspondingly tailored by scaling out or scaling in the Cloud infrastructure over time. This impacts the time complexity of the scheduling problem in a Cloud environment, as defined in Theorem 1.

THEOREM 1. *The time complexity of scheduling n activities in a Cloud with r instance types is* $\frac{(r+n-1)!}{(r-1)!}$.

PROOF. To execute each activity in the Cloud, we need to consider two possible groups of resources: (1) available running resources and (2) potentially new ones. The former group includes the instances that are already leased and not terminated yet, while the latter includes the possible new instances that may be rented from the r instance types offered by the Cloud provider. Since the Cloud offers r different instance types, the new resource can be selected from those r different instances. The resource pool for the first activity A_1 is r because there is no running instance yet. For the second activity, the resource pool contains one available running instance (created by the first activity) and r new alternative instances, which yields $r+1$ maximum possible selections. For the third activity, there is a resource pool of $r+2$ maximum possible selections and so on. The resource pool for the i^{th} activity A_i includes the available running instances (maximum $i-1$) plus r new instances. For the last activity A_n, the scheduler needs to decide from $r+n-1$ maximum available instances. Consequently the number of combinations for scheduling n activities is $\prod_{i=0}^{i=n-1}(i+r)$. This series represents the complexity of scheduling n activities on the Cloud offering r instance types, which is: $\frac{(r+n-1)!}{(r-1)!}$. ☐

The time complexity of scheduling a set of n activities in an environment with a static resource pool of m resources is $O\left(m^n\right)$. Although the time complexity of scheduling in a dynamic Cloud environment is lower, the problem is still NP-hard requiring heuristic algorithms with lower time complexities for solving it. We must note that the privacy will not be an issue since all instances are isolated and private per user that executes a workflow.

5.2 R-MOHEFT algorithm

To solve this three-objective optimisation problem, we extend the MOHEFT algorithm (see Algorithm 1), originally designed as an offline scheduling approach for estimating the Pareto set of workflow scheduling solutions. We tailor this algorithm for the dynamic characteristics of a Cloud environment and customise it to our new three-objective problem in a new approach called *Robust MOHEFT (R-MOHEFT)*, since it absorbs the uncertainty in activities' processing times through several modifications.

Algorithm 2 shows the pseudo-code for the R-MOHEFT algorithm. The inputs are similar to MOHEFT, except of the set of instance types instead of a static resource pool. Because of the dynamic nature of a Cloud environment, we represent each scheduling solution as a tuple that associates to the workflow schedule and the corresponding resource pool of Cloud instances. After the initialisation, we sort the activities based on B-rank metric in the line 3, as in MOHEFT. Similar to MOHEFT, R-MOHEFT contains the same three nested loops that generate all possible sub-workflow schedules by mapping each activity onto each resource separately. Additionally, it extracts the Pareto optimal scheduling solutions for each sub-workflow (line 9), adds all Cloud instance types in the potential resource pool (line 10), and inserts for each instance (lines 11 – 16) the scheduled activity to the scheduling solution and the instance into the utilised resource pool. The result of each outer iteration (line 4–20) is utmost *np* Pareto solutions for the sub-workflow traversed so far. Finally, the algorithm returns the Pareto set *paretoSet* of scheduling solutions (line 21).

The outer loop in lines 4 – 20 iterates n times, the loop in lines 7 – 17 utmost *np* times, while the inner loop in lines 11–16 iterates utmost $(n-1+r)$ times, as discussed in Section

Algorithm 2: R-MOHEFT algorithm.

```
Input: Workflow application: $W = (A, D)$; Cloud instance type set: $T$; Number of Pareto optimal solutions: $np$.
Output: Pareto optimal set of scheduling solutions: $paretoSet$.
begin
    $paretoSet[0 : np - 1] \leftarrow \emptyset$;            /* Pareto set of each iteration.  $paretoSet = \{paretoSolutions\}$ */
    $rankedActivities \leftarrow sortBrank(A)$;                              /* Sort activities based on B-rank */
    foreach $A_i \in rankedActivities$ ;                                 /* Iterate over all ranked activities */
    do
        $solutions \leftarrow \emptyset$;          /* All solutions in each iteration.  $solutions = \{(schedule, resourcePool)\}$ */
        foreach $paretoSolution \in paretoSet$;                     /* Iterate over the current Pareto solutions */
        do
            $(schedule, resourcePool) \leftarrow paretoSolution$;            /* Extract the Pareto solution's elements */
            $tempPool \leftarrow resourcePool \cup T$;          /* Add all Cloud instance types as potential new resources */
            foreach $I_j \in tempPool$ ;                                   /* Iterate over all possible resources */
            do
                $schedule \leftarrow schedule \cup (A_i, I_j)$;         /* Insert scheduled activity to the scheduling solution */
                $resourcePool \leftarrow resourcePool \cup I_j$;                /* Insert resource to the new resource pool */
                $solutions \leftarrow solutions \cup (schedule, resourcePool)$;                     /* Save the new solution */
            end
        end
        $solutions \leftarrow sortCrowdingDistance(solutions)$;          /* Sort the solutions based on crowding distance */
        $paretoSet \leftarrow tail(solutions, np)$;          /* Select the best $np$ solutions based on crowding distance */
    end
    return $paretoSet$
end
```

Table 1: Simulated instance types.

	`micro`	`small`	`medium`	`large`	`xlarge`
Instance size [EC2 ECU]	0.5	1	2	4	8
Price [\$ / hour]	0.02	0.08	0.20	0.32	0.64

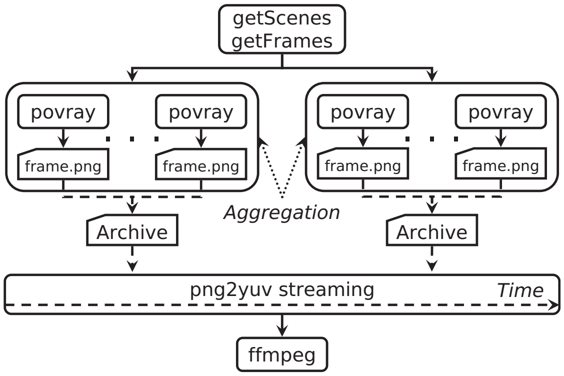


Figure 3: PovRay workflow.

5.1, which yields to a time complexity of $O(n \cdot np \cdot (n - 1 + r))$, which is similar to MOHEFT and much less than the theoretical complexity from Theorem 1.

6. EXPERIMENTAL RESULTS

We ran an extensive set of experiments using the GroudSim Cloud simulator [15] to investigate how ignoring the uncertainty in the processing times affects our objectives, and how robust the scheduling solutions achieved by R-MOHEFT under these uncertain conditions are.

6.1 Experimental environment

We simulated a testing environment with five instance types presented in Table 1. The activities are executed preemptively, that is, a resource is exclusively assigned to one activity until its completion. We generated the boot delay using a uniform distribution between 30 and 60 seconds for each instance type.

We ran the experiments for two real workflow applications: (1) PovRay for creating high-quality three-dimensional graphics based on the POV-Ray tools[1], and (2) WIEN2k [18] material science application for performing electronic structure calculations of solids. The PovRay workflow, depicted in Figure 3, consists of a set of parallel tasks to render frames in PNG format, followed by few sequential tasks to merge all frames, generate the final `mpeg` movie, and transfer it to local host. The WIEN2k workflow contains two parallel sections with sequential synchronisation activities in between, as displayed in Figure 4.

We executed 50 WIEN2k and 50 PovRay workflow applications with various processing times in the $[100, 10000]$ second interval. The uncertainty intervals vary in the $[0, 10000]$ seconds range, such that $lower(t_i^j) \leq upper(t_i^j) \leq 2 \cdot lower(t_i^j)$. We generated the communication time between activities using a uniform distribution between [10,1000] seconds. The number of parallel tasks pt in each loop varies in the interval $[1, 100]$, which means that the total number of activities n varies in the range [5,203] ($n = 3 + 2 \cdot pt$) for the WIEN2k workflow, respectively in the [5,104] ($n = 4 + pt$) range for the PovRay workflow.

To calculate the hypervolume of each Pareto optimal set, we consider the Nadir point as the reference point. The length of an interval does not clearly reflect the variation of the values inside. For instance, two activities with the processing time intervals $[10, 20]$ and $[1000, 1010]$ have the same interval length equal to 10, but the variation for the former is much higher than for the latter. Therefore, we normalise it using the *relative time variation* as follows: $\delta = \frac{upper\left(t_i^j\right) - lower\left(t_i^j\right)}{lower\left(t_i^j\right)}$. Since $upper(t_i^j) \in [lower(t_i^j), 2 \cdot lower(t_i^j)]$, then $\delta \in [0, 1]$. We will experiment with different values of δ with a step of 0.125, ignoring the special case of $lower\left(t_i^j\right) = upper\left(t_i^j\right)$, since there is no uncertainty in this case and the results of MOHEFT and R-MOHEFT

---

[1] http://www.povray.org

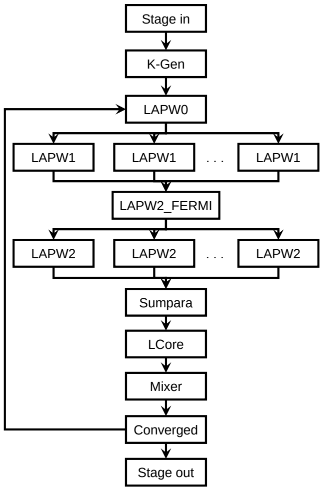


Figure 4: WIEN2K workflow.

are the same.

6.2 MOHEFT robustness

This section shows how suboptimal the MOHEFT solutions can be if we ignore the uncertain intervals $[lower(t_i^j), upper(t_i^j)]$ for processing times and schedule the workflow based on fixed processing times. The first three experiments denoted as MOHEFT(lower), MOHEFT(average), and MOHEFT(upper) use constant values for processing time, namely $lower(t_i^j)$, $\frac{lower(t_i^j)+upper(t_i^j)}{2}$ and $upper\left(t_i^j\right)$. The experiment MOHEFT(lower-upper) maps the solutions of MOHEFT(lower) by applying the upper bounds of the uncertainty intervals, representing by S_{upper} in Figure 2).

To better understand the objective spaces, the Pareto frontiers achieved for a single experiment on PovRay application are shown in Figure 5. As an important result, we observe that the MOHEFT(lower-upper) solutions are not able to dominate those delivered by MOHEFT(upper), which means that the MOHEFT(lower) solutions are not optimal by considering the upper bounds of the uncertainty intervals. In other words, although the solutions delivered in MOHEFT(lower) are Pareto optimal, they become suboptimal when applying the upper bounds $upper\left(t_i^j\right)$ instead of the lower ones $lower\left(t_i^j\right)$ on the same solutions to calculate S_{upper}. Furthermore, a few (two) MOHEFT(lower-upper) solutions are even dominated by other eight solutions, which means that they are not Pareto optimal.

Figure 6 presents the hypervolumes for all four experiments, where the dominated solutions of MOHEFT(lower-upper) are ignored. For a better comparison, we map the hypervolumes in the range $[0, 1]$ by normalising the surface enclosed between the reference points (Nadir points) and the $(0, 0)$ point against the surface enclosed between the reference Nadir points and the Utopia point. We observe that applying different processing times (from the intervals) has a big impact on the Pareto frontier estimated by MOHEFT. The most important observation is that by increasing the relative time variation $\delta \to 1$, the difference between the hypervolumes of MOHEFT(lower) and MOHEFT(lower-upper) is increasing. The hypervolumes are smaller when higher processing times are applied in MOHEFT. This result proves our hypothesis that scheduling decisions based on fixed processing times under uncertainty conditions are not robust under the variation of processing times.

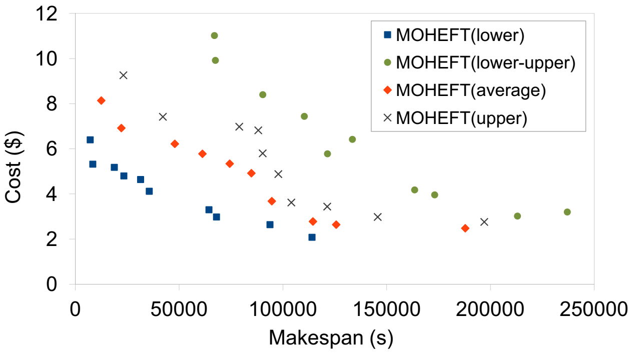


Figure 5: Pareto frontiers of three MOHEFT Pareto optimal sets for different (lower, average, upper) processing times, along with mapping the MOHEFT(lower) solutions by applying the upper bounds of the uncertainty intervals.

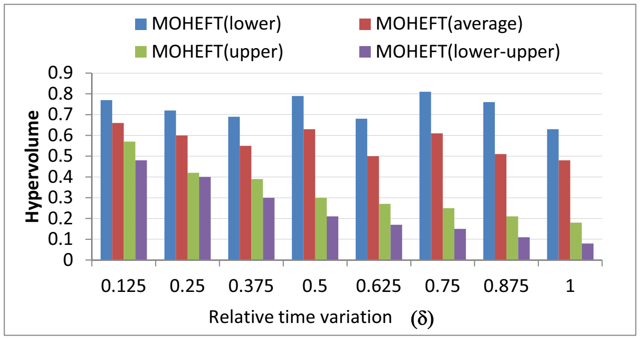


Figure 6: Impact of relative time variation on hypervolume.

6.3 R-MOHEFT robustness

In the next part of our analysis, we investigate the correlation between the robustness of the entire uncertainty interval for a scheduling solution (S_{lower}, S_{upper}) and the scheduling solution (S_{random}, S_{upper}), which denotes the solution in the objective space that considers a random processing time for the activities within the uncertainty interval, instead of the lower bound.

Figure 7 shows a positive correlation between the robustness of S_{lower} and S_{random}. Consequently, shortening the robustness will decrease the fluctuation of the scheduling solutions in objective space for processing times within the uncertainty interval. This observation means that the robustness is a proper factor to handle the uncertainty of the Cloud resource performance in our model. In other words, the scheduling solutions with a shorter uncertainty yield a higher robustness, and viceversa.

6.4 MOHEFT versus R-MOHEFT robustness

Further on, we compare the Pareto frontier achieved by MOHEFT(lower) and R-MOHEFT(lower), and their fluctuations when applying the upper bounds of the uncertainty intervals, denoted as R-MOHEFT(lower-upper) and R-MOHEFT(lower-upper). Figure 8 represents the Pareto optimal

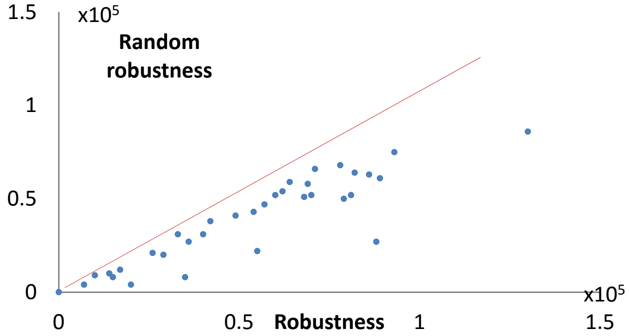


Figure 7: Correlation between (S_{random}, S_{upper}) and (S_{lower}, S_{upper}) robustness.

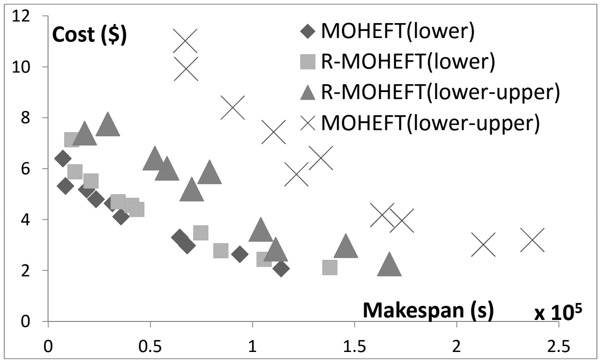


Figure 8: Pareto optimal set comparison between R-MOHEFT and MOHEFT.

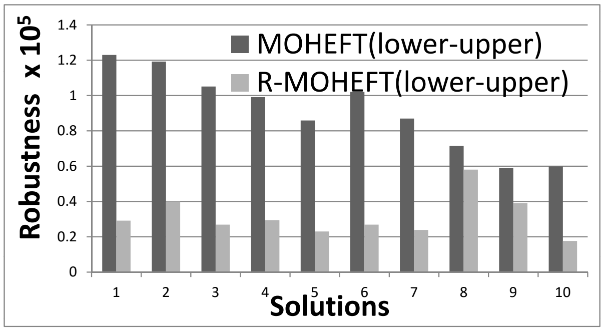


Figure 9: Robustness comparison between R-MOHEFT and MOHEFT.

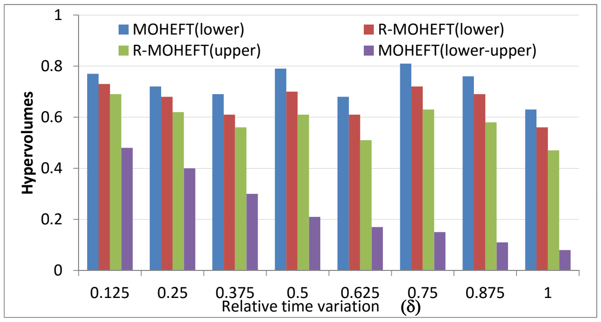


Figure 10: Quantitative analysis of handling the uncertainty. Hypervolume as a function of the relative time variation for the MOHEFT's and R-MOHEFT's scheduling solution sets presented in Figure 8.

set for the same PovRay workflow evaluated in Figure 5. Although MOHEFT(lower) achieves a better hypervolume for the Pareto frontier, half of the solutions achieved by R-MOHEFT(lo-wer) are still not dominated by the solutions of MOHEFT(lower). The comparison between MOHEFT(lower-upper) and R-MOHEFT(lower-upper) shows that the all R-MOHEFT(lower-upper) scheduling solutions are more robust than the MOHEFT(lower-upper)'s ones, as in Figure 9 depicts.

Let us discuss about the angle of the robustness vector with the makespan (X-axis). The angle with the cost (Y-axis) is a complementary to 90 degrees. We observe that the angle is greater for smaller makespan, which means that the uncertainty of the performance of an instance impacts more on the greater objective of a scheduling solution. Comparing the angles of scheduling solutions for MOHEFT and R-MOHEFT for the lower cost, we can conclude that the angles of MOHEFT's solutions are greater than R-MOHEFT's, which means that R-MOHEFT's solutions prefer the increasing of the makespan, rather than the cost when the processing time of activities will increase.

A deeper comparison of the hypervolumes as the summary of the experiments is presented in Figure 10, which shows that the R-MOHEFT algorithm resists the uncertainty in processing times for each relative time variation. The R-MOHEFT(lower-upper)'s hypervolume always follows the R-MOHEFT(lower) one and is three times larger in average than the corresponding MOHEFT(lower-upper) one. The tradeoff for the handling of this uncertainty is that R-MOHEFT(lower)'s hypervolume is smaller by 15%.

In the case of only 12.5% uncertainty in processing time (for $\delta = 0.125$), which is a real case in the Amazon EC2 infrastructure [7], MOHEFT's hypervolume is reduced from 0.77 to 0.48 (or by a significant 37.7%), compared to R-MOHEFT's reduction of only 5.5%, from 0.73 to 0.69. R-MOHEFT even increases its benefits for larger uncertainty intervals and higher relative time variations. For $\delta = 1$, MOHEFT degrades its hypervolume by 87.3%, compared to the R-MOHEFT's degradation of only 16%.

6.5 Competitive ratio analysis

The *competitive ratio* ρ measures the effectiveness of an online scheduling algorithm [16] compared to the best offline scheduling algorithm. An online scheduling algorithm is ρ-competitive if the objective value of the generated scheduling solution is at most ρ times higher than the objective value of the solution achieved by the optimal offline scheduling algorithm. This means that an online solution with a smaller competitive ratio has a better value for the corresponding objective.

To calculate ρ, we considered the solution achieved by MOHEFT(lower) as the optimal offline solution. Figures 11 and 12 represent the competitive ratio of MOHEFT and R-MOHEFT algorithms for the makespan and cost. In comparison to MOHEFT, R-MOHEFT yields always lower competitive ratios both for makespan and cost, which means that the R-MOHEFT's online scheduling solutions are much better than the MOHEFT's ones. Another observation is that increasing δ directly causes a continuous increase in

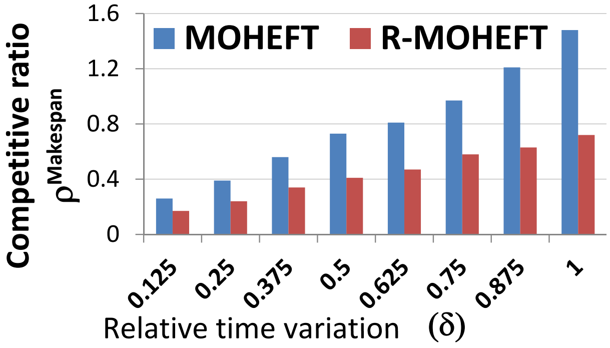


Figure 11: MOHEFT versus R-MOHEFT competitive ratio for the makespan.

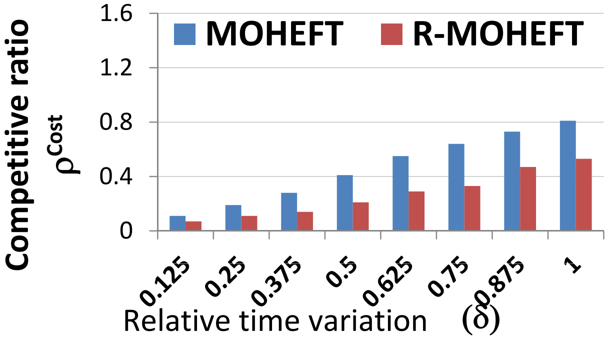


Figure 12: MOHEFT versus R-MOHEFT competitive ratio for the cost.

the competitive ratios for both MOHEFT and R-MOHEFT. Moreover, comparing both competitive ratios ρ, we can observe that increasing the relative time variation δ impacts more the makespan, rather than the cost. This observation is because of the larger difference between interval lenght of time compared to the interval length of cost.

7. DISCUSSION

Scheduling does not find the optimal solution, but close-to-optimal. Nevertheless, in uncertain environment, such as cloud, results show that close-to-optimal solution for constrained environment does not necessarily mean that it will be the optimal solution in uncertain environment. Our new model mitigates these risks by introducing new objective that is relevant for robustness.

R-MOHEFT is a more realistic model compared to others from the literature, since the performance of instances in cloud is variable and less predictable than the premise's one. It uses a dynamic scheduling, since its shows a better performance [11]. Although, for example, activities' incoming rate or instance failure rate could be modeled with a Poisson's distribution [14], their processing times cannot. Some authors modeled the processing times by learning the history and using machine learning algorithms such as support vector machine, while others took the mean value of the historic results. In both cases, the measured and real processing times differ, which could increase the slack and lead to noncompliance with the makespan [20]. Our model uses instead an uncertainty interval of processing times without knowing the probability distribution functions.

The R-MOHEFT model generalises its predecessor MOHEFT by handling the uncertainty in the processing time of an activity. In doing this, it uses the lower bound of the uncertainty interval in order to determine the scheduling solutions. Although one could argue that the average value of the interval should be considered instead, such Poola et al. [17], as Section 6.3 shows, the random robustness follows the robustness of the entire uncertainty interval, thus allowing us to simplify the model. Another motivation in choosing the lower bound is the investigation of the impact of makespan and cost increase rather that decrease, the latter dominating the former. Another improvement is that R-MOHEFT assumes elastic available resources in each step (from r up to $r + n$), instead of MOHEFT's constant resources (total of r resources).

8. CONCLUSION

Several technological advantages of commercial Cloud, such as elasticity, scalability, accessibility, reliability and the pricing model, have made the Clouds popular environments for execution of complex scientific workflow applications. In comparison to the traditional approaches, three special features of commercial Clouds need to be particularly handled: elasticity in resource provisioning, pay-as-you-go pricing model, and uncertainty in resource performance, largely ignored in related work.

In this paper, we proposed an innovative approach for scheduling workflow applications and resource provisioning in commercial Cloud infrastructure, which handles the uncertainty in performance of resources. In our model, we assume neither a precise estimation, nor a probability distribution function for the processing time of activities on Cloud resources. The only available knowledge on the processing time of activities is the lower and upper bounds of their processing times. We approach this problem by proposing a novel multi-objective algorithm called R-MOHEFT that extends the existing MOHEFT algorithm in two directions: (1) it deals with a potentially unlimited amount of on-demand Cloud VM instances and (2) it considers the robustness of scheduling solutions when optimising the makespan and cost of workflow executions under uncertainty. Our new algorithm is able to approximate the Pareto optimal set of scheduling solutions that strongly resist against the fluctuation in processing times, three times better than MOHEFT with a similar time complexity. The tradeoff for this improvements is a slightly worse Pareto frontier of tradeoff solutions of only 15% in hypervolume.

We plan to extend our work by including other uncertainty factors related to the Cloud providers such as carbon emission and energy costs, directly connected to the uncertain processing times of workflow activities.

Acknowledgments

This work is jointly supported by the European Union's Horizon 2020 research and innovation programme under the grant agreement 644179 (ENTICE: dEcentralized repositories for traNsparent and efficienT vIrtual maChine opErations), by the Austrian Science Fund (FWF) project TRP 237-N23 (Workflows on Manycore Processors), and by the Austrian Research Promotion Agency (FFG) project 5077019 (Tiroler Cloud: A federated energy-aware Cloud for industrial, business, and scientific applications).

9. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr 2010.

[2] A. O. Ayodele, J. Rao, and T. E. Boult. Performance measurement and interference profiling in multi-tenant clouds. In *Cloud Computing (CLOUD), 2015 IEEE 8th Int. Conf. on*, pages 941–949, June 2015.

[3] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira. Using relative costs in workflow scheduling to cope with input data uncertainty. In *Proc. of the 10th Int. Workshop on Middleware for Grids, Clouds and e-Science*, MGC '12, pages 8:1–8:6. ACM, 2012.

[4] L.-C. Canon and E. Jeannot. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):532–546, 2010.

[5] C. Chaves, D. Batista, and N. da Fonseca. Scheduling cloud applications under uncertain available bandwidth. In *Communications (ICC), 2013 IEEE Int. Conf. on*, pages 3781–3786, June 2013.

[6] K. Deb. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[7] J. Dejun, G. Pierre, and C.-H. Chi. EC2 performance analysis for resource provisioning of service-oriented applications. In *Proceedings of the 2009 International Conference on Service-oriented Computing*, ICSOC/ServiceWave'09, pages 197–207, Berlin, Heidelberg, 2009. Springer-Verlag.

[8] J. J. Durillo, R. Prodan, and H. M. Fard. MOHEFT: A multi-objective list-based method for workflow scheduling. In *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 185–192, 2012.

[9] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the use of cloud computing for scientific workflows. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 640–645, Dec 2008.

[10] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, Dec. 1999.

[11] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *Proc. of the Int. Conf. on HPC, Networking, Storage and Analysis*, SC '12, pages 1–11, 2012.

[12] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430, June 2012.

[13] R. Montemanni. A mixed integer programming formulation for the total flow time single machine robust scheduling problem with interval data. *J. of Mathematical Modelling and Algorithms*, 6(2):287–296, 2007.

[14] D. Oliveira, K. A. C. S. Ocaña, F. Baião, and M. Mattoso. A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *Journal of Grid Computing*, 10(3):521–552, 2012.

[15] S. Ostermann, K. Plankensteiner, and R. Prodan. Using a new event-based simulation framework for investigating resource provisioning in clouds. *Scient. Programming*, 19(2-3):161–178, 2011.

[16] M. Pinedo. Parallel machine models (deterministic). In *Scheduling*, pages 111–149. Springer US, 2012.

[17] D. Poola, S. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao. Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In *Advanced Information Networking and Applications, 2014 IEEE Int. Conf. on*, pages 858–865, May 2014.

[18] K. Schwarz, P. Blaha, and G. K. H. Madsen. Electronic structure calculations of solids using the wien2k package for material sciences. *Computer Physics Communications*, 147(71), 2002.

[19] M. Sevaux and K. SÃűrensen. A genetic algorithm for robust schedules in a one-machine environment with ready times and due dates. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2(2):129–147, 2004.

[20] Z. Shi, E. Jeannot, and J. J. Dongarra. Robust task scheduling in non-deterministic heterogeneous computing systems. In *2006 IEEE International Conference on Cluster Computing*, pages 1–10, Sept 2006.

[21] Y. N. Sotskov, N. Sotskova, T.-C. Lai, and F. Werner. *Scheduling under uncertainty. Theory and algorithms.* RUE "Publishing House", "Belorusskaya nauka", 2010. Chapter 2, page 84.

[22] A. Tchernykh, J. E. Pecero, A. Barrondo, and E. Schaeffer. Adaptive energy efficient scheduling in peer-to-peer desktop grids. *Future Generation Computer Systems*, 36:209 – 220, 2014.

[23] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, and E. ghazali Talbi. Towards understanding uncertainty in cloud computing resource provisioning. *Procedia Computer Science*, 51:1772 – 1781, 2015. Int. Conf. on Computational Science, {ICCS} 2015.

[24] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. on Par. and Distr. Systems*, 13:260–274, 2002.

[25] T. Vredeveld. Stochastic online scheduling. *Computer Science - Research and Development*, 27(3):181–187, 2012.

[26] F. Werne. Scheduling under uncertainty. In *Proceedings of the 13th International Conference on Project Management and Scheduling (PMS 2012)*, pages 67 – 70, 2012.

[27] M. Wieczorek, A. Hoheisel, and R. Prodan. Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generations Computer Systems*, 25(3):237–256, 2009.

[28] F. Wu, Q. Wu, and Y. Tan. Workflow scheduling in cloud: a survey. *The Journal of Supercomputing*, 71(9):3373–3418, 2015.