This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Pham, Phuong; Fahringer, Thomas

Evolutionary Multi-objective Workflow Scheduling for Volatile Resources in the Cloud

Published in:
IEEE Transactions on Cloud Computing

DOI:
[10.1109/TCC.2020.2993250](https://doi.org/10.1109/TCC.2020.2993250)

E-pub ahead of print: 08/05/2020

Document Version
Peer reviewed version

Please cite the original version:
Pham, P., & Fahringer, T. (2020). Evolutionary Multi-objective Workflow Scheduling for Volatile Resources in the Cloud. *IEEE Transactions on Cloud Computing*, 8(2). <https://doi.org/10.1109/TCC.2020.2993250>

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

Evolutionary Multi-objective Workflow Scheduling for Volatile Resources in the Cloud

Thanh-Phuong Pham and Thomas Fahringer

Abstract—The cloud has been widely used as a distributed computing platform for running scientific workflow applications. Most of the cloud providers encourage the use of their underutilized resources as spot instances for much cheaper prices compared with common resources as on-demand instances, however, the promise of lower costs for resources results in the volatility such that spot instances can be interrupted at any time by cloud providers. Many workflow scheduling algorithms have been proposed to deal with volatile resources. In this work, we consider the two most important features of the volatile resources namely fulfillment and interruption rates to fully model the instability of the cloud infrastructure. Subsequently, we propose a novel evolutionary multi-objective workflow scheduling approach to generate a set of trade-off solutions that outperform state-of-the-art algorithms in both makespan and economic costs. In addition, we explore the fluctuation of makespan and costs for our obtained schedules under different levels of fulfillment and interruption rates. Experimental results with the five well-known real-world workflows demonstrate that our evolutionary multi-objective workflow scheduling algorithm is competitive in terms of makespan and cost compared with state-of-the-art on-demand scheduling techniques.

Index Terms—Workflow Scheduling, Workflow Makespan and Cost, Evolutionary Algorithm, Spot Instance

1 INTRODUCTION

Cloud computing is the latest distributed computing paradigm that offers theoretically unlimited and elastic resources for executing scientific workflows. In addition, most of the cloud providers encourage the use of their underutilized resources as Volatile Instances (VIs) or Volatile Resources (VRs) for much cheaper prices compared with Stable Resources or On-demand instances. However, the promise of less cost computing resources comes with a trade-off against its reliability where the VRs can be interrupted at any time by cloud providers [1], [2], [3].

Scheduling scientific workflows on VRs have been widely considered recently since it is really advanced in terms of economic costs [1], [2], [4], [5]. Nevertheless, all the related work neglected the two important features of VRs: availability (fulfillment rate) and reliability (no-capacity interruption) that are described in [3]. These features open up more non-trivial challenge for workflow scheduling algorithms on VRs.

Most of the related work assume that fulfillment rate of VRs is 100% as on-demand and the interruption is caused by only rising price. Actually, the fulfillment rate of VRs is not always 100%, instead part of the requests cannot be fulfilled until 4 hours after the requests have been submitted [3]. Further to this, the price interruption occurs rarely as being obtained in [3]. This also matches up with the recent policy of cloud providers Amazon EC2¹ and Google Compute Engine (constant price). We observed that most of the interruption caused by no-capacity problem regardless of the bidding price of the VRs [3]. Therefore, it is necessary to have a novel approach that considers both fulfillment and interruption features of VRs.

In this work, we model the cloud infrastructure as a combination of volatile resources (VRs) and stable resources (ODs). We consider both fulfillment and interruption features of the VRs in the cloud model. We then introduce a novel evolutionary multi-objective workflow scheduling algorithm for VRs named (EAVR). While the state-of-the-art algorithm such as CTR [2] can only generate schedule solutions satisfying the short deadline of workflow makespan under budget constraint, our EAVR searches for possible trade-off solutions of both makespan and cost. These solutions help users to select a schedule that is suitable for executing their workflow in VRs.

In order to have a fair comparison with related work, we conducted experiments that assume the VRs are always fulfilled and the interruption might occur with different rates. The obtained results showed that our approach can generate better quality solutions than the state-of-the-art algorithm both in terms of makespan and cost. Following also the results, we observed that it is better to discover a good schedule for a workflow in advance and use the schedule to orchestrate the workflow execution in volatile resources. In case the interruption occurs, our proposed strategy is to replace the interrupted instances by corresponding on-demand instances and keep the original schedule. This can give better performance and cost for the workflow execution since dynamically rescheduling workflows is prone to result in local minimal solutions that are usually not satisfied the strict deadlines of the workflow makespan.

In addition, we evaluate also the same schedules obtained by our EAVR on both VRs and only ODs (baseline). The aim of this evaluation is to estimate the quality of the obtained solutions in terms of different levels of fulfillment and interruption rates compared with the corresponding on-demand state-of-the-art schedulers. In other words, we search for a set of non-dominated schedules with different

1. <https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing/>

makespan and cost for different levels of fulfillment and interruption rates and compare the results with the baseline on-demand approach for both makespan and cost. The results of this evaluation enable users to select a schedule solution as long as the makespan and the cost of the workflow are below certain threshold values under different fulfillment and interruption rates.

The main contributions of our work comprise:

- modeling of fulfillment and interruption rates of cloud volatile resources
- novel evolutionary multi-objective workflow scheduling approach which derives a set of trade-off solutions of workflows for different fulfillment and interruption rates
- generation of schedules that outperform the current state-of-the-art algorithm in both makespan and cost
- evaluation of makespan and costs of workflows under the fluctuation of both fulfillment and interruption rates

The paper is organized as follows. In Section II, we present the important related work. Section III introduces background knowledge in multi-objective optimization and evolutionary algorithm for solving multi-objective problems. Next, section IV is used to define fundamental concepts and model the instability of the VRs. Then, we will present the proposed EAVR in section V. Section VI introduces our performance evaluation method and the experimental setup for the EAVR and the state-of-the-art algorithms. Section VII gives a detailed analysis for the obtained results of the EAVR. Finally, Section VIII concludes our work and presents several directions of future work with the contributions.

2 RELATED WORK

There are several works for scheduling scientific workflows in VRs. Poola et al. propose a fault-tolerant scheduling algorithm for workflows in [1]. In the work, Poola relies on a heuristic called Latest Time to On-Demand (LTO) which is the difference between the deadline and the critical path of the workflow. The work limits the use of spot instance to only one instance type.

In addition, Poola et al. extend also their prior work to use multiple VRs types as well as provide the replication and the retry fault-tolerance mechanisms. The algorithm uses VRs for replication when the deadline is tight, and the VRs are only used for executing tasks when there is sufficient slack time [2]. In case VRs are mainly used for executing scientific workflows, they might cause significant influence on the workflow makespan since the essential critical tasks should not be put on VRs.

Monge et al. [6] propose spot instances aware autoscaling for executing scientific workflows. They assume that VRs or Spot prices follow a uniform distribution; this approach does not match with the recent policies of cloud providers such as Amazon EC2², Google Compute Engine³ and it is against the results in [7], [8] which observed that VR prices do not follow any probability distribution.

Furthermore, Monge et al. introduced a three-objective autoscaler considering makespan, cost and potential out-of-bit error of spot instances [9]. This approach assumes that spot instance requests are always fulfilled and the spot instances are only interrupted due to rising prices.]

There are other studies on workflow scheduling for VRs. Richard et al. [10] construct a Markov Decision Process (MDP) for workflow execution and assume that the probability of spot instances of not being revoked is a constant. This assumption is against experiments/observations reported in [3] which show that spot instances can be revoked at any time regardless of the bidding price. Domanal et al. consider the use of Modified Particle Swarm Optimization algorithm to schedule Bag-of-Task applications on both VRs and on-demand instances [11]. Domanal focuses only on large-scale independent tasks, therefore, the approach is not suitable for workflows.

In summary, there are no existing state-of-the-art scheduling algorithms covering both the fulfillment and interruption rates of volatile resources. To the best of our knowledge, we are the first to consider these two important features following the real measurement [3] and the new policy of spot price announced by Amazon EC2⁴ to construct a multi-objective workflow scheduling problem. Subsequently, we also the first to give a multi-objective approach to address the defined workflow scheduling problem in volatile resources of the clouds.

3 BACKGROUND

3.1 Multi-objective optimization

For the sake of completeness, we formally define the multi-objective optimization problem (MOP), Pareto optimal solutions and Pareto front. Generally, the minimization is the goal of all objectives since a maximization problem can be defined as a minimization too. Formally, the problem is defined in equation 1.

$$\begin{aligned} & \underset{x \in \Omega}{\text{minimize}} && F(x) = (f(x_1), f(x_2), \dots, f(x_n)) \\ & \text{subject to} && c_i(x) \leq 0, \forall i \in \Upsilon \end{aligned} \quad (1)$$

where f_i are $n(\geq 2)$ conflicting objectives that have to be simultaneously minimized and the vector $x = ([x_1, x_2, \dots, x_n])^T$ is the vector of decision variable that belongs to a non-empty set Ω called decision space and satisfies the constraint c_i . Υ is the set of all possible constraint values of c_i

Definition 1 A vector $h = [h_1, h_2, \dots, h_n]^T$ dominates another vector $k = [k_1, k_2, \dots, k_n]^T$ if and only if $\forall i \in [1, 2, \dots, n]$, $h_i \leq k_i$ and $\exists j \in [1, 2, \dots, n]$, $h_j < k_j$. This is denoted $h \prec k$. Two solutions are non-dominated if neither of them dominates the other.

Definition 2 In a general multi-objective optimization problem, a Pareto optimal set is defined as a non-dominated set in the decision space that is formally described in equation (2)

$$S = \{\forall x \in \Omega \mid \nexists x^* \in \lambda, f(x^*) \prec f(x)\} \quad (2)$$

2. <https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing>

3. <https://cloud.google.com/compute/pricing>

4. <https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing>

where λ is a set of all decision variables in Ω that are feasible (satisfy all constraints).

Definition 3 For a given Pareto optimal set P , the Pareto front is the value of Pareto optimal set corresponding to f as it is defined by equation (3)

$$PF = \{f(x), x \in S\} \quad (3)$$

A Pareto front represents the trade-off results that allow decision makers to explore the possible non-dominated solutions following their needs. In this way, users do not have to set their references in advance, conversely, they can choose the preferred regions after the Pareto front is calculated.

3.2 Evolutionary Multi-Objective Optimization

Evolutionary algorithms (EAs) simulate the natural evolution processes which have been used successfully for addressing multi-objective problems with various characteristics [12], [13], [14]. The advantage of EAs in the context of MOP is that they can achieve an approximation of the Pareto front in which each corresponding solution in decision space is a trade-off among the conflict objectives.

The EAs usually begin with an initialization process, where the initial search population is generated. The next step of the process is selecting parent individuals from the initial search population, and applying genetic operators (such as crossover and mutation [12], [14]) to generate offspring. The initial population is usually chosen based on some heuristics so that the process will converge faster to the expected results. Next, the search population will be updated by the new offspring using a replacement strategy. This process is repeated until some termination conditions are met, for instance, after a specific number of iterations and the objectives of the evaluation function are met. Finally, the EA returns the Pareto optimal solutions discovered during the search process.

4 WORKFLOW SCHEDULING

4.1 Scientific Workflow Model

A scientific workflow is often described as a Directed Acyclic Graph (DAG) that consists of a set of computational tasks interconnected through data or control dependencies among the tasks. Formally, the workflow is defined as $W = (A, D)$, where $A = \{a_0, a_1, \dots, a_n\}$ is the set of tasks, and $D = \{(a_u, a_v) | a_u, a_v \in A\}$ represents the dependencies. To model the makespan and cost of a workflow in this work, we define all the predecessor of a task a_u such as follow:

$$P(a_u) = \{a_v | (a_v, a_u) \in D\} \quad (4)$$

The set $P(a_u)$ consists of all the predecessor tasks of the task a_u . In other words, these tasks directly connect to the task a_u . In addition, we also assume that each workflow has a single entry task a_{entry} and a single exit task a_{exit} . This requirement can be easily satisfied by adding pseudo a_{entry} and a_{exit} with zero weight on all the edges connected to them in the DAG.

4.2 Volatile Resource Model

For the approach of EAVR, we consider an Infrastructure as a Service (IaaS) as a system that consists of volatile resources (spot instances) and stable resources (on-demand instances). The IaaS provides computational resources via the concept of virtual machines. Each running virtual machine is considered as an instance. In general, an IaaS offers unlimited number of computing resources for users.

A commercial cloud provider like Amazon commonly offers different resource classes for the instances. For example, the resource classes $\{t2.small, t2.medium, m5.large, m5.xlarge, c5.2xlarge, \dots\}$ are offered based on their computing power or capacity (CPUs, memory, bandwidth, etc). In order to use the computing resources, users have to allocate requests for a number of instances that are in the same or different classes (similar or different CPUs, memory, storage, etc). In our work, each instance can be any kind of the resource classes and the instances can belong to the same or different resource classes. Notably, we avoid using the word "instance type" for each kind of the resource classes in our model, since it can be misunderstood with the instance type (spot or on-demand).

Without loss of generality, we assume that each instance (spot or on-demand) has a fulfillment rate and an interruption rate. Theoretically, on-demand instance type has the fulfillment rate 100% and the interruption rate 0%. We formally define an infinite set $I = \{i_0, i_1, i_2, \dots\}$ to describe all available instances in the IaaS platform. Each $i_p \{e_c, r_l\}$ is an instance that has e_c fulfillment rate and r_l interruption rate. Generally, an instance can be on-demand (stable) or spot (volatile). In case it is an on-demand instance, the e_c and r_l are stable and constant as mentioned above. Otherwise, the e_c and the r_l are random numbers as they are in real cloud systems and we consider all possible values of them in our general approach. Concretely, the fulfillment and interruption rates can have any value in the range $[0, 100]\%$.

4.3 Makespan

For the sake of describing the calculation of the makespan of a workflow, it is necessary to define the execution time of a task a_u on an instance i_b in the IaaS platform. Generally, the execution time $t(a_u, i_b)$ of a task a_u on an instance i_b is the sum of the time required for transferring the biggest input data from a predecessor task $a_p \in P(a_u)$ and the time required to execute a_u in i_b . As described in 4.2, the instance can be a volatile instance (spot) or an on-demand (od) instance.

Let $t(a_u, i_b)$ denote the execution time of task a_u in the i_b instance or in the short form $t(a_u)$. From now on, we simply use $t(a_u)$ to describe the task execution time in a specific instance i_b . The equation (5) illustrates the execution time of the task a_u . $Comm$ is the size of the transferred data between a predecessor task a_p and the task a_u , and $band_{a_p}$, $band_{a_u}$ are the bandwidth between a_p , a_u and the storage respectively. The $workload(a_u)$ represents the workload of the task a_u running in the instance i_b , and the s is the speed

of the instance.

$$t(a_u) = \max_{a_p \in P(a_u)} \left\{ \frac{Comm}{band_{a_p}} \right\} + \left\{ \frac{Comm}{band_{a_u}} \right\} + \frac{workload(a_u)}{s} + \beta * ckpt(a_u) \quad (5)$$

In addition, the instance i_b can be interrupted when it is a volatile instance. In order to overcome the problem, we consider the use of checkpointing technique and a corresponding on-demand instance is used to replace the interrupted spot instance. Checkpoint technique is an effective fault tolerance method for spot markets [1], [15]. It takes the task snapshot periodically and recovers the execution in case of the failure occurs.

Let $ckpt(a_u)$ be the checkpoint overhead of the task a_u when it is running in a spot instance. The checkpoint is necessary to against the interruption of i_b in case it occurs. Notably, the coefficient β represents the checkpoint overhead during the execution of task a_u in the spot i_b , β equals to 1, otherwise its value is 0 since there is no need for doing checkpoint of the task in on-demand instance i_b . Following this, we generally define the execution time of a task a_u in the IaaS with volatile resource (see section 4.2) as in equation (6). Particularly, the coefficient β of $t(a_u)_{spot}$ is 1 and the β of $t(a_u)_{od}$ is 0, where $t(a_u)_{spot}$ is the execution time of task a_u in the spot instance and $t(a_u)_{od}$ is the execution time of task a_u in the corresponding on-demand instance respectively.

$$t(a_u) = \alpha * t(a_u)_{spot} + (1 - \alpha) * t(a_u)_{od} \quad (6)$$

The coefficient α illustrates the amount of workload of task a_u that has been executed in the spot. As a result, $(1 - \alpha)$ represents the remaining workload of the task running in the corresponding on-demand instance that has similar computing resources (CPUs, memory, bandwidth, etc) as the spot instance. When the task completely finishes in the spot, α equals to 1. In case, we use only on-demand instances the value of α is zero. In order to describe the makespan of a workflow, we formally define the maximum completion time of all predecessor tasks $z(P(a_u))$ of the task a_u and its completion time as in equation (7) and equation (8):

$$z(P(a_u)) = \max_{a_p \in P(a_u)} \{t(a_p)\} \quad (7)$$

The completion time of a task is calculated from the starting time of the workflow to the ending time of the task. In the equation (8), $z(a_u)$ is the completion time of task a_u , and $z(P(a_u))$ is the maximum completion time of predecessor tasks of task a_u following equation (7):

$$z(a_u) = \begin{cases} t(a_u) & , P(a_u) = \emptyset \\ z(P(a_u)) + t(a_u) & , P(a_u) \neq \emptyset \end{cases} \quad (8)$$

The makespan $z(W)$ of a workflow is the maximum completion time of all tasks and it is also the completion time of the exit task. The completion time of task a_{exit} is the makespan of the workflow following equation (9).

$$z(W) = \max_{a_u \in A} \{z(a_u)\} = z(a_{exit}) \quad (9)$$

4.4 Economic Cost

Amazon IaaS platforms offer different pricing models to encourage users to use their computing resources. In this work, we implement both second-based and hour-based models to calculate the economic cost of using on-demand and spot instances for executing the workflows. Nevertheless, the latter model is used to give an equal comparison with related work in terms of economic cost. We calculate the spot instance price follow the Amazon EC2 spot price model. In this model, users do not pay the bid price, however, they will pay the spot price at the time the instance is allocated. Furthermore, users do not pay for the partial hour when the spot instances are terminated by the provider, however, they have to pay for the full hour in case they terminate the spot instances.

The communication between tasks of the workflows are usually in the form of files. In this paper, we use the global shared storage such as Amazon S3 for processing file transfer among tasks since it offers many advantages. For instance, it is suitable for fault tolerance technique such as checkpoint with regard to providing persistent storage. We also neglect the cost for storing checkpoints as well as the cost of using storage since they are negligible⁵.

The economic cost for executing a workflow basically depends on the time and the number of spot and on-demand instances being used to execute the workflow. Furthermore, the task could be executed on a spot instance, or both spot and on-demand instances to be completed. We define a function $cost(i_b)_{type}$ to calculate the economic cost of using instance i_b for running workflow W , where $type$ is either *spot* or *od*. If the instance is *od*, the on-demand price of that instance is used, otherwise, the spot price is used to calculate the cost.

The cost of executing the workflow $W = (A, D)$ is the sum of the cost of all instances used for the execution illustrated in the equation (10):

$$cost(W) = \sum_{i_b \in I}^n cost(i_b)_{type} \quad (10)$$

where $cost(W)$ is the total cost of executing the workflow, n is the number of instances used for the execution and the function $cost(i_b)_{type}$ is as mentioned above. Notably, the spot instances are not charged for partial hour when being terminated by the provider. In such cases, the cost of using the spot instances is ignored.

4.5 Problem Formulation

Given a workflow $W = (A, D)$ and a IaaS platform $I = \{i_0, i_1, i_2, \dots\}$, a workflow schedule is a mapping solution $Sched = (Orders, Instances)$ that describes the orchestration of the workflow tasks on the available computing resources. The *Instance* and the *Order* are respectively the instances and the scheduling order of the tasks on the instances. The *Order* is usually determined by the dependencies among tasks in the workflow. However, it also depends on the goal or the objective that users want to achieve when execute the workflows.

5. <https://aws.amazon.com/s3/pricing/>

For the sake of describing the workflow scheduling problem in this work, we generally consider the IaaS platform as a combination of unlimited pool of virtual machine resources ($VM_{Pool(e_c, r_l)}$), where each pool is defined by a pair of arbitrary fulfillment and interruption rates (e_c, r_l). Formally, let denote the pool of VM resource as follow:

$$VM_{Pool(e_c, r_l)} = \{i_v, e_c, r_l\}, \forall v \in I \quad (11)$$

The definition in equation (11) illustrates that each pool of VM resource consists of all possible instance i_v and corresponding with an arbitrary pair of fulfillment and interruption rates (e_c, r_l). Without loss of generality, the IaaS is formulated as follow:

$$IaaS = \bigcup_1^{\infty} (VM_{Pool(e_c, r_l)}), \forall (e_c, r_l) \in [0, 1] \quad (12)$$

To shorten the notation of a resource pool, we use $Pool_i$ instead of $VM_{Pool(e_c, r_l)}$, where each i is corresponding to a pair (e_c, r_l). The considered IaaS is indeed a combination of all resource pools and the number of pools are infinite as in equation (12). In order to formalize the scheduling problem in the IaaS as a multi-objective problem (MOP), we define the objective f_i as a function of makespan and cost of the workflow in the resource pool i as in equation (13):

$$f_i(W) = ((z(W), cost(W))_{Pool_i}, (i \in N, i \geq 1) \quad (13)$$

The notation $z(W)$ and $cost(W)$ are in turn the makespan and cost of a workflow defined in equation (9) and (10). Following this, we define the scheduling problem for the volatile infrastructure I as a MOP whose conflict objectives are both makespan and cost of the workflow following a schedule crossing different resource pools as in equation (14):

$$\underset{IaaS}{\text{minimize}} \quad F(x) = (f_1(W), \dots, f_k(W)), (k \geq 1) \quad (14)$$

where f_i is defined in equation (13), k is the number of resource pools. In other words, instead of finding an optimal schedule for one pool of resource, we have to find an optimal schedule across k pools of resources, where k increases from an arbitrary positive integer number to infinite ($k \geq 1$). It is notable that each pool has two objectives: makespan and cost. If we consider k pools in the implementation, the number of objectives is $(2 * k)$. Each schedule solution is a non-dominated solution for $(2 * k)$ objectives.

5 EVOLUTIONARY MULTI-OBJECTIVE WORKFLOW SCHEDULING FOR VOLATILE RESOURCES

The defined workflow scheduling problem is a well-known NP-hard problem. In order to discover appropriate solutions of the problem in a reasonable amount of time, we choose the approach of EAs which have been shown to be increasingly successful for addressing the multi-objective workflow scheduling problem [16], [17]. In this work, we present a novel evolutionary multi-objective approach for VRs (EAVR) to derive a series of trade-off solutions that reflect different fulfillment and interruption rates to deteriorate makespan and cost out of users' expectation. As far as the fulfillment and interruption rates do not go beyond

Schedule_1									
1	3	4	1	4	2	6	5	5	4
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9

Schedule_2									
1	3	3	1	2	2	6	3	5	4
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9

Fig. 1: Modeling the scheduling problem

the considering range which goes from the starting pool to the end pool, the obtained schedules will satisfy the users' requirement for the makespan and costs of workflows.

5.1 Encoding the scheduling problem

The first step for applying the EAs is to model the scheduling problem for which there exist many different approaches [16], [17]. In this paper, we propose a meta-schedule that defines the mapping of each task to each virtual machine. We represent a schedule solution or a chromosome by using a list (or an array) where the index depicts the task identifier (id) and its value describes the virtual machine or instance identifier (vmid) associated with that task. Our solution is similar to Zhang [17] which reduces the time complexity of the evolutionary process compared to other work [16] and it is in line with the workflow simulation environment [18], [19] which we are using for our work.

Examples of the schedule solutions or chromosomes are illustrated in the Figure 1, where the task identifier is from 0 to 9 and the instance identifier goes from 1 to 6. Following the Schedule_1, the task a_0 and a_3 will be executed on instance vmid 1, the task a_2 and a_4 will run on instance vmid 4, and so on. In case two or more tasks are mapped to the same instance, the order of their execution is also kept based on the dependencies among them [19]. For instance, the task a_0 and the task a_3 in the Schedule_1 are mapped into the same instance that has vmid 1 and the index of a_0 is before a_3 , it does not mean that the execution of task a_0 must be before task a_3 . The starting time of each task depends on the hosting instance local scheduler and its predecessor.

5.2 Evaluation Function

In the defined workflow scheduling problem, we consider two objectives (makespan and cost) for each resource pool which is defined by a pair of fulfillment and interruption rate (see section 4.5). In order to evaluate the quality of each schedule candidate for the scheduling problem, we define an evaluate function as in the algorithm 1. This function calculates the makespan and cost for each resource pool at in lines 12-20, that are actually the objectives for a given candidate solution.

To evaluate every generated schedule of the evolutionary process, we have to allocate local memory and load the schedule variable. We define two functions *Variable_Initialization()* and *Update_Schedule()* to allocate memory and load the new value of the schedule variable, respectively. Both makespan and cost of each resource pool are calculated and assigned to the objective vector

Algorithm 1 Evaluate function for multiple resource pools

Input:

Workflow (wf)
VMs in each pool (vms)
List of resource pools (Fulfil,Intr) (pools)

```

1: procedure EVALUATE(wf, vms, pools)
2:   // allocate memory for the schedule variable
3:   Variables_Initialization()
4:
5:   // update schedule variable
6:   Update_Schedule()
7:
8:   // define objective vector f[]
9:   numObjectives  $\leftarrow$  pools.size() * 2
10:  f[]  $\leftarrow$  Initialization(numObjectives)
11:
12:  // calculate the objective vector
13:  k  $\leftarrow$  0
14:  j  $\leftarrow$  0
15:  while k < pools.size() do
16:    f[j]  $\leftarrow$  Makespan(wf, vms, pool[k])
17:    f[j + 1]  $\leftarrow$  Cost(wf, vms, pool[k])
18:    k = k + 1
19:    j = j + 2
20:  end while
21:
22:  // set objectives for the evolution process
23:  Set_Objectives(f)
24: end procedure

```

$f[]$. In addition, this algorithm is used in the evolutionary process, therefore the value of each schedule candidate have to be updated by the function *Update_Schedule()*. The vector objectives $f[]$ is set at the end of the function since it will be used to calculate the non-dominated schedule solutions that satisfy all the objectives.

5.3 Initial Population

In evolutionary approach, the initial population usually gives hints for discovering high quality offspring and it also helps the process converges faster to expected results. The initial population of a workflow scheduling problem is usually a set of potential schedule solutions that can be considered at the starting points for applying the evolutionary algorithm. These solutions are often high quality schedules that can help to reduce the evolutionary algorithm performance as well as discover expected schedule solutions.

It has been shown that the HEFT scheduling algorithm is one of the most potential algorithms to schedule workflow applications on distributed systems such as the cloud. For example, most of the popular related work of multi-objective workflow scheduling algorithms started from HEFT: MOHEFT [20] is an extension of HEFT to discover multi-objective schedules, EMS-C [16] uses HEFT as its initial population, etc. As a result, we also use HEFT solution as our initial population. In our case, we choose only one resource pool with fulfillment rate 100% and interruption rate 0% for calculate the HEFT solution. Since

it is used once at the beginning of the evolutionary process, we implement a function that calculates the HEFT solution for each workflow and use it as the initial population of our algorithm. In order to accelerate the evolutionary process, we create other initial schedules that follow the HEFT solution with a small change. Besides these, we also create some random schedules since they might contribute to the convergence of the evolutionary algorithm.

5.4 Evolutionary Operators

The scientific workflow scheduling problem for volatile resources is a NP-hard problem, thus it is necessary to find appropriate evolutionary or genetic operators to speed up the search performance and the convergence of the algorithm. Following the workflow simulation environment [18], [19], each schedule solution is a planning algorithm that is planned in advance before the real execution of the workflow. In other words, it is a meta-schedule that assigns tasks to instances, however, the order of task execution still follows the dependency among the tasks of the workflow. Therefore, the evolutionary operators such as crossover and mutation do not need to follow the order that was proposed by Zhu et al. [16].

Crossover: This is also called recombination operator that works by exchanging information between two parent candidates to generate the offspring. In this work, we apply the partially-mapped crossover operator [21] to pass the ordering and value information from the parent schedule candidates to the offspring. The operator creates an offspring by uniformly selecting two random cut points of the two parent candidates. The subsection between the cut points is called the mapping section. Then the mapping section of the first parent is copied to the second offspring and the mapping section of the second parent is copied to the first offspring. The offspring is finalized by copying the remaining elements of the corresponding parent. For the case where an element in the offspring is already present (due to the mapping section of the other parent), it will be replaced by the next unmatched element [21].

Mutation: This operator is usually applied to the offspring produced by the crossover operator. We use a swap mutation operator for this work. The operator randomly selects two positions in the schedule offspring and swap their positions. The aim of using this operator is to discover more potential schedules that might have less effect by the unfulfilled problem of the volatile resources.

5.5 Evolutionary Approach For Volatile Resources

In this section, we describe the main idea of EAVR to deal with the fluctuation of volatile resources. The scheduling problem is defined according to algorithm 2. Firstly, each schedule is generated by the function *Create_Candidate()*, then the function *System_Initialization()* is invoked which defines the workflow, virtual machine types, and resource pools to be used by our simulation. The simulation environment of our work is based on [2], [18], [19] which we have extended with several features as described in section 6.

In case the required instances are 100% fulfilled, the planning schedule for the workflow is kept as its original

Algorithm 2 Defining the scheduling problem**Input:**

Workflow (wf)
VMs in each pool (vms)
List of resource pools (Fulfil,Intr) (pools)

```

1: procedure SCHEDULEPROBLEM(wf, vms, pools)
2:   schedule  $\leftarrow$  Create_Candidate(wf, vms)
3:
4:   // initialize the workflow, vms and pools
5:   System_Initialization(wf, vms, pools)
6:
7:   // check the fulfillment of all vms
8:   if (allVms = True) then
9:     Planning_Schedule(schedule)
10:  else
11:    adj_schedule  $\leftarrow$  adjust(schedule)
12:    Planning_Schedule(adj_schedule)
13:  end if
14:
15:  // evaluate schedule results
16:  Evaluate(wf, vms, pools)
17: end procedure

```

version generated by *Create_Candidate()* function. Otherwise, the schedule will be adjusted by the *adjust()* function (see line 11 in the algorithm 2). This function will replace the id of unfulfilled instances by other fulfilled instances. We choose the replaced instances by using the largest fulfilled instance (using *vmid*). Therefore, the schedule that we use to evaluate the objectives under unfulfilled case is also deterministic.

For the case 100% of instances are fulfilled, it is significant to notice that the schedule will be immutable during the execution of workflow even the interruption of some instances occurs. The schedule can be kept since the interrupted instances will be replaced by corresponding on-demand instances. We keep the schedule and evaluate the influence of interruption on the workflow makespan and cost. Therefore, the schedule is only adjusted in case the required instances are not fulfilled.

After the planning schedule is set as in line 8-13, the evaluation of the schedule is started in line 16. The *Evaluate* function is described in algorithm 1. This method will evaluate the schedule across the list of resource pools and the results of it will be used to compute the non-dominated solutions at the end of the evolutionary process.

The evolutionary process for volatile resources is described in algorithm 3. Line 3 of algorithm 3 (*Configuration_Setup*) allows users to select the workflow, virtual machine types and the number of resource pools to be tested. These values are used by the function *System_Initialization()* at line 5 of algorithm 2. The algorithm starts by defining the schedule problem for the workflow (*wf*), the computing infrastructure (*vms*), and the list of resource pools that the users wants to consider. Then the algorithm will generate the initial population following the requirements of the problem (line 10).

The next step of the process is to select the ge-

Algorithm 3 Evolution of the scheduling problem**Input:**

Workflow (wf)
VMs in each pool (vms)
List of resource pools (Fulfil,Intr) (pools)

Output:

Non-dominated Schedule Solutions

```

1: procedure EVOLUTION(wf, vms, pools)
2:   // select workflow, vm types and pools
3:   Configuration_Setup(wf, vms, pools)
4:
5:   // define schedule problem
6:   problem  $\leftarrow$  ScheduleProblem(wf, vms, pools)
7:
8:   // check initial population
9:   if initialization = True then
10:    initial_Pop  $\leftarrow$  Initial_Schedule(wf, problem)
11:  else
12:    initial_Pop  $\leftarrow$  Initial_Random(wf, problem)
13:  end if
14:
15:  // define evolutionary operators
16:  selection  $\leftarrow$  TournamentSelection()
17:  scheduleCross  $\leftarrow$  ScheduleCrossover()
18:  scheduleMutate  $\leftarrow$  ScheduleMutation()
19:
20:  // construct evolutionary algorithm
21:  algorithm  $\leftarrow$  EvolAlg(problem,
22:    selection,
23:    scheduleCross,
24:    scheduleMutate)
25:
26:  // starting evolutionary process
27:  while (algorithm.getNumEvaluation()  $\leq$  L) do
28:    algorithm.start()
29:    algorithm.increaseNumEvaluation()
30:  end while
31:
32:  // obtain non-dominated schedules
33:  results  $\leftarrow$  algorithm.getResults()
34: end procedure

```

netic or evolutionary operators. As mentioned in 5.4, we implement the evolutionary operators via two methods *ScheduleCrossover()* and *ScheduleMutation()*. These methods are then used as the input for the evolutionary algorithm *EvolAlg*. The *EvolAlg* is a general interface in our implementation which can be any evolutionary framework such as NSGA II, SPEA2, etc. This interface will take the problem and evolutionary operators as its input and construct the overall algorithm in the evolutionary process (see line 21-24 of the algorithm 3).

Finally, the process starts to search for the expected solutions under a specific number of iterations. The termination of this process will generate the final population that is also the obtained schedules for the scheduling problem. The method *getResults()* at in line 33 of the algorithm 3 collects all of the non-dominated schedules in the end.

5.6 Complexity

The time complexity of our EAVR is calculated by considering the initialization, evolutionary operators and evaluation function. Given a workflow which has m tasks and an infrastructure which has n resources, the computational complexity for initializing HEFT is $O(m * n)$. The time complexity for both crossover and mutation are $O(N)$ where N is the population. The evaluation function or fitness function of a solution is $O(k * N * m * n)$, where k is number of resource pools. Therefore, the overall time complexity of our proposed algorithm is $O(N * m * n)$.

6 PERFORMANCE EVALUATION

6.1 Experimental Setup

In this work, we use CloudSim [18] to simulate the cloud environment. We have extended CloudSim to support workflow applications [19] and model Amazon spot characteristics [2], [3]. Furthermore, we have included in this version of CloudSim our own implementation of Critical Task Replication (CTR [2]) thus we have a baseline for our EAVR results.

Finally, CloudSim has been extended for more realistic VRs considering recent policies of real cloud providers such as Amazon EC2⁶. We have evaluated makespan & cost for all workflows for different fulfillment & interruption rates and compared them with the on-demand baseline where the fulfillment is always 100% and the interruption is 0%. The additional features are described as follows:

Bidding Strategy: We use the on-demand price as bidding strategy for our experiments. We implement the dynamically request to the Amazon spot price for checking the price. Since the spot price is almost stable following the new policy of Amazon EC2⁷, we model the interruption based on no-capacity problem which is presented as follows.

Fulfillment rate: This is an important parameter that is ignored by related work since most of the work assume that all spot requests are fulfilled (even immediately without waiting time). It is presented in our previous work that the fulfillment rate of spot instance is not always 100% [3], thus, we consider the fulfillment rate by specifying the probability for each resource pool in the range $[0, 100]$ %).

Interruption rate (no-capacity): The no-capacity interruption is another feature that is neglected by related work. We simulate the interruption by specifying the probability $[0, 100]$ % for each resource pool. Each instance is randomly interrupted following the probability determined for each pool.

Workflow applications: We consider in our evaluation 5 well-known workflows: Montage, CyberShake, Epigenomics, Inspiral, and Sipht that are used by related work to evaluate the quality of workflow scheduling algorithms [2], [16]. The detailed information of these workflows are presented in [22], the workflows consist of all basic components for evaluating the workflow scheduling algorithm such as data aggregation, data distribution, pipeline structure, etc. For evaluating the fluctuation of makespan and cost of each

TABLE 1: Instances used for our experimental validation

Amazon EC2 Instance	vCPU	Mem (GB)	Price	
			Spot	On-demand
m1.small	1	1.7	0.0047	0.047
m1.medium	1	3.75	0.0095	0.095
m1.large	2	7.5	0.019	0.19
m1.xlarge	4	15	0.0379	0.379
m2.xlarge	2	17	0.0275	0.275
m2.2xlarge	4	34	0.055	0.55
m2.4xlarge	8	68	0.11	1.1
m3.xlarge	4	15	0.0586	0.308
m3.2xlarge	8	30	0.1172	0.616

workflow due to fulfillment and interruption rates, we use the workflows with size 1000 tasks since the workflows with smaller sizes finish in really short periods of time.

Evolutionary Framework: We use NSGA II [14] framework for searching schedule solutions since it is shown that NSGA II has been increasingly successful for dealing with scheduling problems [16], [17]. However, our approach is general and it can be extended to use any popular evolutionary multi-objective framework such as MOEA/D, SPEA2, etc. For example, the *EvolAlg* in the algorithm 3 can be extended to use the implementation of well-known evolutionary frameworks such as MOEA/D, SPEA2, etc.

6.2 Baseline Evaluation

In this paper, we give an evaluation between our EAVR and the state-of-the-art workflow scheduling algorithm for volatile resources. Concretely, we estimate the results of EAVR with Critical Task Replication (CTR) proposed by Deepak et.al in [2] since CTR has produced the best performance for scheduling workflows using both on-demand and spot instances. Although our EAVR is static and it is not really suitable to compare EAVR with CTR. However, we use the results of CTR like baseline values to evaluate the quality of EAVR.

Clearly, static and dynamic scheduling approaches have advantages and disadvantages. For example, the dynamic one can actively change to satisfy a criterion at runtime while the static one cannot. On the contrary, the static approach usually discovers nearly optimal solutions while the dynamic approach often results in local minimum. In this work, we discover that our obtained schedules outperform state-of-the-art results under different interruption rates.

In order to evaluate the results of EAVR and use the results of CTR like baseline values, we use the same 9 instance types as the related work [1], [2], for both on-demand and spot instances. Table 1 gives a detailed description of these instances. In this evaluation, we consider two resource pools that have 100% fulfillment rates, however, the interruption rates are in turn 10% and 30%. In other words, we evaluate the schedule solutions obtained by the EAVR (algorithm 3) across two resource pools: *Pool_1*(100%, 10%) and *Pool_2*(100%, 30%). We choose the interruption rates based on the probability failure range of the related work for an appropriate evaluation [2].

6.3 Evaluation of fulfillment and interruption rates

In order to evaluate the performance and cost fluctuation of workflows across different levels of fulfillment and interrup-

6. <https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing>

7. <https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing>

tion rates, we explore three types of scenarios *SpotCase_1*, *SpotCase_2* and *SpotCase_3* such as follow:

- *SpotCase_1*: Instances are always fulfilled and interruption might happens. Generally, we consider the two resources pools *Pool_1*(100%,10%) and *Pool_2*(100%,30%) as in section 6.2.
- *SpotCase_2*: Instances might be unfulfilled and the interruption does not happen. In general, the two considered resource pools are *Pool_1*(90%,0%) and *Pool_2*(80%,0%) respectively.
- *SpotCase_3*: Instances are not always fulfilled and the interruption does happen. In general, the two considered resource pools are in turn *Pool_1*(90%,10%) and *Pool_2*(80%,20%).

We run EAVR using only on-demand instances and select the best performance schedule as the baseline value for evaluating the three scenarios. In addition, we use the corresponding cost of the selected schedule as the upper bound of economic cost for evaluating all scenarios. Each experiment was run at least 10 times for each workflow. The aim of this work is to explore the makespan and cost of the workflows in advance for different kinds of instability compared with on-demand baseline, thus, users can make their own decision effectively for using VRs.

SpotCase_1: The effect of the interruption

This test is used to evaluate the decrease in performance due to different levels of interruption. As described in the algorithm 1, we evaluate each schedule candidate across all the considered resource pools. In this case, each interrupted instance will be replaced by a corresponding on-demand instance. We expect to discover the best schedule solutions for dealing with different interruption rates of volatile resources in this case.

SpotCase_2: The effect of the fulfillment

For the sake of describing the usefulness of the EAVR approach in case the requested instances are not fulfilled, we consider an example such as in Figure 1. Without loss of generality, we assume that users have to select a schedule for *Pool_1* and *Pool_2* respectively and both *Pools* have 0% interruption rate. In addition, we assume that *Schedule_1* outperforms the *Schedule_2* when all instances are fulfilled.

However, when the instances are not fulfilled, for example, instance id 4 is not fulfilled in the *Pool_1* and the instances id 4 and id 5 are not fulfilled in the *Pool_2*. Obviously, the *Schedule_2* is less affected by the unfulfilled instances, thus it becomes a better solution than *Schedule_1* for the considered resource *Pools*. We aim to search for such kind of schedule solutions to precisely give an advance evaluation about the fluctuation of workflow makespan and cost to users. Therefore, they can estimate to some extent the volatility of resources are appropriate for executing the workflows.

SpotCase_3: The effect of fulfillment and interruption

This scenario is an extension of the previous case where we consider the occurrence of both fulfillment and interruption during the execution of the workflows. It is expected that this scenario will give an overview on the decrease in performance and the increase in cost of the workflow execution in the worst case. Users, therefore, can make a decision on using volatile resources upon their demands.

7 RESULT ANALYSIS

In this section, we analyse the performance of the EAVR. We present the obtained results of EAVR compared with the state-of-the-art algorithm CTR. Furthermore, we also describe the fluctuation of the makespan and cost of the considered workflows following three scenarios mentioned in section 6.3. We compare the results with the on-demand baseline and give a detailed explanation for the obtained results.

7.1 Evaluating EAVR and CTR

In order to have a baseline for the EAVR, we analyse the results of the state-of-the-art algorithm CTR as shown in table 2. Notably, the makespan is calculated in seconds and the cost is in US dollar(\$). In this work, we investigate the limits of CTR and use the limits as the constraints for evaluating our EAVR approach. In order to discover the limits of CTR, we start by setting strict deadlines and highly relaxed budgets for the algorithm. The aim of these setting is to release the algorithm from the budget constraint and let it discovers the best possible makespan. We observed that CTR could not satisfy the strict deadlines even it still had a huge amount of extra budget. The results are as expected since CTR is a dynamic approach that could fall into local optimal under strict deadlines.

In addition, we also observed that the stricter the deadline is, the greater error the algorithm will be. We obtained the same results for all 5 considered workflows. For example, it is shown in the table 2 that the CyberShake workflow makespans are not satisfied when the deadline is lower than 4000 seconds, the makespans of Inspiral workflow are missed under 3000 seconds, and the Sipht workflow has rather similar results. In addition, the makespans of Epigenomics and Montage workflows are not satisfied following the considered deadlines. Consequently, we consider also the deadline constraints as standard values to evaluate the quality of our EAVR approach.

As mentioned in section 6.2, we consider obtained schedule results of EAVR across the two resource pools: *Pool_1*(100%,10%) and *Pool_2*(100%,30%) for comparing with the CTR algorithm. These results are presented in the table 2 where each row of the table is a non-dominated solution that satisfies 4 objectives (each pool has two objectives: makespan and cost). We observed that EAVR discovered multiple non-dominated solutions that satisfy the smallest deadline while the CTR could not achieve, for all considered workflows. Moreover, the results of EAVR also outperform the results of CTR both on makespan and cost.

It can be observed from the table 2 that the makespans of the workflows fluctuate from hundreds to thousands seconds under the considered interruption rates. This fluctuation goes from around 3% of the workflow makespan (such as Sipht workflow) up to around 45% (as in Montage workflow). However, the makespans are still better than the results obtained by CTR. Furthermore, they are far more from the smallest deadline constraint for each workflow. In other words, the deterioration of performance of each workflow across different interruption rates is still satisfied the strict deadline constraints.

TABLE 2: A Comparison of EAVR with CTR

Workflow	EAVR				CTR			
	Pool_1 (100%, 10%)		Pool_2 (100%, 30%)		Makespan	Cost	Deadline	Budget
	Makespan	Cost	Makespan	Cost				
CyberShake	416.9258	2.792	549.6567	2.8375	8768	37.146	10000	10000
	410.3724	0.5571	558.945	1.6361	6114	52.72	9000	10000
	402.2465	1.9816	563.8445	2.6351	6492	51.085	8000	10000
	416.9261	0.4616	529.5018	1.5089	5456	73.7775	7000	10000
					4764	111.4037	6000	10000
					4127	276.7813	5000	10000
					9845	47.2108	4000	10000
					12841	47.5348	3000	10000
					12841	47.4698	2000	10000
					18786	45.8522	1000	10000
Epigenomics	15134.5487	11.6825	16476.6305	38.9891	88747	6012.5779	75000	10000
	16476.6305	14.2821	18043.7512	36.6584	452311	6023.9937	70000	10000
	15133.9487	22.8913	18043.2749	25.3447	408103	6021.3285	65000	10000
	15134.0275	18.4067	15512.9628	38.082	398224	6019.892	60000	10000
	15134.8851	14.6303	15512.7363	34.6407	398224	6019.0888	55000	10000
	15133.8762	21.8004	16911.0515	27.9153	288671	6009.0741	50000	10000
	15133.8762	27.2637	15513.1601	31.3835	291964	5977.9894	45000	10000
	15133.8762	14.5492	16945.0215	29.7261	421202	5915.1281	40000	10000
	15134.4036	17.9852	16476.3305	27.0907	422585	5838.9475	35000	10000
	15134.8486	15.4225	15506.8674	24.0653	693012	5659.0556	30000	10000
Inspiral	1155.4656	3.2834	1268.9009	9.874	4061	380.9476	12000	10000
	1155.3451	3.4062	1255.3519	10.3324	3930	400.7676	11000	10000
	1155.2595	5.4189	1255.2596	9.4036	3930	445.75	10000	10000
	1155.6622	2.9606	1265.0954	10.9164	3930	445.75	9000	10000
	1152.0157	5.7928	1255.2602	12.0873	3930	445.75	8000	10000
	1151.9636	6.2014	1255.4933	11.6422	3895	445.99	7000	10000
	1155.3457	5.2289	1263.0515	8.4099	3880	448.31	6000	10000
	1155.2252	6.5873	1255.2318	11.7351	3849	450.23	5000	10000
	1149.0879	7.5006	1261.2469	14.3373	3845	438.71	4000	10000
	1155.2595	6.2547	1258.9163	8.8787	4541	910.9879	3000	10000
Montage	227.8672	0.4491	338.5611	1.7973	23472	57.4795	5000	10000
	227.8002	0.4491	341.8761	2.1879	23291	58.021	4500	10000
	231.8996	0.4491	336.4164	1.8007	23119	56.95	4000	10000
	229.5389	1.7708	334.7507	2.0165	27883	37.1539	3500	10000
					26714	33.0088	3000	10000
					26714	33.0088	2500	10000
					26714	33.0088	2000	10000
					26714	33.0088	1500	10000
					26714	33.0088	1000	10000
Sipt	6047.6641	3.24	6149.3314	16.8085	13530	808.22	19000	10000
	6044.4311	3.2984	6148.7192	15.0443	13530	808.22	18000	10000
	6047.8946	3.0953	6150.7126	10.4012	13530	808.22	17000	10000
	6048.0323	3.0169	6150.4098	10.8466	13452	790.81	16000	10000
	6048.1527	4.159	6207.7359	9.7301	13495	844.56	15000	10000
	6047.2613	3.5256	6154.7363	10.3908	13392	876.53	14000	10000
	6049.9848	3.8219	6167.8869	7.8205	14034	928.8325	13000	10000
	6047.2586	5.939	6149.7199	10.7009	13440	993.0494	12000	10000
					13440	990.8688	11000	10000
					13440	987.4935	10000	10000

Nevertheless, the cost of workflow execution varies from 3 (as in CyberShake workflow) to 5 times more expensive (as in Montage workflow). The increase of economic cost is significantly high, however, it is still much cheaper than the state-of-the-art approach. The reason for this is our approach and CTR are completely inverse; while CTR uses on-demand instances mainly and uses spot instances when it is possible, we use mainly spot instances and replace the interrupted one by the corresponding on-demand instance.

Following the obtained results, we conclude that it is better to discover a good schedule for a workflow in advance and use the schedule to orchestrate the workflow execution on volatile resources. In case the interruption occurs, our proposed strategy is to replace the interrupted instances by corresponding on-demand instances and keep the original schedule. This can give a better performance and save economic costs for the workflow execution.

7.2 The effect of fulfillment and interruption rates

For this experiment, we analyse the effect of fulfillment and interruption on the makespan and cost of the workflows. The aim of this work is to evaluate the decrease in performance as well as the increase in economic costs of EAVR

compared with a pure on-demand evolutionary approach. The user is given an overview to what extent the workflow makespan and cost will suffer under different levels of fulfillment and interruption rates.

Without loss of generality, we run our evolutionary approach with only on-demand instances, where we define only one resource pool with 100% fulfillment rate and 0% interruption rate. The economic cost is calculated using corresponding on-demand price. We select the schedule that has the best performance obtained from the approach as the baseline value for evaluating three *Spot_Case* scenarios mentioned in section 6.3. The obtained results are presented in Figure 2 where *OD* stands for our on-demand baseline value.

In general, our EAVR approach discovers schedule solutions that give an overview about the average makespan and cost as expected from the three scenarios. The average makespan will be the smallest in the *SpotCase_1* when the fulfillment rate is 100%. The makespan is worse when the fulfillment rate decreases to 90% and 80% as in the *SpotCase_2* and it becomes the worst in the *SpotCase_3* where instances are not fulfilled and the interruption also occurs. It can be observed that the pattern is mostly repeated through all the considered workflows except a minor difference in

Fig. 2: Scheduling results of the workflows for the considered scenarios

the cases of Montage and Inspiral. The reason for this is that the random interrupted instances of SpotCase_3 do not always lead to the worse makespan than they do in SpotCase_2. However, the results also show that the worst case is always in SpotCase_3.

Conversely, the economic cost follows an inverse pattern. The cheapest price achieves at the SpotCase_2, which is reasonable, since the case consists of only spot instances and there is no interruption. When the interruption occurs, on-demand instances are used to replace the interrupted spot as in the SpotCase_3, the cost increases slightly. Finally, the most expensive case is in the SpotCase_1, where the interruption rate can go up to 30%. It can be observed that when the makespan is short as in the case of CyberShake and Montage workflows (hundreds of seconds), the saving cost is negligible (around 1\$). On the contrary, when the makespan is sufficiently long such as in Sipht, Inspiral and Epigenomics, the saving cost is considerable. For example, the saving costs of the workflows are around 10\$, 36\$, and 120\$ in the case of Inspiral, Sipht and Epigenomics respectively.

Overall, the fluctuation of the makespan can go from 10% up to 300% in the worst case of three scenarios, however, the cost can decline to around 12.5% compared with

the on-demand baseline. This is a kind of useful information that a multi-objective approach can offer. Any schedule solution in the non-dominated set can be considered as a schedule candidate as long as it is satisfied users' interest. For example, the obtained schedules of the SpotCase_3 can be taken in case users are interested in relaxed deadline and they want to save the cost as much as possible. Obviously, the degradation of the workflow makespan might be high, however, the amount of saving budget is also considerable.

7.3 Discussion

In this section, we discuss the usefulness of the discovered schedules for the SpotCase_2 and SpotCase_3. Since the unfulfilled instance is unknown before requesting it, selecting a specific schedule in advance for the considered scenarios is a non-trivial problem. The schedule can only be allocated when additional information about the considered spot instances is available. For example, the fulfillment rate of a particular instance type (m3.xlarge, m2.4xlarge, etc) is announced by the cloud provider or the rate is discovered by lots of experiments as being done by our previous work [3]. In that work, we conducted thousands of experiments during the last two weeks of Jan 2018 and the results revealed that some instance types are usually

unfulfilled, for example, m4.10xlarge, m4.16xlarge in São Paulo region [3]. As long as the information does exist, a deterministic schedule can be allocated to users. Since we know in advance the instance that has a high chance to be unfulfilled, a suitable schedule can be selected.

8 CONCLUSIONS AND FUTURE WORK

In summary, in this paper we have characterized two important features of cloud volatile resources and we defined the cloud volatile resource model. We have introduced a novel evolutionary multi-objective scheduling method for workflow applications that use cloud volatile resources. Our novel approach enables users to evaluate makespan and cost of their workflows in the context of fluctuating fulfillment and interruption rates. Following this, users can select a schedule depending on their needs.

As part of future work, we will extend our scheduling algorithm for different fulfillment and interruption rates. Moreover, we will explore load balancing and system utilization effects of our algorithm.

ACKNOWLEDGMENT

The authors would like to thank Deepak Poola et al. for providing the implementation of the state-of-the-art algorithm that help us evaluating our proposed approach.

REFERENCES

- [1] D. Poola, K. Ramamohanarao, and R. Buyya, "Fault-tolerant workflow scheduling using spot instances on clouds," *Procedia Computer Science*, vol. 29, pp. 523–533, 2014, 2014 International Conference on Computational Science.
- [2] —, "Enhancing reliability of workflow execution using task replication and spot instances," *ACM Trans. Auton. Adapt. Syst.*, vol. 10, no. 4, pp. 30:1–30:21, Feb. 2016.
- [3] T.-P. Pham, S. Ristov, and T. Fahringer, "Performance and behavior characterization of amazon ec2 spot instances," *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 73–81, 2018.
- [4] D. A. Monge, E. Pacini, C. Mateos, and C. G. Garino, "Meta-heuristic based autoscaling of cloud-based parameter sweep experiments with unreliable virtual machines instances," *Computers Electrical Engineering*, vol. 69, pp. 364–377, 2018.
- [5] L. Chen, X. Li, and R. Ruiz, "Cloud workflow scheduling with on-demand and spot block instances," in *2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, April 2017, pp. 451–456.
- [6] D. A. Monge and C. García Garino, "Adaptive spot-instances aware autoscaling for scientific workflows on the cloud," in *High Performance Computing*, G. Hernández, C. J. Barrios Hernández, G. Díaz, C. García Garino, S. Nesmachnow, T. Pérez-Acle, M. Storti, and M. Vázquez, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 13–27.
- [7] V. Khandelwal, A. Chaturvedi, and C. P. Gupta, "Amazon ec2 spot price prediction using regression random forests," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2017.
- [8] M. Baughman, C. Haas, R. Wolski, I. Foster, and K. Chard, "Predicting amazon spot prices with lstm networks," in *Proceedings of the 9th Workshop on Scientific Cloud Computing*, ser. Science-Cloud'18. New York, NY, USA: Association for Computing Machinery, 2018.
- [9] D. A. Monge, E. Pacini, C. Mateos, and C. G. Garino, "Meta-heuristic based autoscaling of cloud-based parameter sweep experiments with unreliable virtual machines instances," *Computers Electrical Engineering*, vol. 69, pp. 364–377, 2018.
- [10] R. G. Martinez, A. Lopes, and L. Rodrigues, "Planning workflow executions when using spot instances in the cloud," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '19. New York, NY, USA: ACM, 2019, pp. 310–317.
- [11] S. G. Domanal and G. R. M. Reddy, "An efficient cost optimized scheduling for spot instances in heterogeneous cloud environment," *Future Generation Computer Systems*, vol. 84, pp. 11–21, 2018.
- [12] K. Deb, *Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction*. London: Springer London, 2011, pp. 3–34.
- [13] S. Seifollahi-Aghmiuni and O. Bozorg Haddad, "Multi objective optimization with a new evolutionary algorithm," *Water Resources Management*, vol. 32, no. 12, pp. 4013–4030, Sep 2018.
- [14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.
- [15] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, ser. CLOUD '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 236–243.
- [16] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1344–1357, May 2016.
- [17] M. Zhang, H. Li, L. Liu, and R. Buyya, "An adaptive multi-objective evolutionary algorithm for constrained workflow scheduling in clouds," *Distrib. Parallel Databases*, vol. 36, no. 2, pp. 339–368, Jun. 2018.
- [18] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.
- [19] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *2012 IEEE 8th International Conference on E-Science*, Oct 2012, pp. 1–8.
- [20] J. J. Durillo, H. M. Fard, and R. Prodan, "Moheft: A multi-objective list-based method for workflow scheduling," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, Dec 2012, pp. 185–192.
- [21] D. E. Goldberg and R. Lingle, Jr., "Alleleslociand the traveling salesman problem," in *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, pp. 154–159.
- [22] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682–692, Mar. 2013.

Thanh-Phuong Pham received the Master degree at faculty of computer science and engineering, University of Technology, Vietnam in 2011. He received the second master in computer science from University of Nice Sophia Antipolis, France, in 2012. He is a PhD student in the Distributed and Parallel Systems Group at the University of Innsbruck, Austria. His research interests include fault tolerance, machine learning, data analysis, performance modelling and workflow scheduling for distributed systems.

Thomas Fahringer received his PhD degree from the Vienna University of Technology in 1993. Since 2003, he has been a full professor of computer science at the Institute of Computer Science, University of Innsbruck, Austria. His main research interests include software architectures, programming paradigms, compiler technology, performance analysis, and prediction for parallel and distributed systems. He is a member of the IEEE.