A Provisioning Approach of Cloud Resources for Dynamic Workflows

Fairouz Fakhfakh
ReDCAD Laboratory
FSEGS, University of Sfax
B.P. 1088, 3018 Sfax, Tunisia
fairouz.fakhfakh@redcad.org

Hatem Hadj Kacem
ReDCAD Laboratory
FSEGS, University of Sfax
B.P. 1088, 3018 Sfax, Tunisia
hatem.hadjkacem@redcad.org

Ahmed Hadj Kacem
ReDCAD Laboratory
FSEGS, University of Sfax
B.P. 1088, 3018 Sfax, Tunisia
ahmed.hadjkacem@fsegs.rnu.tn

***Abstract*—Workflow technologies have become an efficient mean for the development of different applications. One well-known challenge for executing workflows on cloud computing is the resources provisioning. The latter consists in making an appropriate decision when mapping tasks to resources considering multiple objectives that are often contradictory. The problem of resources provisioning for workflow applications in the cloud has been widely studied in the literature. However, the existing works didn't consider the change in workflow instances at runtime. This functionality has become a major requirement to deal with unusual situations and evolutions. In this paper, we present a first step towards the resources provisioning for a dynamic workflow in the cloud. In fact, we propose a provisioning algorithm which takes into account some constraints. After that, we extend it in order to support the dynamic changes of workflow. Our algorithm is evaluated using CloudSim simulator. The different experiments that we present show the efficiency of our approach in terms of financial execution cost and overhead.**

Keywords*—Cloud computing, resources provisioning, dynamic workflow.

I. Introduction

Cloud computing has emerged as a new technology which provides large amounts of computing and data storage capacity to its users. It aims to overcome many problems arising from the rapid evolution of enterprises and the growth of their data. Currently, cloud environments are making use of virtualization technologies. Instead of running programs on an individual desktop computer, everything is hosted in the cloud. This latter is a model enabling on-demand network access to a shared pool of configurable computing resources (storage, applications, services, etc.). It delivers infrastructure, platform and software as services. The users can consume these cloud services based on a Service Level Agreement (SLA for short) which defines their required Quality of Service (QoS for short) parameters, on a "pay-as-you-go" basis.
In this context, the problem of resources provisioning which consists in mapping tasks to resources by satisfying QoS requirements is mainly addressed. In a cloud environment, this problem has become more complex, especially in the case of complex jobs like workflows. In fact, relationships among tasks must be considered during the resource provisioning. In addition, there are multiple types of resources provided by many commercial clouds. These resources are in units of virtual machines (VMs for short) instances. Each type has its distinct configuration (CPU type, memory size, etc.) and price. Furthermore, the wasted time fractions produced by the strategy of payment by hour should be considered. All these reasons make the resources provisioning problem hard to solve.

Workflow applications constitute a common model for describing a wide range of scientific applications in distributed systems [12]. Their interest comes mainly from the need to build upon legacy codes that would be too costly to rewrite.
A workflow is commonly represented by a Directed Acyclic Graph (DAG) in which nodes represent compute tasks and edges represent precedence and flow constraints between tasks. Workflow applications can be classified into two categories [10]:

- **Static workflows**: They are characterized by a structure and QoS constraints which do not change during the execution.
- **Dynamic workflows**: They can support change at runtime in order to face problems or modifications due to evolution. The change may be at the functional level (adding or deleting tasks, etc.) or at the non-functional level (such as time constraints).

Cloud computing has some potential benefits for executing workflow applications [11] [15]. Firstly, cloud platforms give the illusion that the available computing resources are unlimited. This means that users can acquire resources to their needs at any time. Secondly, computing resources can be elastically scaled on demand. Thus, the number of resources used to execute a workflow application can be changed at runtime according to some requirements. Thirdly, IaaS clouds offer the ability to provision resources according to the "pay-as-you-go" model where users are charged based on their consumption. As a result, executing workflow applications in the cloud can achieve better performance.

A critical challenge in integrating workflow systems with resource provisioning technologies is to determine the appropriate amount of resources required for the execution of workflows. In fact, resource under-provisioning may hurt the performance and deteriorate the QoS. In contrast, over-provisioning can result in idle instances and cause additional costs.

The resources provisioning for workflow applications has been a topic of intensive researches. Several algorithms based on heuristics have been proposed in the literature [13] [4] [9] [17] [18] [19]. Each one takes into account some metrics to ensure the resources provisioning.

2159-6190/15 $31.00 © 2015 IEEE
DOI 10.1109/CLOUD.2015.69

A major limitation of the studied workflows is their lack of support for dynamic adaptations. This functionality is essential in order to provide sufficient flexibility to cope with exceptional situations and failures. Moreover, the rapidly changing and expanding global marketplace have imposed the support of runtime modifications mainly for long-running processes [3] [25]. Such requirements will bring new challenges to resource management.

Then, we aim to address the problem of resources provisioning for dynamic workflows in the cloud. The proposed solution must take into account changes that may occur during the execution, while satisfying some performance criteria defined by the user.
In our work, we present a first step towards the resources provisioning for a dynamic workflow. Our proposed algorithm consists of three stages. In the first stage, the number and the type of VMs associated to each task are determined. The goal is to minimize the execution cost, while meeting a user specified deadline. In the second stage, we are based on the result of the first stage and we apply an adjustment process. The latter reduces the wasted time fractions produced by the first stage. It consists in consolidating tasks in the same VM instance. Afterwards, we extend our algorithm to take into account the dynamic changes of workflow. In this paper, we present the solution of two adaptation actions (adding tasks and deleting tasks at runtime) by an example.
At the best of our knowledge, this is the first work which addresses the problem of resource provisioning for a dynamic workflow in the cloud. The different experiments that we illustrate prove the efficiency of our approach in terms of financial execution cost and overhead.

This paper is organized as follows: In section II, we present an overview of our proposed approach. Section III describes our application and resources models and gives a precise problem statement. Section IV describes our algorithm of resources provisioning and details the solution proposed for some adaptation actions. Experimental results are given in section V. Section VI discusses related work. Finally, the last section concludes and outlines areas for our future research.

II. The Proposed Approach

The modifications of running workflows has become a major requirement to deal with unusual situations and evolutions. The elastic nature of cloud environment enables such dynamic workflow to be enacted more efficiently since it facilitates the changing of resource quantities at runtime. According to [16], cloud may be used as an effective accelerator to treat changing computational requirements as well as that of QoS constraints (e.g., deadlines) for a dynamic workflow. Depending on the requirements of the workflow, resources can be added or released at runtime on demand. Thus, the cloud can improve the application time-to-completion and handle unexpected situations. In contrast, using an environment with fixed resources leads to a poor performance.
Then, our goal is to address the problem of Resources Provisioning For Dynamic Workflow applications in the cloud (RP4DW for short).

The dynamic adaptation strategies of workflows are based on defined rules. These rules are often in an event-condition-action (ECA) format [22] as illustrated in Listing 1.

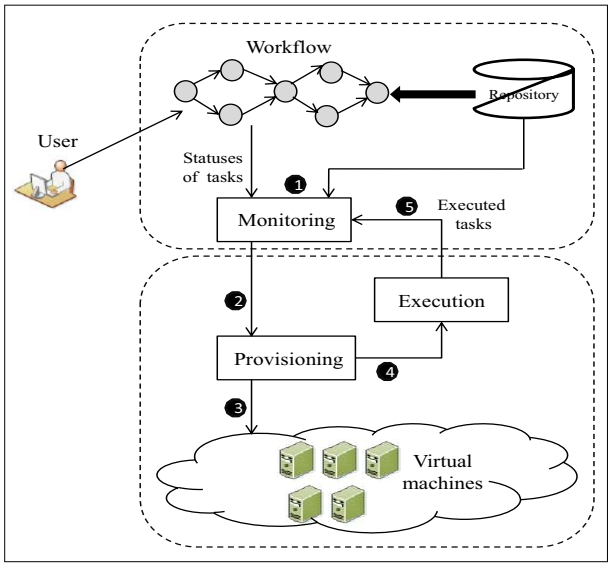

Fig. 1. Overview of our approach

Listing 1. ECA rules

```
WHEN exceptional event
WITH condition
THEN adaptation action
VALID_TIME time period
```

The event-condition part specifies which event constitutes an exceptional event and under which condition. The action part states the actions to be performed on a workflow instance to cope with the event, for example, which activities have to be dropped or added.

Figure 1 gives a global overview of the architecture that we have proposed in order to ensure resources provisioning for a dynamic workflow. The latter uses ECA rules stored in a repository which associates the adaptation actions to each exception.
In order to ensure the resources management for this type of workflow, it is necessary to store also the resources estimations of the adaptation actions in the repository.
The monitor continuously tracks the workflow in order to detect if an exception has been raised during the execution. It can acquire the statuses of workflow tasks. In fact, during the execution of a workflow, a task can have different statuses.
At first, all workflow tasks are *'unmapped'*. A task has the status *'mapped'* if it is assigned to a resource but has not been submitted. After that, a task becomes *'ready'* if all their task predecessors have finished their execution. When a *'ready'* task is submitted to its adequate resource and begins its running, it has the status *'running'*. Once a task has finished its execution and the result is ready for use or transfer, it has the status *'finished'*.
When an exception is triggered, the monitor retrieves the resources estimation from the repository (this step is depicted by the label ❶). Then, it asks the resources provisioning service (RPS for short) to recompute the mapping solutions for tasks which have not yet been started their execution ❷. After that, the RPS requests the acquisition of VMs to resources manager only for the ready tasks ❸. Once these tasks are executed ❹,

a notification is sent to the monitor ❺ in order to update the statuses of tasks.
The different components of our proposed architecture aim to assign each task to resources for execution and adjust the allocated resources in order to meet the changing requirements of workflow at runtime. This adjustment must be performed in an automatic and a rapid manner.

We are interested in this paper on the provisioning level. We give in the next section a precise problem statement. Then, we propose a provisioning algorithm and we extend it to handle the case of adding and deleting tasks at runtime.

III. Problem statement

In this section, we state the assumptions on application model and resources model. Then, we describe the performance criteria of our provisioning problem.

A. Application model

In our work, a workflow application is modeled as a directed acyclic graph G(T,E), where T =T_1, T_2,..., T_N is the set of N tasks, and E represents a set of dependencies. We assume that G has a single entry task and a single exit task. An edge $e_{i,j}$ of the form (T_i, T_j) exists if there is a precedence constraint between T_i and T_j. In this case, T_i is called the parent task of T_j and T_j is the child task of T_i. Based on this definition, a child task cannot be executed until all of its parent tasks are completed. A sample workflow is shown in Figure 2.
The data transfer time between two tasks T_i and T_j is labeled DT(T_i, T_j). In Figure 2, the labels of the arcs denote the data transfer times in minutes.
In our work, each workflow has a deadline constraint (D) associated to it. A deadline is defined as a time limit for the execution of the workflow. Also, as previously said, the considered workflow can change at runtime according to some rules defined in a repository.

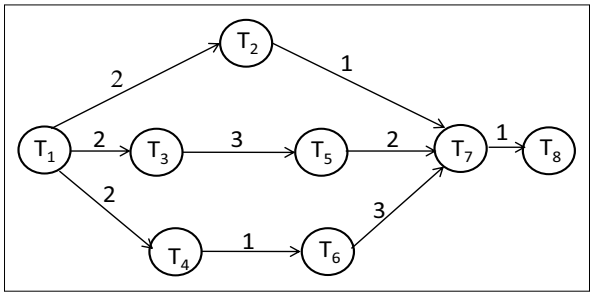

Fig. 2. An illustrative workflow example

B. Resources model

Our cloud model consists of an IaaS provider which offers virtualized resources to its clients. These VMs have different processing power and prices. We define a VM type in terms of the number of virtual CPU, memory size, the monetary cost per running hour and the estimated instance acquisition lag. VM_t denotes the t^{th} type of VM.

$$VM_t = \{CPU_t, M_t, C_t, lag_t\}, \mathbf{t} \in \{1, 2, ..., M\}$$

We assume that there is no limitation on using each resource. In fact, the user can launch any number of instances at any time.

In our work, the hourly pricing model is applied, i.e., users are charged based on the actual number of hours for which the cloud resources were used. Partial-hour consumption is always rounded up to one hour.
We assume that all instances of a service provider are located on the same physical region. This means that the bandwidth between VMs is roughly the same. Then, the data transfer time of a dependency $e_{i,j}$ only depends on the amount of data to be transferred between corresponding tasks and it is independent of VM instances which execute them. But, when task T_j and its immediate predecessor T_i are executed on the same VM instance, data transfer time DT(T_i, T_j) is equal to zero.
We note d_i^t and c_i^t respectively the execution time and cost of task T_i on VM_t.
We define the total execution time of a task T_i on VM_t by the following formula:

$$TET_i^t = lag_t + d_i^t + max(DT(T_j, T_i)),\ (j, i) \in E$$

This is the sum of the estimated acquisition lag of VM_t, the execution time of T_i on VM_t and the maximum of data transfer time between T_i and their predecessors T_j.
The total execution cost C_i^t of task T_i on a VM of type VM_t is defined as follow:

$$C_i^t = Nbh_i^t * C_t$$

It is the product of the number of required hours to execute T_i on VM_t (Nbh_i^t) by the monetary cost per running hour of VM_t (C_t).

C. Definition of the provisioning Problem

Our work consists in determining the required amount of cloud resources to run a dynamic workflow. The performance criterion of our provisioning algorithm is to minimize the financial execution cost, while completing the workflow before the user specified deadline. Furthermore, our solution should take into account the dynamic adaptations that may occur during the workflow execution. In fact, resources can be acquired or released at runtime to meet the dynamic changes of a workflow instance.
In the following, we detail our proposed algorithm which ensures the resources provisioning by satisfying our performance criterion. Then, we give two examples which address the dynamic adding and deleting of tasks.

IV. Proposed algorithm

In this section, we describe our algorithm of resources provisoning for a static workflow. Then, we extend it to take into account some dynamic changes. We detail our proposed solution in case of adding and deleting tasks at runtime.

A. Static workflow case

Our proposed algorithm consists of two phases. The first phase consists in determining the number and the type of VMs which must be allocated to each task while minimizing the

cost and respecting a deadline constraint (D). The problem is formally modeled and solved using CPLEX[1] to obtain a solution of mapping tasks to resources.
In the second phase, an adjustment process is used to reduce the cost. It takes into account the idle time fractions produced by the model of payment by hour.
In our previous work [14], we have presented the assignment results of an example of workflow before and after applying the adjustment process.

1) Assignment algorithm of tasks to VMs: Our assignment problem is modeled as a mixed integer linear program (MILP) and solved by CPLEX in order to determine the VM type of each task. $X_i^t = 1$ means that the task T_i is assigned to VM of type t, otherwise $X_i^t = 0$.
f_i denotes the finish time of task T_i. Our MILP problem can be formally described as follows (an objective function (1) and some constraints (2 to 7)):

$$\min \sum_{i=1}^{N} \sum_{t=1}^{M} C_i^t \quad (1)$$

$$S.t \sum_{t=1}^{M} X_i^t = 1; 1 \leq i \leq N \quad (2)$$

$$f_i + \sum_{t=2}^{M} X_j^t * (d_j^t + lag_t) + DT(T_i, T_j) \leq f_j, \quad \forall (i,j) \in E \quad (3)$$

$$f_1 \geq \sum_{t=1}^{M} d_1^t X_1^t \quad (4)$$

$$X_i^t \in \{0,1\}, 1 \leq t \leq M \quad (5)$$

$$d_i^t \in \mathbb{N}^+, 1 \leq i \leq N, 1 \leq t \leq M \quad (6)$$

$$f_N \leq D \quad (7)$$

The goal of the objective function (1) is to minimize the total execution cost of the workflow (the cost of the wasted time fractions are not included). Each task is assigned to one and only one VM type according to constraint (2). Constraints (3) and (4) guarantee the precedence constraints. Constraint (5) denotes a binary decision variable. According to constraint (6), the execution times of tasks are integers (in minutes). Finally, the deadline is met according to the constraint (7).
The tasks of the workflow are scheduled one by one. Each task is assigned to the appropriate resource and can only execute once its parent tasks have completed.

2) The adjustment process: This step is based on the assignment result of the first phase. It takes into account the wasted time fractions of VMs produced by the model of payment by hour. In fact, we propose a process of instance consolidation which aims to execute many tasks in the same VM instance. These tasks may belong to VMs having the same type or different types. Figure 3 illustrates this idea.

T_1 and T_2 are originally scheduled respectively on VM_5 during the time interval [0, 31] and VM_2 during the interval [31, 59]. The execution time includes the acquisition time of the VM which is represented by 'VM booting' in the figure. Both tasks only consume a partial instance-hour. The fact of executing the two tasks in the same instance (VM_5) eliminates

[1]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

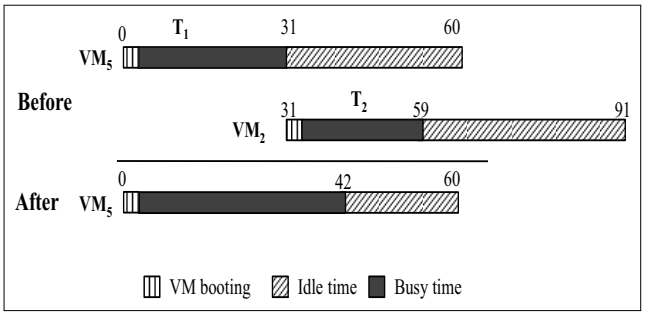

Fig. 3. Example using the adjustment process

the acquisition time of VM_2 and the data transfer time between T_1 and T_2. Also, T_2 spends only 11 minutes to be executed on VM_5. The pseudo code of the algorithm 1 describes the details of the adjustment process which aims to consolidate tasks in the same VM. It consists in verifying if each task in the workflow can run with one of its predecessors. It is an effective technique to increase resource utilization and spend less money.

Algorithm 1: Adjustment algorithm

Input: FE and D

```
1  Adjustment(FE, D)
2  begin
3      for (each task T in WF except T_1) do
4          VM_t := VM_Type_of(T);
5          /*VM_t is the VM assigned to the task T*/
           Pred_T := {P_T ∈ T; (P_T, T) ∈ E};
6          i = 1;
7          while (i <= Pred_T.size() and the
           consolidation has not been applied to the task
           T) do
8              VM_p := VM_Type_of(P_T[i]);
9              /*P_T[i] is a predecessor of the task T*/
               /*VM_p is the VM assigned to the task
               P_T[i]*/
10             if I=[FE(P_T[i], VM_p), FE(P_T[i], VM_p)+
               d_T^t] is free and
               (FE(P_T[i], VM_p) + d_T^t < D) then
11                 /*If T can run with its predecessor
                   P_T[i]*/
12                 Schedule T on the idle interval of VM_p
13             else
14                 i++;
```

The solution of mapping tasks to resources is already computed in the last phase. Our algorithm has as input the finish execution time (FE) of each task T on the assigned type of VM and the deadline of the workflow (D). The choice of the VM assigned to the input task T_1 does not change. We note $Pred_T$ the predecessor list of the task T. The proposed algorithm consists in verifying if the task T can run with one of its predecessors $P_T[i]$ (line 10).
$FE(P_T[i], VM_p)$ indicates the finish execution time of the

task $P_T[i]$ on VM_p. The variable "I" is the time interval necessary to execute the task T with its predecessor $P_T[i]$. If the time interval "I" is free and its right born does not exceed the deadline, then T can be executed in the VM assigned to $P_T[i]$ (noted VM_p). The tasks T and $P_T[i]$ can have the same type or different types of VM.

B. Dynamic workflow case

We give in this section two adaptation examples of the workflow in Figure 2 and the necessary changes at provisioning level. In this paper, we are not interested in the event and the condition under which the exception was raised. We take into account only the adaptation actions.

1) Adding tasks: In this first example, we suppose that an exception is detected at the level of task T_2. According to an ECA rule, this exception requires the adding of two tasks T_9 and T_{10} during the workflow execution and after executing T_2. Figure 4 shows the workflow after adaptation. Each added task is characterized by its predecessor tasks, its successor tasks, the data transfer time with its successors and its execution time on all types of VM. These informations are stored in KB'.

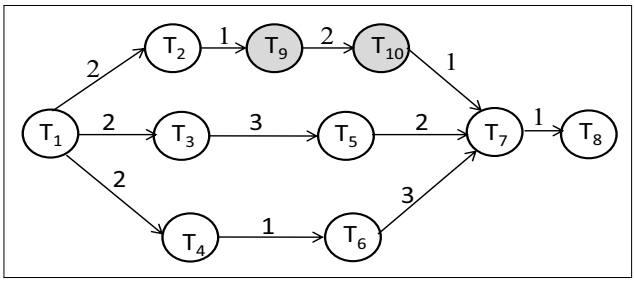

Fig. 4. Example of adding tasks at runtime

At provisioning level, it is necessary to recompute the mapping solutions of unexecuted tasks (for example: T_5, T_6, T_7 and T_8) and those added (T_9 and T_{10}) and reassign them to the appropriate types of VMs. The proposed solution takes into account the finish times of tasks already executed to satisfy the deadline constraint. Once the mapping solutions are calculated, we apply the consolidation process to the newly assigned tasks in order to reduce the cost.

2) Deleting tasks: In this example, we suppose that an exception is raised at the level of the task T_4. According to an ECA rule, this exception requires the deletion of task T_6 (Figure 5).

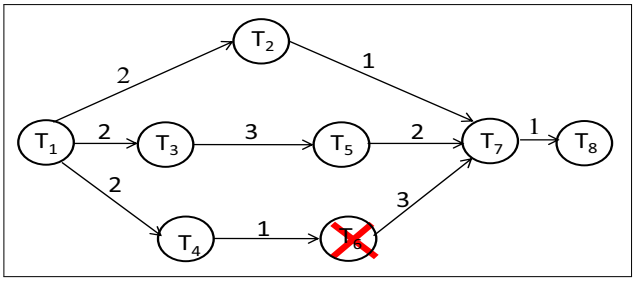

Fig. 5. Deleting task at runtime

Our MILP problem requires that T_7 starts running after their predecessors (T_2, T_5 and T_6) complete their execution and their data transfer.
In this case, if f_6 + DT(T_6, T_7) $\geq$ f_2 + DT(T_2, T_7) or f_6 + DT(T_6, T_7) $\geq$ f_5 + DT(T_5, T_7) then the task T_7 and T_8 can be reassigned to a less costly resources.

V. EXPERIMENTS

In this section we present the experiments conducted in order to evaluate the performance of our proposed approach. We used the CloudSim framework [8] to simulate a cloud environment. For solving the problem of mapping tasks to VM types, the ILOG CPLEX v12.6 is used in order to resolve the constructed MILP model of the problem formulated in section IV.

A. Experimental setup

We modeled an IaaS provider offering a single data center and five different types of VMs.The configurations and prices of VMs used in this work are presented in Table I. We used a VM billing interval of one hour and a boot time of 60 seconds.

TABLE I. CONFIGURATIONS AND PRICES OF VMS

VM type	Number of CPUs	Memory (GiB)	Cost ($ per hour)
small	1	2	0.036
meduim	1	4	0.133
large	2	8	0.266
xlarge	4	15	0.532
2xlarge	8	30	1.064

B. Experimental result

In order to evaluate our approach, we designed some experiments with the following goals:

- Evaluate the overhead of our algorithm
- Demonstrate the efficiency of our approach in term of cost reduction

1) Overhead of our approach: One of the potential problems with a dynamic approach of provisioning is the high runtime overhead. We will show in our experiments that this not the case with our approach. That means that the overhead caused by our algorithm is quite small. For demonstrating this, we compared two different executions as follows. In the first case, we apply an optimal execution where no exception is raised during execution. In the other case, we use an adaptive execution and we assume that the tasks associated to the raised exceptions are generated randomly. Also, we suppose that the maximum number of the raised exceptions is one-third the number of tasks and the number of added tasks at runtime does not exceed three. Thus, we need to recompute the mapping solutions for tasks which have not yet been executed. We report the measured execution time for the two versions of algorithms by varying the number of workflow tasks. As the number of tasks increase, the execution time of the two algorithms also increases. For each number of tasks, the simulation results displayed in Figure 6 are the averages of 50 workflows generated randomly. The overhead is around 1.29%, 8.14%, 10.97% and 11,65% for the 4 cases. But, overall, this overhead is quite small.

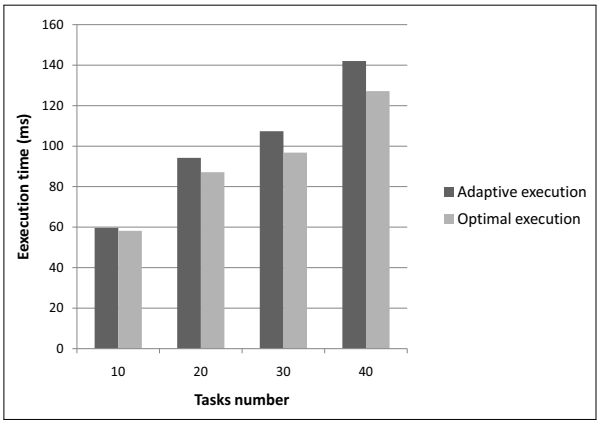

Fig. 6. Comparing the execution time of an adaptive execution with an optimal execution

2) Resource cost: In this part, we present the experimental results of our approach focusing on its performance in term of the financial cost. In these experiments, we have generated randomly 50 workflows having 40 tasks. The tasks parameters (length, file size, output size) are random values. Also, for each workflow, the tasks associated to the raised exceptions are generated randomly. We present our evaluation in the case of two scenarios: adding and deleting tasks at runtime.

- **Adding tasks**: We defined two experiments that reflect the objective that we want to highlight. In the first experiment, we have used our algorithm and we have varied the percentage of adding tasks (or exception triggering) at runtime. This percentage takes different values {10%, 20%, 30%, 40%, ..., 90%}. We have taken into account the average of the reassignment time caused by the raised exceptions. This value is already computed in the last experience (Figure 6). In the second experiment, we have used a naive solution that determines the assignment of tasks to VMs before the workflow execution.
  From Figure 7, we notice that our algorithm performs significantly better than the naive solution. In fact, our solution makes a more intelligent mapping of tasks to resources since it acquires resources only for tasks that will be executed.
- **Deleting tasks**: The experiments performed in this scenario are similar to the last experiments. In the first experiment, we have varied the percentage of deleting tasks at runtime. We have considered the average of the reassignment time due to the raised exceptions.
  The second experiment consists in determining the mapping solutions without taking into account that some tasks can be deleted at runtime. That means that the resources amount is determined before execution. The results provided in Figure 8 indicates the efficiency of our approach in term of cost reduction.

VI. Related work

The resources provisioning problem for workflow applications in the cloud has been widely studied over the recent years [13], [20], [1], [6], [18], [7], [23]. In fact, the type and

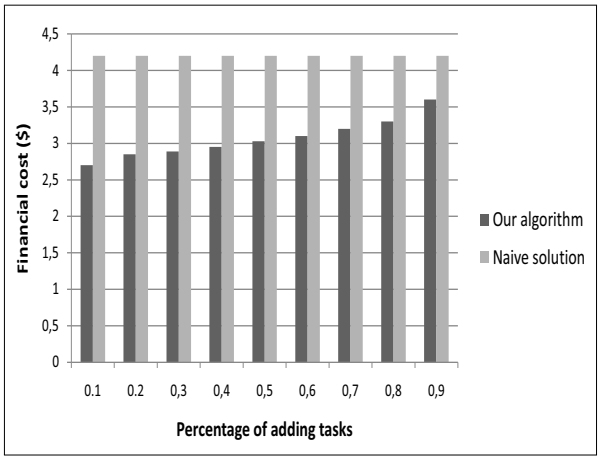

Fig. 7. Financial cost comparison of our approach with a naive solution in case of adding tasks

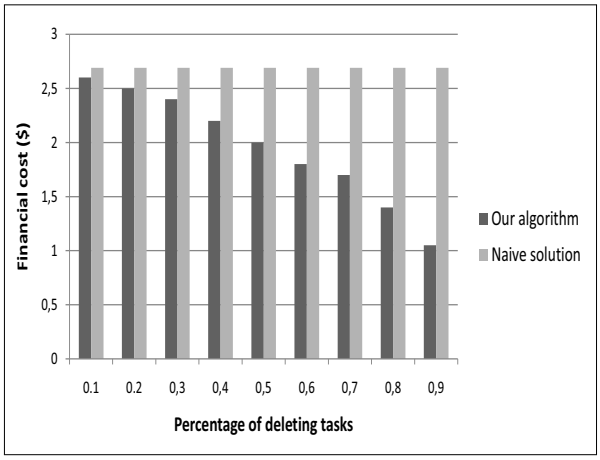

Fig. 8. Financial cost comparison of our approach with a naive solution in case of deleting tasks

the number of the allocated resources affect the execution time of workflow and determine the financial cost.

Most proposed solutions are interested in minimizing time and cost of task execution. For instance, Bessai et al. [4] propose a strategy to tackle the allocation and scheduling workflow problems in cloud environments. This strategy is based on three complementary bi-criteria approaches. It takes into account the overall execution time and the cost incurred by using a set of resources. The first approach enables to minimize the execution cost. The second attempts to reduce the overall completion time. Finally, the third approach combines the objectives of the above approaches by selecting only the non-dominated solutions.

The major contribution of Pandey et al. [20] is a scheduling heuristic based on the Particle Swarm Optimization (PSO) method. It aims to minimize the execution cost of running the workflow application on the cloud. PSO takes into account both the computation cost and the data transfer cost between tasks. The authors propose a dynamic algorithm which updates the communication costs in every scheduling loop, based on

the current network and resource conditions. The idea of the proposed algorithm is to compute the PSO mapping and assign only the ready tasks to the compute resources. These steps are repeated until all the tasks in the workflow are scheduled. Several improvements of this algorithm are proposed like CSO [5], RDPSO [24] and PSO_HOM [21].

Other approaches focus on optimizing the cost of workflow execution while meeting a user-defined deadline. One of the most recent works in this area has been introduced by Abrishami et al. [1]. They extend a previous algorithm for deadline constrained workflow scheduling in Grid environment [2]. This extension designs two new algorithms (IC-PCP and IC-PCPD2) for the IaaS cloud environment. These algorithms aim to minimize the execution cost of a workflow, while completing the workflow before the user's specified deadline. They take into account the data transfer time between tasks that can affect both the performance and the cost mainly for data-intensive workflows.

Additionally, the approach cited in [6] introduces a PBTS (Partitioned Balanced Time Scheduling) method to estimate the minimum resources required to execute a workflow within a deadline constraint. The application's running time is divided into time-partitions whose length is equal to the time charge unit specified by the cloud provider. PBTS iterates all time-partitions of the application one by one in the time order at runtime. It has three phases to determine the resource capacity of the next time-partition. First, PBTS determines the set of tasks for the next partition based on the approximate resource capacity. Then, it estimates the minimum number of required resources and the schedule of the selected tasks for the time-partition. Finally, it allocates actual resources as estimated and executes the selected tasks on the resources.

A new auto-scaling mechanism for an ensemble of workflows is described by Mao et al. [18]. The goal is to ensure that all jobs finish before their respective deadlines by using resources which cost less. They use deadline assignment techniques to calculate an optimized resource plan for each job and determine the number of instances. They consider the problem of the wasted partial instance hours. In fact, cloud VMs are currently billed by instance hours. Thus, partial-hour consumption is always rounded up to one hour. Then, a process of instance consolidation is proposed in order to avoid partial instance-hour waste. It consists in putting tasks on the same instance even if some tasks may not run the better cost-efficiency on this machine. What's more, their proposed auto-scaling mechanism is aware of the acquisition time of every instance. In fact, every instance takes some time to be ready to be used. Such waiting time is called "instance acquisition lag".

Based on the latest observations, the approaches addressing the problem of resources provisioning in the literature are destined for static workflow applications. However, the major drawback of this model is its rigidity. It lacks supporting dynamic adaptations which consist in changing workflows at the instance level. This functionality has become a major requirement especially for a long-running process. It helps to cope with unusual situations and failures that may occur during the execution of workflow instance. In addition, it can handle new situations to manage the rapid changes demanded by the dynamic nature of the marketplace. Examples of such changes include the market evolution, changes in the customer behavior and strategy shifts. In fact, the economic success of enterprises increasingly depends on their ability to react to changes in their environment in a quick and flexible way [3].
Then, the main contribution in our work is to address the problem of resources provisioning for a dynamic workflow in the cloud. The proposed solution takes into account the changes at workflow level, while satisfying some constraints defined by the user.
Trying to attain our aims, Zhu et al. present in [26] an approach of resources provisoning for adaptive applications. They consider the problem where there is a fixed time-limit and a budget constraint for a particular task. Nevertheless, this work is not suitable for a workflow application. In fact, it considers a single application while our approach takes into account the dependency relationships between workflow tasks.

VII. Conclusion and future work

In this paper, we have highlighted the inability of the resources provisioning solutions to deal with changes at runtime. In fact, the support of workflow adaptation has become crucial for emergency management during execution. Moreover, the rapidly changing and expanding global marketplace have imposed the support of runtime modifications mainly for long-running processes.
Our proposed algorithm enables to determine the resource amount to execute a workflow on IaaS cloud. It aims to minimize the execution cost while meeting a user defined deadline. To further reduce the cost, we used an adjustment process which takes into account the wasted time fractions. Then, we have presented a first step towards the provisioning for a dynamic workflow by addressing only the case of adding and deleting tasks at runtime. The different experiments that we have presented show the efficiency of our solution in terms of financial cost and the overhead caused by the remapping time.

In the current work, we assume that a workflow has a DAG structure which allows the type of branching ANDsplit. As an ongoing work, we are working on dealing with other structures such as ORsplit and XORsplit. As a future work, we plan to address other adaptation actions. We also intend to improve our algorithm for the real cloud environments, such as Amazon. The most important problem in the real environment is the inaccuracy of the estimated execution and transmission times. It is interesting also to choose a case study supporting dynamic workflows.

References

[1] S. Abrishami, M. Naghibzadeh, and D. J. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[2] S. Abrishami, M. Naghibzadeh, and D. Epema, "Cost-Driven Scheduling of Grid Workflows Using Partial Critical Paths," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1400–1414, Aug 2012.

[3] M. Adams, "Dynamic Workflow," in *Modern Business Process Automation*. Springer, 2010, pp. 123–145.

[4] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan, "Bi-criteria Workflow Tasks Allocation and Scheduling in Cloud Computing Environments," in *IEEE Cloud*, 2012, pp. 638–645.

[5] S. Bilgaiyan, S. Sagnika, and M. Das, "Workflow scheduling in cloud computing environment using cat swarm optimization," in *Advance Computing Conference (IACC), IEEE International*. Gurgaon, India: IEEE Computer Society, February 21-22 2014, pp. 680–685.

[6] E. Byun, Y. Kee, J. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011–1026, 2011.

[7] Z. Cai, X. Li, L. Chen, and J. N. D. Gupta, "Bi-direction Adjust Heuristic for Workflow Scheduling in Clouds," in *Proceedings of the 19th International Conference on Parallel and Distributed Systems (ICPADS)*. Seoul, Korea: IEEE, December 15-18 2013, pp. 94–101.

[8] 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," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[9] E. Caron, F. Desprez, A. Muresan, and F. Suter, "Budget constrained resource allocation for non-deterministic workflows on an iaas cloud," in *Proceedings of the 12th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ser. ICA3PP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 186–201.

[10] P. Dadam and M. Reichert, "The ADEPT project: a decade of research and development for robust and flexible process support," *Computer Science - R&D*, vol. 23, no. 2, pp. 81–97, 2009.

[11] E. Deelman, "Grids and Clouds: Making Workflow Applications Work in Heterogeneous Distributed Environments," *International Journal of High Performance Computing Applications*, vol. 24, no. 3, pp. 284–298, 2010.

[12] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[13] F. Fakhfakh, H. Hadj-Kacem, and A. Hadj-Kacem, "Workflow Scheduling in Cloud Computing: A survey," in *the 18th IEEE International conference on Enterprise Distributed Object Computing Conference Workshops, EDOC, Ulm, Germany*, September 01-05 2014, pp. 372–378.

[14] ——, "RP4DW: Resources Provisioning For Dynamic Workflows," *Studies in Computational Intelligence, Springer*, 2015.

[15] G. Juve, E. Deelman, K. Vahi, G. Mehta, and B. Berriman, "Scientific workflow applications on Amazon EC2," in *Cloud Computing Workshop in Conjunction with e-Science*. IEEE, 2009.

[16] H. Kim, Y. el Khamra, S. Jha, and M. Parashar, "Exploring application and infrastructure adaptation on hybrid grid-cloud infrastructure," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 402–412.

[17] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC'12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012.

[18] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC'11. New York, NY, USA: ACM, 2011.

[19] ——, "Scaling and Scheduling to Maximize Application Performance Within Budget Constraints in Cloud Workflows," in *Proceedings of the 27th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '13. Washington, DC, USA: IEEE Computer Society, May 20-24 2013, pp. 67–78.

[20] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, ser. AINA '10, Washington, DC, USA, 2010, pp. 400–407.

[21] M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioningand Scheduling Algorithm for Scientific Workflows on Clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[22] S. Urban, L. Gao, R. Shrestha, and A. Courter, "The Dynamics of Process Modeling: New Directions for the Use of Events and Rules in Service-Oriented Computing," in *The Evolution of Conceptual Modeling*, ser. Lecture Notes in Computer Science, R. Kaschek and L. Delcambre, Eds. Springer Berlin Heidelberg, 2011, vol. 6520, pp. 205–224.

[23] L. Wang, R. Duan, X. Li, S. Lu, T. Hung, R. Calheiros, and A. R. Buyya, "An Iterative Optimization Framework for Adaptive Workflow Management in Computational Clouds," in *Symposium on Parallel and Distributed Processing with Applications*, 2013.

[24] Z. Wu, Z. Ni, L. Gu, and X. Liu, "A Revised Discrete Particle Swarm Optimization for Cloud Workflow Scheduling," in *Proceedings of the sixth International Conference on Computational Intelligence and Security*, ser. CIS'2010. Nanning, China: IEEE, 2010, pp. 184–188.

[25] J. Yu, Q. Z. Sheng, J. K. Y. Swee, J. Han, C. Liu, and T. H. Noor, "Model-driven development of adaptive web service processes with aspects and rules," *Journal of Computer and System Sciences*, vol. 81, no. 3, pp. 533–552, 2015.

[26] Q. Zhu and G. Agrawal, "Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 497–511, 2012.