Bi-criteria workflow tasks allocation and scheduling in Cloud computing environments

Kahina Bessai*, Samir Youcef**, Ammar Oulamara**, Claude Godart** and Selmin Nurcan*

*University of Paris1-Panthéon- Sorbonne
Centre de Recherche en Informatique
90, rue de Tolbiac 75634 Paris, France
{name}@univ-paris1.fr

**University of Lorraine
LORIA-INRIA-UMR 7503
BP 239, Vandoeuvre-les-Nancy, France
{name}@loria.fr

Abstract—**Although there are few efficient algorithms in the literature for scientific workflow tasks allocation and scheduling for heterogeneous resources such as those proposed in grid computing context, they usually require a bounded number of computer resources that cannot be applied in Cloud computing environment. Indeed, unlike grid, elastic computing, such as Amazon's EC2, allows users to allocate and release compute resources on-demand and pay only for what they use. Therefore, it is reasonable to assume that the number of resources is infinite. This feature of Clouds has been called "illusion of infinite resources". However, despite the proven benefits of using Cloud to run scientific workflows, users lack guidance for choosing between multiple offering while taking into account several objectives which are often conflicting.**

On the other side, the workflow tasks allocation and scheduling have been shown to be NP-complete problems. Thus, it is convenient to use heuristic rather than deterministic algorithm. The objective of this paper is to design an allocation strategy for Cloud computing platform. More precisely, we propose three complementary bi-criteria approaches for scheduling workflows on distributed Cloud resources, taking into account the overall execution time and the cost incurred by using a set of resources.

I. Introduction

Cloud computing is based on resources virtualization and its business model is based on the concept of paying only for what customers use, which overcome the limitations of the traditional software sales model. In this context, IT organizations need to think in terms of managing services rather than managing devices.

In practice, Cloud service providers offer services that can be classified in three categories, [1][2] namely:

- Software as a service (SaaS) is a software distribution model in which applications are hosted by a service provider delivered as a service to consumer but without controlling the host environment. Examples of this model are the Google Apps (http://www.google.com/apps/) and Salesforce (http://www.salesforce.com/).
- Platform as a Service (PaaS) allows customers to rent virtualized servers for using existing applications or developing and testing new ones. Examples of this model are the Google App Engine (http://www.code.google.com/in1/en/appengine/) and Amazon Web Services (http://www.aws.amazon.com/) .
- Infrastructure as a Service (IaaS) allows customers to use computing resources such as storage and processing power, and they pay on a per-use basis. Amazon Elastic Compute Cloud (http://aws.amazon.com/ec2) is an example of this model.

As a result, the Cloud computing has quickly changed the way that compute resources can be used and allows users to access computer on the fly, according to the application's needs. For instance, currently Amazon's EC2 provide a Web service through which user can boot an Amazon Machine Image for creating a virtual machine that can run any desired software.

Furthermore, unlike grids, Could platforms give the illusion that the available computing resources are unlimited. This means that users can request, and are likely to obtain, sufficient resources for their need at any time. This feature of Clouds has been called "*illusion of infinite resources*" [4] [3]. The *elasticity* is one of the most important feature of Clouds which means that users can release resources on-demand. In addition for provisioning on-demand and the elasticity characteristics, unlike grids, Cloud platforms allow to users to directly allocate resources as required to schedule their computations. Due to these characteristics and the Cloud model's proven benefits, such as flexibility, scalability and cost-effectiveness, it has been rapidly gaining interest for distributing many enterprise software such as banking, e-commerce business software [6].

Generally, two most important types of workflow are distinguished, namely: i) business process workflows and ii) scientific workflows. In this paper, we focus only on the second type of workflows. The extension to the first category is under investigation. In fact, Cloud computing paradigm has also naturally and rapidly gained interest in the scientific community for workflow simulation, image processing, data analysis, etc., in many scientific disciplines, such that astronomy and bioinformatic domains [5]. Indeed, workflows are able to be used, in many situations, to express different types of scientific analysis.

Scientific workflow tasks are commonly represented as a directed acyclic graph (DAG) in which nodes represent application tasks and edges represent dependencies between these tasks [7]. One well-known challenge for executing workflow applications on Cloud is tasks allocation and scheduling, that is, making decisions for mapping computations to resource in order to optimize some performance metric, such that completion time and execution cost, and so that tasks-precedence requirement are satisfied. The tasks matching and scheduling is NP-complete in the general case [9], as well as some restricted cases [8] [7]. Therefore, it is convenient to use heuristic rather than an exact algorithm to deal with this problem.

The objective of this paper is to design allocation and scheduling strategies for Cloud computing platforms. More precisely, we consider the bi-criteria approach for the allocation and scheduling of workflow applications on Cloud. The objectives are: i) minimizing the workflow application completion time and ii) minimizing the amount cost incurred by using resources.

Before summarizing the contributions of this paper, we start by giving an overview on the approaches used to deal with bi-criterion allocation and scheduling problems that are extensively considered in the literature. Mainly three streams are considered [10] [11] [12]:

- Aggregation approach: both criteria f_1 and f_2 are aggregated into one composite function F$(f_1(\pi), f_2(\pi))$ for some given F, usually an additive function, where π is the schedule. In this approach the bi-criteria problem is transformed into a mono-criterion problem.
- ϵ-approach: in this approach the criterion f_2 is minimized under the additional constraint $f_1 \leq \epsilon$, where ϵ is a given value fixed by decision maker. This approach is also viewed as a lexicographical approach in which the more important criterion f_1 is minimized and let f_1^* the obtained optimal value, and then the second criterion f_2 is minimized under the additional constraint $f_1 \leq f_1^*$, here $\epsilon = f_1^*$.
- Pareto approach: this approach is applied when no criterion is dominant, in that case no-dominant solutions are computed, also called Pareto solutions, that will be introduced in Section IV.

By studying the possible transformation of the addressed problem in this paper into mono-criteria one, we remarked that no criterion is dominant. Therefore, the Pareto approach seems us the most appropriate for addressing our problem.

To summarize, in this paper we make the following contributions:

1) We formalize a model for workflow tasks matching and scheduling in Cloud environment.
2) We propose a first approach for workflow tasks matching and scheduling based on minimizing the cost execution incurred by using a set of resources.
3) We propose a second approach based on the overall completion time.
4) Finally, we present a third approach combining the above objectives.

The remainder of the paper is organized as follows. Section II presents related works and compare them with our proposition. Section III describes our system model and the objective functions. In Section IV, we detail the proposed approaches for workflow tasks matching and scheduling. Experimental results are presented in Section V. Section VI summarizes the contributions and outlines areas for future research.

II. Related work

Due to its importance on performance, the workflow tasks allocation and scheduling problem on heterogeneous computing environment such as grid has been intensively explored. Beside, the tasks allocation to compute resources is an NP-complete problem in the general form [9]. So, past works have proposed heuristic driven approaches for scheduling workflow applications [13] [14]. These heuristics cannot be applied in Clouds computing environments because they require a bounded number of resources.

Therefore, allocation and scheduling workflow tasks in Cloud has gained popularity in recent times and few algorithms are proposed to deal with this problem [15] [16] [17] [18]. In [15], the authors developed a model that uses particle swarm optimization (PSO) for task-resource mapping to minimize the overall cost of execution such that it completes within deadline that user specifies. To tackle the problem of choosing resource among different cloud providers, a binary integer program is proposed in [15], where the objective is to minimize the total infrastructure capacity under budget and load balancing constraints. In [16], the authors presented a model for formulation of the generalized federated placement problem and application of this problem to load balancing and consolidation within a cloud, where one cloud can subcontract workloads to partnering clouds to meet peaks in demand. They used an Integer Program formulation of the placement program and provide a 2-approximation algorithm. In [17], the authors proposed a binary integer program formulation for cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In [18], the authors proposed a set of heuristics to cost-efficiently schedule deadline-constrained computational applications on both public cloud providers and private infrastructure. Although, most of these studies consider unbounded number of resources, however, they convert the initial problem (bi-cretiria) to the $\epsilon-$ constraints problem.

To overcome these limitations, we propose new approaches which are distinct from the related work as they take into account Clouds elasticity feature, which means that users can rent and release resources according to the need of its applications. Moreover, our approaches consider the overall completion time and the execution cost together, where the problem of tasks allocation and scheduling problem has not been converted neither to mono-criterion nor to $\epsilon-$constraints problem. Therefore, the matching and scheduling problem described here does not appear to have been studied before.

III. Workflow tasks allocation and scheduling problems formulation in Cloud computing

In the following, we start by refining the problem definition and then present the cost and time objective functions that we consider in this work.

Recall that the main scope of this paper is to deal with the workflow tasks allocation and scheduling problem in the Cloud environment. Scientific workflow applications are commonly represented as a directed acyclic graph (DAG). Formally, a workflow application is a DAG represented by $G = (T, E)$, where $T = \{t_1, ..., t_n\}$ is a finite set of tasks. E represents the set of directed edges. An edge (t_i, t_j) of graph G corresponds to the data dependencies between these tasks (the data generated by t_i is consumed by t_j). Task t_i is called the immediate parent of t_j which is the immediate child task of t_i. Note that the child task cannot be executed until all of its parents tasks are completed. In a given graph, a task without any precedents is called an input task, denoted t_{input} and a task without successors is called an exit task, denoted t_{exit}. Let $Data$ be a $n \times n$ matrix of communication data, where $data[i, j]$ is the amount of data required to be transmitted from task t_i to task t_j.

Most of the workflow tasks allocation and scheduling algorithms require single input (t_{input}) and single exit (t_{exit}) task graphs. So, if there is more than one input (exit) task, they are connected to a zero-cost (and zero-time) `pseudo-input` `(-exit)` task with zero communication cost and time, which does not affect the allocation and the schedule. This modification allows to obtain a DAG with only one input and one exit tasks.

Moreover, a task is an indivisible unit of work and is non-preemptive. Figure 1 (left side) shows an example of workflow application defined by ten tasks, which are represented as nodes. The dependencies between tasks are represented as arrows. The initial task may have an input file denoted (`input.file`) and the exit task produces an output file (`output.file`).

To execute a given workflow application (DAG), an infinite set of resources (virtual machine) can be used on-demand. The latters are represented as a directed graph denoted RG. Formally, a resources graph is represented by $RG = (VM, V)$, where $VM = \{VM_1, ..., VM_m\}$ is a finite set of virtual machines types that define a virtual machine images and a data center locations. However, we consider that there is enough virtual machines for each type. Thus, a costumer can request and obtain sufficient virtual machines at any time. This assumption is reasonable in rather Cloud computing environment because it gives for user an "illusion of infinite resources". When there is no ambiguity, we omit term type and use virtual machine instead virtual machine type. V represents the set of directed edges. Each edge is denoted (VM_i, VM_j) corresponding to the link between these virtual machines. Let B be a $m \times m$ matrix, in which $B[i, j]$ is the bandwidth between virtual machines VM_i and VM_j, where $B[i, i] \longrightarrow \infty$ means that there is no transfer data. Figure 1 (right side) shows an example of resources graph with real-life measurement of data transfer speeds (bandwidth) between different data centers of the Cloud provider Amazon.

Let $VM(t_j)$ denotes the virtual machine that executes task t_j. The transfer time $TT(VM(t_i), VM(t_j))$, which is for transferring data from task t_i (executed on $VM(t_i)$) to task t_j (executed on $VM(t_j)$) is defined by:

$$TT(VM(t_i), VM(t_j)) = \frac{data[i,j]}{B[VM(t_i), VM(t_j)]}$$

Let ET be a $n \times m$ execution time matrix in which $ET(t_i, VM_j)$ gives the execution time estimation to complete task t_i on virtual machine VM_j. The real execution of a given task depends on the amount of input generated data.

Let UEC be a $m-$dimensional unit execution cost vector, where $UEC(VM_j)$ represents the cost per time unit incurred by using the virtual machine VM_j. Let EC be a $n \times m$ execution cost matrix in which $EC(t_i, VM_j)$ gives the execution cost to complete task t_i on virtual machine VM_j defined by:

$$EC(t_i, VM_j) = ET(t_i, VM_j) \times UEC(VM_j)$$

We assume that the data transfer cost $TC(VM(t_i), VM(t_j))$, which is the cost incurred due to the transfer of data from task t_i (executed on $VM(t_i)$) to task t_j (executed on $VM(t_j)$), is defined by:

$$TC(VM(t_i), VM(t_j)) = data[i, j] \times (C_{out}(VM(t_i)) + C_{in}(VM(t_j)))$$

where $C_{out}(VM(t_i))$ and $C_{in}(VM(t_j))$ represent respectively the cost of transferring data from $VM(t_i)$ and the cost of receiving data on $VM(t_j)$. The transferring data cost is determined by mutual agreement between the consumer and the provider in the Service Level Agreement (SLA) [1].

A. *Time objective function*

Before presenting the time objective function, it is necessary to define the ST and FT attributes, which are derived from a given partial allocation and scheduling of tasks to virtual machines (i.e. a task t_i is assigned to virtual machine $VM(t_i)$). The partial allocation and scheduling refers to the fact that for each task the ST and FT values are computed using only the tasks that must be performed before it as shown in the following. More precisely, $ST(t_j)$ and $FT(t_j)$ are the *earliest* start execution time and the *earliest* finish execution time of task t_j. For the input task t_{input}:

$$ST(t_{input}) = 0,$$

$$FT(t_{input}) = ST(T_{input}) + ET(t_{input}, VM(t_{input})) \quad (1)$$

For the other tasks in the graph, the ST and the FT values are computed recursively, starting from the initial task, as shown in Equation 2 and Equation 3. In order to compute the FT of a task t_j, all immediate predecessor tasks of t_j must have been assigned and scheduled with the consideration of the transfer time:

$$FT(t_j) = ST(t_j) + ET(t_j, VM(t_j)) \quad (2)$$

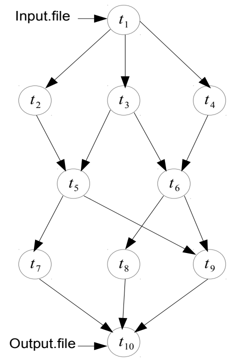
$$ST(t_j) = \max_{t_p \in pred(t_j)} \{FT(t_p) + TT(VM(t_p), VM(t_j))\} \quad (3)$$

where $pred(t_j)$ is the set of immediate predecessors of task t_j.

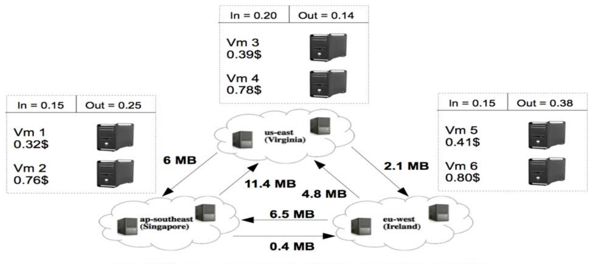
After all tasks in a graph are scheduled, the schedule length (i.e., the overall completion time) will be the finish time of the exit task. The schedule length, also called *makespan*, is defined as:

$$makespan = FT(t_{exit}) \quad (4)$$

Therefore, the *time objective function* is to determine the assignment of tasks of a given workflow application to virtual machines such that its schedule length is minimized.



VM / Task	VM_1	VM_2	VM_3	VM_4	VM_5	VM_6
t_1	13	19	18	11	8	16
t_2	24	14	13	19	21	10
t_3	24	8	14	17	18	11
t_4	21	70	16	18	21	19
t_5	8	5	14	10	18	6
t_6	18	12	11	20	18	10
t_7	7	15	11	21	20	19
t_8	8	17	9	14	13	24
t_9	12	13	21	6	18	14
t_{10}	25	11	25	11	13	8



Real-life measurement of data transfer speeds between different data centers of the Cloud provider Amazon [4]

$data_1$	$data_2$	$data_3$	$data_4$	$data_5$	$data_6$	$data_7$	$data_8$	$data_9$	$data_{10}$
100	56	512	418	98	540	890	919	245	645

Fig. 1. A workflow application and a set of virtual machines

B. *Cost objective function*

In the following, we focus on the cost objective function which is the total expense for workflow tasks execution including i) the tasks execution cost and ii) the data transfer cost between the used virtual machines. The cost function is a structure independent criterion defined as the sum of the costs of executing all workflow tasks, given by:

$$cost = \sum_{j=1}^{n} \left\{ EC(VM(t_j)) + \sum_{p \in pred(t_j)} TC(VM(t_j), VM(t_p)) \right\} \tag{5}$$

Thus, the *cost objective function* is to determinate the assignment of tasks of a given workflow application such that its overall execution cost is minimized.

The problem addressed in this work deals with the tasks workflow application allocation and scheduling while simultaneously minimizing the *makespan* and the overall *cost* execution.

As mentioned previously, this problem can be approached in several ways. In our work, we have opted for the effective solutions (Pareto solutions) computation. To achieve this objective, we propose in the following three multi-objective approaches. The first one is based on the overall *cost* execution function while the second one is conducted by the *makespan* function. Finally, we propose an approach taking into account the two functions together.

IV. Workflow tasks allocation and scheduling approaches in cloud computing context

As mentioned above, we propose in the following three algorithms for allocation and scheduling workflow application in Cloud computing environment. More precisely, the first algorithm aims to minimize the execution and communication costs (cost-based approach) using several allocation strategies. For each obtained solution the overall completion time corresponding is computed. The second algorithm proceeds similarly as the first one by attempting to minimize the execution and the communication times (time-based approach) and for each obtained solution the overall computation cost corresponding is computed. Finally, we propose a third algorithm, called cost-time-based approach, based on the obtained Pareto solutions by the two first algorithms. In other words, using the solutions produced by the cost-based and the time-based approaches only the non-dominated solutions are selected.

Therefore before giving their details, we introduce the Pareto optimization approach. A multi-objective problem can be defined as:

$$\min f(x) = (f_1(x), ..., f_n(x))$$

where $x \in X$; X is a set of feasible solutions (or decision space). A feasible solution $x \in X$ is a Pareto solution (or efficient solution) if there does not exist any $x' \in X$ such that $\forall i \in \{1, ..., n\}, f_i(x') \leq f_i(x) \ \wedge \ \exists j \in \{1, ..., n\}, f_j(x') < f_j(x)$

Figure 2 shows solutions for the minimization problem of two conflicting objectives $f_1(x)$ and $f_2(x)$. The solution x_4 dominates x_1, because both objective values of x_4 are lower than those of x_1 (i.e. $f_1(x_4) < f_1(x_1)$ and $f_2(x_4) < f_2(x_1)$). However, x_2 does not dominate x_4, since $f_1(x_2) > f_1(x_4)$. We say that x_4 and x_2 are non-dominated solutions (Pareto solutions). The set of non-dominated solutions forms a non-dominated front.

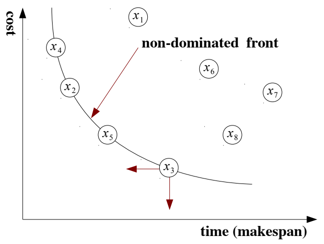


Fig. 2. Four non-dominated solution ($x_2 - x_5$) and four dominated solutions ($x_1, x_6 - x_8$) for two objectives cost and time

A. *Cost-based approach*

In the cost-based approach, we focus only on minimizing the execution and communication costs of using a set of virtual machines incurred by the execution of a given workflow. However, for each obtained feasible solution by using an allocation strategy the overall completion time corresponding is computed. Recall that the objective is to assign tasks to virtual machines respecting the precedence constraints. The cost-based approach is an application allocation and scheduling algorithm for an unbounded number of virtual machines. As mentioned previously, users can request and obtain sufficient resources at any time. The approach proposed has three major phases, namely: i) tasks sorting phase, ii) resource allocation phase and iii) Pareto selection phase. An overview of the cost-based approach is shown by Algorithm 1.

Tasks sorting phase In order to group the tasks that are independent of each other, the given workflow (DAG) is traversed in a top-down fashion to sort tasks at each level. As a result, tasks belonging to the same level do not exchange data and can be executed in parallel (because they are not related by precedence constraints). Given a DAG $G = (T, E)$, the level 1 and the last level contains respectively the input t_{input} task and the exist t_{exit} task. Level k, denoted l_k, consists of all tasks t_j such that for all edges (t_i, t_j), task t_i is in a level $k' < k$ and there exists at least one edge (t_i, t_j) such that t_i is in level $k-1$. Let L the number of levels of a given workflow.

For instance, for the tasks graph given in Figure 1, there are five levels (i.e. $L = 5$): level l_1 consists of task t_1 (initial task), level l_2 consists of task t_2, t_3, t_4 , level l_3 consists of tasks t_5, t_6, level l_4 consists of task t_7, t_8, t_9 and the level five l_5 contains task t_{10} (exit task). The line 2 of Algorithm 1 corresponds to the tasks sorting phase.

Resource allocation phase In this phase the selection of an "optimal" virtual machine for each task is decided. In other words, the virtual machine which gives minimum execution and communication costs for a task is selected and the task is assigned to that virtual machine. More precisely, given the labeling of tasks in the graph levels, the allocation process explores the graph by starting the allocation tasks of level k, where the value of k is given by the following strategies: i) top-down, iii) bottom-up and iii) mixed exploration and allocation strategy.

The top-down strategy consists of starting by the allocation of the initial task (level l_1) to the virtual machine which gives minimum execution cost. After this assignment, the graph is traversed in a top-down fashion from level 2 to level L. At level k, the task is assigned to the virtual machine which gives minimum of execution and communication costs as follows: $\forall t_i \in l_k, VM(t_i) = VM_s$ such that:

$$EC(t_i, VM_s) + \textstyle\sum_{t_h \in pred(t_i)} TC(VM(t_h), VM(t_i)) =$$

$$\min_{VM_j} \left\{ EC(t_i, VM_j) + \sum_{t_h \in pred(t_i)} TC(VM(t_h), VM(t_i)) \right\} \tag{6}$$

where $pred(t_i)$ is the set of immediate predecessors of task t_i.

The lines 3 (with $k = 1$) to 10 of Algorithm 1 corresponds to the top-dow strategy.

The bottom-up strategy consists on starting by the allocation of the finish task (the last level). After this allocation, the graph is traversed in a bottom-up fashion from level $L - 1$ to level 1. At level k, the task is assigned to the virtual machine which gives minimum of execution and communication costs as follows:

$\forall t_i \in l_k, VM(t_i) = VM_s$ such that:

$$EC(t_i, VM_s) + \textstyle\sum_{t_h \in succ(t_i)} TC(VM(t_i), VM(t_h)) =$$

$$\min_{VM_j} \left\{ EC(t_i, VM_j) + \sum_{t_h \in succ(t_i)} TC\left(VM(t_i), VM(t_h)\right) \right\} \tag{7}$$

where $succ(t_i)$ is the set of immediate successors of task t_i.

The lines 11 (with $k = L$) to 15 of Algorithm 1 corresponds to the bottom-up strategy.

The mixed strategy starts by assigning the tasks belonging to the intermediate level, i.e. $k \in \{2, ..., L-1\}$. Given starting level k, therefore the assignment of the tasks belonging to this level is only based on the execution cost $(EC(t_i), VM_j)$, given by the following equation: $\forall t_i \in l_k, VM(t_i) = VM_k$ such that:

$$EC(t_i, VM_k) = \min_{VM_j} EC(t_i, VM_j) \tag{8}$$

For the tasks belonging to the level $k' < k$ and $k' > k$, the assignment of a task t_i is impacted by the previous assignments by taking into account the both costs (execution and communication costs), using bottomp-up and top-down strategy respectively. Equation 6 and 7 are recursively applied until all the tasks are assigned using the mixed strategy for all $k \in \{2, ..., L-1\}$. In the case of $k \in \{1, L\}$, we use the top-down and bottom-up strategy respectively. Note that, equations 6 and 7 have respectively one variable $(VM(t_i))$ because $VM(t_h)$ is computed in the previous iteration.

The lines 4-17 ($k \in \{2, ..., L-1\}$) of Algorithm 1 corresponds to the mixed strategy.

Hence, for the mixed strategy we obtain $L-2$ solutions (each of them consists on the assignment of all tasks) and one solution respectively for top-down and bottom-up strategy.

Pareto selection phase Recall that at the end of the previous phase a L assignment of all tasks is obtained. In this phase, we first compute the overall completion time (using Equation 1, 2 and 3) corresponding to each assignment and then only non-dominated solutions are maintained. The lines 18 and 19 of Algorithm 1 corresponds to the Pareto selection phase.

B. Time-based approach

While the previous approach is based on minimizing the *cost* function, the time-based approach, detailed in the following, is based on the *makespan* criterion. More precisely, the time-based approach attempts to minimize the overall completion time (i.e. execution and communication time). As the cost-based approach, the time-based approach is an application matching and scheduling algorithm for an "unbounded" number of virtual machines, which has three major phases, namely: a tasks sorting phase i), ii) an allocation phase and iii) Pareto selection phase. An overview of this approach is given by Algorithm 2.

Tasks sorting phase This phase is the same as for the cost-based algorithm. Recall that this phase allows to group the workflow application tasks that are independent of each other.

Resource allocation phase The cost-based and the time-based approaches differ mainly at resource allocation phase. Indeed, the first one approach focus on minimizing the *cost* function

Algorithm 1 Cost-based approach

```
1: read the DAG, the RG and associated attributes values;
2: sort tasks at each level by traversing the DAG in a top-down fashion; // let L be
   the set of levels and $l_k$ the tasks // belonging to the level $k$
3: $k \leftarrow 1$; // first level
4: while ($k \leq L$) do
5:   for all tasks $t_i \in l_k$, compute $VM(t_i)$ using equation 8
     // assign task $t_i$ to the virtual machine $VM(t_i)$
     // that minimizes the execution cost
     $mincost[k, t_i] \leftarrow VM(t_i)$; // $mincost$ is a $L \times m$ matrix
     // where line $k$ corresponds to the assignment of all tasks
     // obtained starting by the tasks assignement belonging to
     // this level
6:   $h \leftarrow k + 1$; // compute $VM(t_i)$ for all tasks that belong // to levels $h > k$
7:   while ($h \leq L$) do
8:     for all tasks $t_i \in l_h$, compute $VM(t_i)$ using equation 6
       $mincost[k, t_i] \leftarrow VM(t_i)$;
9:     $h \leftarrow h + 1$
10:  end while
11:  $h \leftarrow k - 1$; // compute $VM(t_i)$ for all tasks that belong // to levels $h < k$
12:  while ($h \geq 1$) do
13:    for all tasks $t_i \in l_h$, compute $VM(t_i)$ using equation 7
       $mincost[k, t_i] \leftarrow VM(t_i)$;
14:    $h \leftarrow h - 1$
15:  end while
16:  $k \leftarrow k + 1$
17: endwhile
18: for each assignment, compute $FT(t_{exit})$ using equations 1, 2 and 3;
19: select the Pareto solutions among $L$ solutions;
```

while the second approach attempts to minimize the *time* function. The objective of the resource allocation phase of the time-based approach is to choose the virtual machine that minimizes the finish time of each task and the task is assigned to that virtual machine. The three allocation strategies, detailed above, are also applied to explore the given graph (DAG). The lines 3 (with $k = 1$) to 10 of Algorithm 2 corresponds to the top-dow strategy. The lines 11 (with $k = L$) to 15 of Algorithm 2 corresponds to the bottomp-up strategy. The lines 4-17 ($k \in \{2, ..., L-1\}$) of Algorithm 2 corresponds to the mixed strategy. Therefore, the obtained solutions number is equal to the number of graph levels (i.e. L). So, If the resource allocation phase starts at the level k, therefore the assignment of the tasks belonging to this level is only based on the execution time $(ET(t_i), VM_j)$, given by the following equation:

$\forall t_i \in l, VM(t_i) = VM_k$ such that:

$$ET(t_i, VM_k) = \min_{VM_j} ET(t_i, VM_j) \tag{9}$$

For the tasks belonging to the level l_{k+1} and l_{k-1}, the allocation decision of a task t_i are recursively defined by respectively the following equations until all the tasks are assigned:

$\forall t_i \in l_{k+1}, VM(t_i) = VM_k$ such that:

$$ET(t_i, VM_k) + \textstyle\sum_{t_h \in pred(t_i)} TT(VM(t_h), VM(t_i)) =$$

$$\min_{VM_j} \left\{ ET(t_i, VM_j) + \sum_{t_h \in pred(t_i)} TT(VM(t_h), VM(t_i)) \right\} \tag{10}$$

$\forall t_i \in l_{k-1}, VM(t_i) = VM_k$ such that:

$$ET(t_i, VM_k) + \textstyle\sum_{t_h \in succ(t_i)} TC(VM(t_i), VM(t_h)) =$$

$$\min_{VM_j} \left\{ ET(t_i, VM_j) + \sum_{t_h \in succ(t_i)} TC(VM(t_i), VM(t_h)) \right\} \tag{11}$$

Algorithm 2 Time-based approach

```
1: read the DAG, the RG and associated attributes values;
2: sort tasks at each level by traversing the DAG in a top-down fashion; // let L be
   the set of levels and $l_k$ the tasks // belonging to the level $k$
3: $k \leftarrow 1$; // first level
4: while ($k \leq L$) do
5:   for all tasks $t_i \in l_k$, compute $VM(t_i)$ using equation 9
     // assign task $t_k$ to the virtual machine $VM(t_i)$
     //that minimizes the execution time
     $mintime[k, t_i] \leftarrow VM(t_i)$; // $mincost$ is a $L \times m$ matrix
     // where line $k$ corresponds to the assignment of all tasks
     // obtained starting by the tasks assignement belonging to
     // this level
6:   $h \leftarrow k + 1$; // compute $VM(t_i)$ for all tasks that belong // to levels $h > k$
7:   while ($h \leq L$) do
8:     for all tasks $t_i \in l_h$, compute $VM(t_i)$ using equation 10
       $mintime[k, t_i] \leftarrow VM(t_i)$;
9:     $h \leftarrow h + 1$
10:  end while
11:  $h \leftarrow k - 1$; // compute $VM(t_i)$ for all tasks that belong // to levels $h < k$
12:  while ($h \geq 1$) do
13:    for all tasks $t_i \in l_h$, compute $VM(t_i)$ using equation 11
       $mintime[k, t_i] \leftarrow VM(t_i)$;
14:    $h \leftarrow h - 1$
15:  end while
16:  $k \leftarrow k + 1$
17: endwhile
18: for each assignment, compute $cost$ using equation 5;
19: select the Pareto solutions among $L$ solutions;
```

Pareto selection phase Recall that at the end of the previous phase L assignment of all tasks is obtained. In this phase, we first compute the overall completion time (using Equation 5) corresponding to each assignment and then only non-dominated solutions are maintained. The lines 18 and 19 of Algorithm 2 corresponds to the Pareto selection phase.

C. Cost-time-based approach

The cost-time-based approach objective is to offer a guidance for choosing between different offering in order to complete the workflow application considering the two criterion together. It is composed by two phases. The first one consists to execute Algorithm 1 and 2. The second one allows to select only the non-dominated solutions.

D. Lower bounds

In order to assess the quality of the obtained solutions by our algorithms with experiments, we need to know the value of an optimal solution of each criterion. But as the problem is NP-Hard, we are thus looking for good lower bounds.

For time criterion, a lower bound LB_t is obtained by the execution of the t_{exit} task in the critical path of the relaxed problem for the workflow $G = (T, E)$, in which each task t_j is executed on the virtual machine $VM^*(t_j)$ with the minimal execution time, i.e., $ET(t_j, VM^*(t_j)) = \min_{VM_k} ET(t_j, VM_k)$ and each edge (t_i, t_j) of G is valued with a lower bound of the transfer time between t_i and t_j. The transfer time $TT(i,j)$ is defined by the following formula:

$$TT(t_i,t_j) = \min \begin{cases} ET(t_j, VM^*(t_i)) - ET(t_j, VM^*(t_j)) & (1) \\ ET(t_i, VM^*(t_j)) - ET(t_i, VM^*(t_i)) & (2) \\ ET(t_i, t_j, VM_k) & (3) \end{cases}$$

where $ET(t_i,t_j,VM_k) = \min_{VM_l}\{ET(t_i,VM_l(t_j)) - ET(t_i,VM^*(t_i)) + ET(t_j,VM_l(t_j)) - ET(t_j,VM^*(t_j))\}$. In the formula of $TT(t_i,t_j)$, the first term (1) represents the additional execution time of task t_i if it is executed on virtual machine $VM^*(t_j)$ instead of $VM^*(t_i)$, the second term (2) represents the additional execution time of task t_j if it is executed on virtual machine $VM^*(t_i)$ instead of $VM^*(t_j)$, and the last term (3) represents the additional execution times of tasks t_i and t_j if they are executed on virtual machine $VM_k(t_i)$ instead of $VM^*(t_i)$, and on virtual machine $VM_k(t_j)$ instead of $VM^*(t_j)$, respectively.

Concerning the cost criterion, we use the same relaxation than for the computation of the LB_t, but on the execution cost instead the execution time. Then the lowed bound LB_c for the cost criterion is calculated as the total execution cost of the tasks of the relaxed valued graph, in which each the task t_j is executed on the virtual machine $VM^*(t_j)$ with minimal execution cost, i.e., $EC(t_j,VM^*(t_j)) = \min_{VM_k} EC(t_j,VM_k(t_j))$ and each edge (t_i,t_j) of G is valued with a lower bound of the transfer cost between t_i and t_j, then $TC(t_i,t_j)$ is given by the following formula:

$$TC(t_i,t_j) = \min \begin{cases} EC(t_j, VM^*(t_i)) - EC(t_j, VM^*(t_j)) & (1) \\ EC(t_i, VM^*(t_j)) - EC(t_i, VM^*(t_i)) & (2) \\ EC(t_i, t_j, VM_k) & (3) \end{cases}$$

where $EC(t_i,t_j,VM_k) = \min_{VM_l}\{EC(t_i,VM_l(t_i)) - EC(t_i,VM^*(t_i)) + EC(t_j,VM_k(t_j)) - EC(t_j,VM^*(t_j))\}$. In the formula of $TC(t_i,t_j)$, the term (1) represents the additional execution cost of task t_i if it is executed on virtual machine $VM^*(t_j)$ instead of $VM^*(t_i)$, the second term (2) represents the additional execution cost of task t_j if it is executed on virtual machine $VM^*(t_i)$ instead of $VM^*(t_j)$, and the last term (3) represents the additional execution costs of tasks t_i and t_j if they are executed on virtual machine $VM_k(t_i)$ instead of $VM^*(t_i)$, and on virtual machine $VM_k(t_j)$ instead of $VM^*(t_j)$, respectively.

V. Experimental results and discussion

We perform an empirical study of the heuristics discussed in Section IV. In the next subsection, we describe how we generate the data. Finally, we report the results and analysis in Section V-B.

A. Data setting

In our experiment, we simulate a Cloud computing environment in which five families of instances are randomly generated. A family is associated on tasks $n \in \{50, 100, 300, 600, 1000\}$. Each family of instances is subdivided into three series S, M and L with small, medium and large density of precedence constraints, respectively. For each pair (t_i,t_j) of tasks it is decided with a certain probability p that t_i precedes t_j, such that G is acyclic. For the series S, M, and L the corresponding probabilities are $p_S = 0.2$, $p_M = 0.4$ and $p_L = 0.6$. We consider the number of virtual machines types m as a function of the number of tasks, i.e., $m = n/10$. We assume that each task can be executed on all virtual machines types. For all families of instances, processing times of tasks on virtual machines are generated according to an uniform distribution, varying in $[1, 10]$. The unit execution cost on each virtual machine type is uniformly generated in interval $[0.1, 0.9]$. The bandwidth matrix between virtual machines B is uniformly generated, in the interval $[1, 9]$ and the transfer data matrix D is also uniformly generated in interval $[10, 90]$. Finally the transfer C_{out} and the receiving cost C_{in} of virtual machines are uniformly generated in the interval $[1, 9]$. Each series consists of 1000 instances.

B. Summary of results and discussion

The obtained Pareto solutions divided by the number of the overall obtained solutions is considered as the ratio of each approach. Figure's 3 histograms indicate two things. First, in most cases the ratio increases with the density of precedence constraints (i.e p), which is mainly due to the importance of time and cost communications. So, It is interesting to consider not all levels of the graph, but only those where the obtained solution is very likely a Pareto solution. Second, the ratio of Pareto solutions obtained by the cost-time based approach is almost equal to the sum of the ratios of the cost-based and the time-based approach divided by two, which means that no criterion dominates the other one. Therefore, the Pareto approach is the most appropriate one to deal with the allocation and scheduling workflow tasks in Cloud computing environment.

The result of the quality of the obtained solutions by our approaches is given in Figure 4, plotting the minimum D_{min}, the average $D_{average}$ and the maximum D_{max} performance criterion. The D_{min}, $D_{average}$ and D_{max} are obtained by dividing the minimum value, the average value and the maximum value among all obtained Pareto solutions and the lower bound, respectively, by considering the two criteria (i.e. cost and time). These results show that in average the considered performance criterion do not exceed 2.9, which means that the obtained Pareto solutions are less than to 2.9 times the lower bound. Note that this observation is the same in three proposed approaches.

VI. Conclusion

In this paper, we have proposed three bi-criteria complementary approaches to tackle the allocation and scheduling workflow problems in Cloud environments. Moreover, in order to assess the quality of obtained solutions by our approaches we have proposed two lower bounds for each considered criterion. Unlike existing works, these approaches take into account two conflicting criteria simultaneously: i) execution cost and ii) execution time. Moreover, they offer more flexibility to consumer to estimate their preferences and choose a desired schedule from the obtained efficient solutions. More precisely, the first one focused on the cost incurred by using a set of resources, while the second approach attempts to minimize the

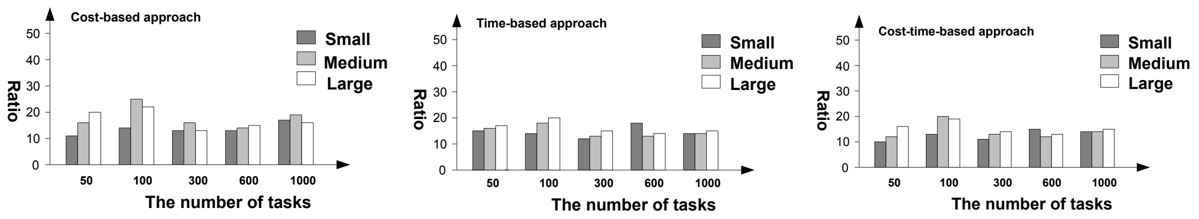


Fig. 3. Scheduling 1000 randomly generated workflows versus the ratio of the obtained Pareto solutions for each proposed approach

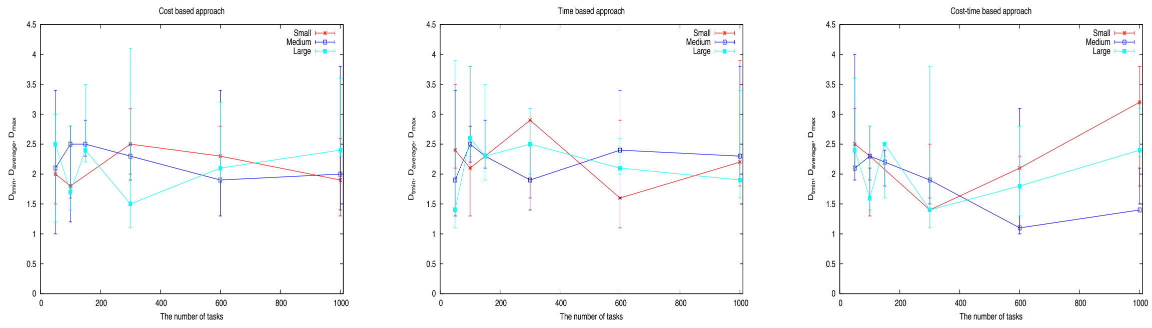


Fig. 4. Scheduling 1000 randomly generated workflows versus the ratio D_{min}, $D_{average}$ and D_{max} of the obtained Pareto solutions

overall execution time. The third approach is based on the two first approaches for selecting only the Pareto solutions.

Results presented in this paper are very encouraging but they in the same time open up new and interesting questions. Therefore, we focus on several perspectives for this work. Currently, we are extending the proposed approaches for business process workflows considering workflow patterns and compositions. Moreover, we plan to extend the proposed work to take into account others criteria like carbon emission and energy cost. In addition, it is interesting to adapt the proposed approaches to the case where a task cannot be executed on all virtual machine types.

REFERENCES

[1] P. Mell, T. Grance, *The NIST Definition of Cloud Computing*, (Draft)-Recommendations of the National Institute of Standards and Technology. Special publication 800-145 (draft), Gaithersburg (MD).

[2] R. Buyya, S. Pandey and C. Vecchiola, *Cloudbus toolkit for market-oriented cloud computing*, In CloudCom'09: Proceedings of the 1st International Conference on Cloud Computing, volume 5931 of LNCS, pp. 24-44, Springer, Germany, December 2009.

[3] G. Juve and E. Deelman, *Scientific Workflows in the Cloud, in Grids, Clouds and Virtualization*, M. Cafaro Eds. Springer, pp. 71-91, 2010.

[4] G. Juve, E. Deelman, K. Vahi,, G. Mehta, B. Berriman, B. P. Ber-man and P. Maechling, *Scientific workflow applications on Amazon EC2*. In Cloud Computing Workshop in Conjunction with e-Science, Oxford, UK, 2009, IEEE.

[5] A. V. Mladen, *Cloud Computing - Issues, Research and Implementations*, in Journal of Computing and Information Technology, Volume 16, Issue 4, 2008, pp. 235-246.

[6] 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, Volume 25, Issue 5, pp. 528-540, 2009.

[7] H. Topcuoglu, S. Hariri, M. Y . Wu, *Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing*, IEEE Transactions on Parallel and Distributed Systems, Volume 13, Issue 3, 2002.

[8] J. D. Ullman, *NP-Complete Scheduling Problems*, J. Computer and Systems Sciences, Volume 10, pp.3 84-393, 1975.

[9] M. R. Gary and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.

[10] H. Hoogeveen,*Multicriteria scheduling*, In European Journal of Operational Research 167, pp. 592-623, 2005.

[11] M. Wieczorek, S. Podlipnig, R. Prodan, T. Fahringer, *Bi-criteria Scheduling of Scientific Workfows for the Grid*, Eighth IEEE International Symposium on Cluster Computing and the Grid, pp 9-16, 2008.

[12] V. T'kindt, J. C. Billaut, *Multicriteria Scheduling. theory, models and algorithms*, Springer, 2nd Edition, 2006.

[13] H. Topcuoglu, S. Hariri and M. Y. Wu, *Performance effective and low-complexity task scheduling for heterogeneous computing*, IEEE Trans. on Parallel and Distributed Systems, Volume 13, Issue 3, pp. 260-274, 2002.

[14] E. Ilavarasan and P. Thambidurai, *Low complexity performance effective task scheduling algorithm for heterogeneous computing environments*, Journal of Computer Sciences, Volume 3, Issue 2, pp. 94-103, 2007.

[15] J. Tordsson, R. Montero, R. Moreno-Vozmediano,and I. Llorente,*Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers*, Future Generation Computer Systems, Volume 28, Issue 2, pp. 358-367, 2012.

[16] D. Breitgand, A. Marashini and J. Tordsson, *Policy-driven service placement optimization in federated clouds.*, Technical report, IBM Haifa Labs, 2011.

[17] R. V. den Bossche, K. Vanmechelen and J. Broeckhove, *Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads*, In Proceedings of the 3rd IEEE international conference on cloud computing, pp. 228-235, 2010.

[18] R. Van den Bossche, K. Vanmechelen and J. Broeckhove, *Cost-Efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds*, on Cloud Computing Technology and Science, IEEE International Conference, pp. 320-327, 2011.

[19] R. Buyya, C. S. Yeo, and S. Venugopal, *Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities*, Proc. Of the 10th IEEE International Conference on High Performance Computing and Communications, 2008, pp. 5-13.