Workflow Scheduling for SaaS / PaaS Cloud Providers Considering Two SLA Levels

Thiago A. L. Genez, Luiz F. Bittencourt, Edmundo R. M. Madeira
Institute of Computing – University of Campinas – UNICAMP
Av. Albert Einstein, 1252, 13083-852 – Campinas – São Paulo – Brazil
e-mail: thiagogenez@lrc.ic.unicamp.br, {bit,edmundo}@ic.unicamp.br

***Abstract*—Cloud computing is being used to avoid maintenance costs and upfront investment, while providing elasticity to the available computational power in a pay-per-use basis. Customers can make use of the cloud as a software (SaaS), platform (PaaS), or infrastructure (IaaS) provider. When one customer utilizes an environment provided by a SaaS cloud, she is unaware of any details about the computational infrastructure where her requests are being processed. Therefore, such infrastructure can be composed of computational resources from a datacenter owned by the SaaS or its resources can be leased from a cloud infrastructure provider. In this paper we present an integer linear program (ILP) formulation for the problem of scheduling SaaS customer's workflows into multiple IaaS providers where SLA exists at two levels. In addition, we present heuristics to solve the relaxed version of the presented ILP. Simulation results show that the proposed ILP is able to find low-cost solutions for short deadlines, while the proposed heuristics are effective when deadlines are larger.**

I. Introduction

A SaaS cloud provides software as services to the users through the Internet [1], [2], [3]. The number of clients requesting services from the SaaS can be seasonal, thus with variations over the time. To be able to provide a good quality of service (QoS) within the customer's service level agreements (SLA), the SaaS must be prepared to attend a peak of demand. Thereupon, the computational capacity of the SaaS provider should be sufficient to deal with the peak number of customers, which brings high maintenance costs and underutilization on off-peak demand times. To contour this situation, the SaaS may rely on resources leased from IaaS cloud providers also through SLA, bringing *elasticity* to its computational power. With that, more resources can be leased from the IaaS when they are needed to comply with customers' QoS.

In this paper we consider a SaaS which provides a workflow execution service to its customers. The customer submits his/her workflow to be executed by the SaaS along with a response time (deadline) to be obeyed. The SaaS provider aims to execute the customer's workflow within the deadline using computational resources leased from multiple IaaS providers. To achieve this, we formulate a scheduler as an integer linear program with the objective of minimizing the monetary execution costs while meeting the deadline stipulated by the user in the SLA contract. The scheduling algorithm runs in the SaaS provider and determines which IaaS provider and which type of virtual machine (VM) have to be leased to execute the workflow in order to guarantee QoS to SaaS's costumers. It is worth noting that this scenario and, thus, the presented algorithms are also applicable to platform as a service (PaaS) providers. We use only SaaS in the rest of the text for the sake of clarity.

The main contributions of this paper are: (i) the integer linear program (ILP) for the proposed scheduling problem at the two-level SLA cloud; (ii) two heuristics to obtain integer solutions from the relaxed version of the proposed ILP; and (iii) simulations that evaluate scheduling results from the ILP and the proposed heuristics.

The remainder of the paper is organized as follows. Section II shows related works. In Section III we present the scheduling problem and conceptual background, while Section IV shows the proposed scenario. The scheduling algorithm is introduced in Section V. Simulation results are discussed in Section VI, while the final remarks and conclusions are presented in Section VII.

II. Related Works

Wu et al. describe in [4] a resource allocation algorithm for SaaS providers that want to minimize infrastructure cost and SLA violations. They present cost-effective policies for mapping and scheduling in order to achieve profit maximization for the SaaS provider through the use of multiple IaaS providers. Li and Guo describe in [5] a stochastic ILP for optimal resource scheduling in cloud computing. They show how to select resources from public clouds to perform abstract services in business process instances while satisfying costs defined in SLAs. Reig et al. propose in [6] a strategy to minimize the computational resource consumption through a prediction system which determines the minimum cost resource for a job to be executed before its deadline to prevent SLA violations. Although these works present advances in scheduling tasks on clouds, none of them consider workflows.

Bittencourt and Madeira propose in [7] a strategy to schedule service workflows in a hybrid cloud. They show an algorithm called Hybrid Cloud Optimized Cost (HCOC) that decides which resources should be leased from the public cloud to increase the processing power of the private cloud. Although HCOC minimizes the monetary cost of the workflow execution within a deadline, Bittencourt and Madeira do not consider the notion of SLA, which is an important aspect in the business model proposed by the cloud paradigm. Pandey et al.

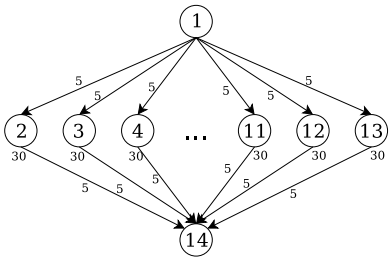


Fig. 1. Example of fork-join DAG with 14 nodes (from [7]).

describe in [8] a particle swarm optimization (PSO) heuristic to schedule application workflows. They show a model for task-resource mapping to minimize the overall execution cost in cloud computing environments. Nevertheless, they do not consider the notion of SLA on both user and provider sides.

III. Background and Concepts

In this section we briefly introduce the workflow representation, cloud computing and SLAs in clouds.

A. Workflow Representation

A workflow is generally represented by a Directed Acyclic Graph (DAG) $\mathcal{G} = \{\mathcal{U}, \mathcal{E}\}$, where each node $u_i \in \mathcal{U}$ represents a service and each edge $e_{i,j} \in \mathcal{E}$ represents a data dependency between services i and j, that is, $e_{i,j}$ is the data produced by u_i and consumed by u_j. Labels on nodes represent computation costs (number of instructions, for instance), while labels on edges represents communication costs (bytes to transmit, for instance), as shown in the example of Figure 1. We assume, without loss of generality, that all DAGs have only one entry node (first node) and one exit node (last node). In addition, we assume that each node $u \in \mathcal{U}$ is indivisible and will execute in only one virtualized processing core. Therefore, the main objective of the workflow scheduler is to decide where each node component of the DAG will execute.

B. Cloud Computing and SLA

Cloud customers usually need the services offered by service providers with certain Quality-of-Service (QoS) guarantees specified in Service Level Agreements (SLA) [9]. SLA is a legal document where the descriptions of such services are formally defined, delivered, and charged [10]. The SaaS provider can lease VM instances provided by IaaS providers through two types of SLA contracts: on-demand or reserved. In the first SLA type, the SaaS provider will lease VMs with no long-term commitments and will pay only for the computing power per time unit (usually per hour). On the other hand, in the reserved VMs SLA, the SaaS provider will lease VMs with a long-term commitment (1 to 3 years, for instance) and will pay an advance fee to reserve each VM. In turn, the SaaS provider receives a significant discount on the time unit usage charge for each reserved instance. The idea of SLA contracts used in this paper will be similar to the Amazon EC2 contract [1].

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

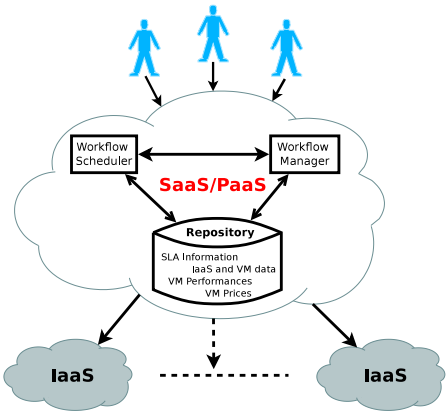


Fig. 2. The proposed scenario

IV. The Cloud Scenario

The proposed scenario for serving SaaS cloud's customers is shown in Figure 2. Its main components are a workflow manager, a workflow scheduler, and a repository used by these components. In practice, there is a limited number of VMs that the IaaS provider can offer per cloud user [4]. The importance of this scenario is that, to maximize its profit, the SaaS provider must not refuse customer requests when its reserved computational resources (and/or its own infrastructure) are entirely busy on peaks of demand.

In this scenario the SaaS's customer submits his/her workflow to be executed and he/she wants to have it finished before a certain time (deadline). The workflow manager analyzes the workflow, consulting the repository (database), which contains information such as VM performance and VM prices. The repository may also have historical information about previous executions of services requested by the SaaS customers and resources performance for each one of them. The workflow manager sends such information to the workflow scheduler which uses it to perform the scheduling using the proposed scheduling algorithms. If the current infrastructure (SaaS private resources and/or reserved resources from IaaS providers) is not enough to obey the workflow deadline, the scheduling algorithm will also return schedules to IaaS resources which should be leased on-demand.

V. The Scheduling Algorithm

The proposed scheduling algorithm deals with the trade-off between leasing cheap or powerful (and expensive) VMs. The objective is to find solutions that minimize the global SaaS infrastructure monetary costs and SLA violations.

A. Notation and Problem Modeling

The following notation is used for the integer linear programming formulation of the problem:

- n: number of nodes in the DAG ($n \in \mathbb{N}$);
- $\mathcal{W} = \{w_1, \ldots, w_n\}$: set of processing demands for each node $u \in \mathcal{U}$, expressed as the number of instructions to be processed ($w_k \in \mathbb{R}^+$);

- $f_{i,j}$: number of data units transmitted between $u_i \in \mathcal{U}$ and $u_j \in \mathcal{U}$ ($f_{i,j} \in \mathbb{R}^+$)
- $\mathcal{H}(j) = \{ij : i < j$, there exists an arc from vertex i to vertex j in the DAG$\}$, is the set of immediate predecessors of $u_j \in \mathcal{U}$;
- $\mathcal{D}_\mathcal{G}$: finish time (deadline) desired by the SaaS customer for the DAG $\mathcal{G}$;
- $\mathcal{I} = \{i_1, \ldots, i_m\}$: the set of IaaS providers composing the infrastructure;
- δ_i: maximum number of VMs that a cloud customer can have leased from the IaaS $i \in \mathcal{I}$ at any given time t ($\delta_i \in \mathbb{N}$).

Our scenario is composed of two SLA layers. The first one contains all SLAs where deadline $\mathcal{D}$ is defined. The second one is composed of all SLAs that defines the characteristics of each VM leased, such as number of processing cores, amount of memory, and prices. Let $\mathcal{S}_i$ be the set of all SLAs that have been signed between the SaaS provider and each IaaS provider $i \in \mathcal{I}$. We define the set $\mathcal{S}_i = \sigma_i^r \cup \sigma_i^o$ which consists of two subsets, where the subset σ_i^r includes SLAs for reserved VMs readily available, and the subset σ_i^o includes only SLAs for on-demand VMs that can be leased on-the-fly.

Let $\mathcal{V}_i^r$ be the set that includes only VMs associated with prices for reserved resources from IaaS $i \in \mathcal{I}$ and $\mathcal{V}_i^o$ be the set that includes only VMs associated with prices for on-demand resources from IaaS $i \in \mathcal{I}$. Each SLA $s_r \in \sigma_i^r$ is related to one VM $v \in \mathcal{V}_i^r$, and each SLA $s_o \in \sigma_i^o$ is related to one VM $v \in \mathcal{V}_i^o$. One of the parameters defined in each contract agreement $s \in \mathcal{S}_i$ is the number of reserved VMs $\alpha_s \in \mathbb{N}^+$ available for the SaaS provider. Another SLA parameter indicates the duration time $t_s \in \mathbb{N}^+$ of the term commitment. Furthermore, the provisioning of virtual machines is limited by the each IaaS provider for each client, then the following restriction must be strictly obeyed at any given time t: $\sum_{s \in \mathcal{S}_i} \alpha_s \leq \delta_i, \quad \forall\, i \in \mathcal{I}$.

The resources available for leasing come from a set of infrastructure providers (IaaS) $\mathcal{I} = \{i_1, \ldots, i_m\}$ and a set of VMs $\mathcal{V} = \left(\bigcup_{i=1}^{m} \mathcal{V}_i \right)$, where each $\mathcal{V}_i = \mathcal{V}_i^r \cup \mathcal{V}_i^o$ consists of the union of two subsets having reserved and on-demand resources from the IaaS i, according to the SLAs in $\mathcal{S}_i$. Moreover, we define the following IaaS characteristics for the ILP formulation:

- m: number of IaaS providers ($m \in \mathbb{N}$);
- $\mathcal{P}_v$: number of processing cores of $v \in \mathcal{V}$, where $\mathcal{P}_v \in \mathbb{N}$;
- $\mathcal{J}_v$: time units that VM $v \in \mathcal{V}$ takes to execute 1 instruction, with $\mathcal{J}_v \in \mathbb{R}^+$;
- $\mathcal{L}_{v_g,v_h}$: time units taken to transmit 1 data unit over the network link between $v_g \in \mathcal{V}$ and $v_h \in \mathcal{V}$, with $\mathcal{L}_{v_g,v_g} = 0$ and $\mathcal{L}_{v_g,v_h} \in \mathbb{R}^+$;
- $\mathcal{B}_{i,v}$: binary variable that assumes a value of 1 if $v \in \mathcal{V}$ belongs to the IaaS $i \in \mathcal{I}$, and 0 otherwise;
- $\mathcal{C}_v$: price to lease $v \in \mathcal{V}$ during one time unit, with $\mathcal{C}_v \in \mathbb{R}^+$.

Besides that, on-demand pricing of any $v_o \in \mathcal{V}_i^o$ is generally greater than reserved pricing for the same kind of VM $v_r \in \mathcal{V}_i^r$ in the same $i \in \mathcal{I}$, which is also a usual configuration in real environments. In other words, $\mathcal{C}_{v_o} \geqslant \mathcal{C}_{v_r}, \quad \forall\, v_o \in \mathcal{V}_i^o, \quad \forall\, v_r \in \mathcal{V}_i^r$. Let $\zeta = \{\sigma_1^r, \ldots, \sigma_m^r\}$ be the set that contains all SLAs associated with reserved VMs. Thus, we define the variable $\mathcal{K}_{s,v}$ that assumes a value of 1 if the SLA $s \in \zeta$ is related to the reserved VM $v \in \mathcal{V}$, or 0 otherwise. That means, if $\mathcal{K}_{s,v} = 1$ and $\mathcal{B}_{i,v} = 1$, then $s \in \sigma_i^r$ and $v \in \mathcal{V}_i^r$.

The scheduling cost-optimization problem can be stated as follows: "*Find a feasible mapping M between the nodes of the DAG $\mathcal{G}$ and the VMs from multiple IaaS providers, such that the sum of the monetary computation cost for all nodes $u \in \mathcal{U}$ on a VM $v \in \mathcal{V}$ is minimal, the dependencies among nodes are not violated, and the total execution time of the mapping M (makespan) $\mathcal{M}_\mathcal{G}$ is at most equal to the deadline required by user, i.e., $\mathcal{M}_\mathcal{G} \leq \mathcal{D}_\mathcal{G}$*".

B. Formulation of the Integer Linear Program

The integer linear program solves the scheduling problem through the binary variables x and y and the constant $\mathcal{C}_v$:

- $x_{u,t,v}$: binary variable that assumes the value 1 if the node u finishes at time t in the VM v; otherwise this variable assumes the value 0;
- $y_{t,v}$: binary variable that assumes the value 1 if the VM v is being used at time t; otherwise this variable assumes the value 0;
- $\mathcal{C}_v$: constant that assumes the cost per time unit for using the VM v.

The formulation of the integer linear programming problem considers discrete time intervals. Let $\mathcal{T} = \{1, \ldots, \mathcal{D}_\mathcal{G}\}$ be the timeline of the possible workflow execution time, which ranges from 1 to the desired deadline $\mathcal{D}_\mathcal{G}$. The integer linear program is formulated as follows.

$$\text{Minimize} \quad \sum_{t \in \mathcal{T}} \sum_{v \in \mathcal{V}} y_{t,v} \times \mathcal{C}_v, \quad \text{Subject to:}$$

$$\text{(C1)} \quad \sum_{t \in \mathcal{T}} \sum_{v \in \mathcal{V}} x_{u,t,v} = 1; \quad \forall\, u \in \mathcal{U};$$

$$\text{(C2)} \quad \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{V}} \sum_{t=1}^{\lceil w_u \times \mathcal{J}_v \rceil} x_{u,t,v} = 0;$$

$$\text{(C3)} \quad \sum_{s=1}^{t - \lceil w_z \times \mathcal{J}_r + f_{u,z} \times \mathcal{L}_{i,j} \rceil} x_{u,s,v} \geq \sum_{s=1}^{t} x_{z,s,r} \quad \forall\, z \in \mathcal{U}, \quad \forall\, u \in \mathcal{H}(z), \quad \forall\, r, v \in \mathcal{V}, \quad \forall\, t \in \mathcal{T}, \quad \forall\, i, j \in \mathcal{I} \mid \mathcal{B}_{i,v} = 1, \quad \mathcal{B}_{j,r} = 1;$$

$$\text{(C4)} \quad \sum_{u \in \mathcal{U}} \sum_{s=t:\, t \leq \mathcal{D}_\mathcal{G} - \lceil w_u \times \mathcal{J}_v \rceil}^{t + \lceil w_u \times \mathcal{J}_v \rceil - 1} x_{u,s,v} \leq \mathcal{P}_v \quad \forall\, v \in \mathcal{V}, \quad \forall\, t \in \mathcal{T};$$

$$\text{(C5)} \quad \sum_{s=t - \lceil w_u \times \mathcal{J}_v \rceil + 1}^{t} y_{s,v} \geq x_{u,t,v} \times \left(\lceil w_u \times \mathcal{J}_v \rceil \right) \quad \forall\, u \in \mathcal{U}, \quad \forall\, v \in \mathcal{V}, \quad \forall\, t \in \{\lceil w_u \times \mathcal{J}_v \rceil, \ldots, \mathcal{D}_\mathcal{G}\};$$

$$\text{(C6)} \quad \sum_{v \in \mathcal{V}} y_{t,v} \leq \delta_i \quad \forall\, i \in \mathcal{I}, \quad \forall\, t \in \mathcal{T} \mid \mathcal{B}_{i,v} = 1;$$

$$\text{(C7)} \quad \sum_{v \in \mathcal{V}} y_{t,v} \leq \alpha_s \quad \forall\, s \in \zeta, \quad \forall\, t \in \mathcal{T} \mid \mathcal{K}_{s,v} = 1;$$

$$\text{(C8)} \quad x_{u,t,v} \in \{0,1\} \quad \forall\, u \in \mathcal{U}, \quad \forall\, t \in \mathcal{T}, \quad \forall\, v \in \mathcal{V};$$

$$\text{(C9)} \quad y_{t,v} \in \{0,1\} \quad \forall\, t \in \mathcal{T}, \quad \forall\, v \in \mathcal{V};$$

The constraint (C1) specifies that a service must be executed only once and in a single VM. The constraint (C2) establishes that a DAG node u cannot be set as finished until it has been executed in a VM v. The constraint (C3) determines that a DAG node z cannot begin its execution until all preceding nodes have finished their processing and the resulting data has arrived at the VM that will run z. The constraint (C4) specifies that the number of DAG nodes executing on a VM v at a given time t cannot exceed the number of processing cores of v. The constraint (C5) determines that a VM must stay active (i.e., with status *being used* enabled in the variable y) while it is executing the node which requires it. The constraint (C6) specifies that the number of reserved VMs plus the number of on-demand VMs cannot exceed the maximum number allowed by each IaaS provider. The constraint (C7) establishes that the amount of virtual machines being used cannot exceed the limit stipulated in the SLA. The last two constraints, (C8) and (C9), specify that the variables of this integer linear program, namely x and y, will only assume the binary values 0 or 1.

Given the NP-Completeness of the workflow scheduling problem, the time taken by the ILP solver to find the optimal integer solution with the proposed ILP increases exponentially with the input size. We considered three different manners to cope with this: (i) *optimal approach*: find an optimal solution within a time limit that, if reached, returns the best solution so far; (ii) *first solution approach*: returns the first integer solution found within a time limit, if reached, returns the best solution so far; and (iii) *relaxed approach*: solve the relaxed ILP and apply heuristics to the fractional solution in order to find the integer solution. Approaches (i) and (ii) are directly obtained by changing the solver parameters. By doing this, the solver stops when a certain execution time is reached or when the optimal solution is found (for approach (i)) or when the first integer solution is found (for approach (ii)). In both cases, if the time limit is reached, the solver may return the best solution so far or declare the problem instance unfeasible. Approach (iii), besides telling the solver to relax the solution, needs the implementation of a heuristic over its output, since in this case the variables x and y can assume values in $\mathbb{R}^+$. Next, we describe two heuristics proposed to find integer solutions based on relaxed ones given by the ILP solver.

C. ILP Relaxation

The relaxation of the integer linear program consists in changing the integer set $\{0,1\}$ of the constraints in (C8) and (C9) to its real $[0,1]$ interval. Thereupon, the relaxed integer linear program returns real numbers in the variable $x_{u,t,v}$ representing the best way to split a node among virtual machines, therefore potentially returning more than one VM option to execute each node. However, each node in the DAG is indivisible. In this section we describe two heuristics that use an iterative method to obtain an integer solution from a relaxed one in the proposed ILP. Both heuristics are based on the variable $x_{u,t,v}$, which provides the node id u, node finish time t, and VM v to execute it.

The iterative method works as follows. Given a DAG $\mathcal{G}$, a deadline $\mathcal{D}_\mathcal{G}$, and the set $\mathcal{V}$ of IaaSs, the first step is to call the ILP solver to obtain a relaxed solution for the input instance. With this, we obtain real values for the variable $x_{u,t,v}$. Based on these values, we select an output $x_{u,t,v}$ and make it integer by adding a new constraint $x_{u,t,v} = 1$. By setting a $x_{u,t,v}$ to 1 we are actually deciding that node u is scheduled to run on VM v finishing at time t. These steps are repeated $k \leq |\mathcal{U}| = n$ times, until all resulting variables present an integer solution. The two proposed heuristics differ on how the $x_{u,t,v}$ is selected from the relaxed solution to be set to 1 at each iteration. The general iterative algorithm is shown in Figure 3.

Require: DAG $\mathcal{G} = \{\mathcal{U}, \mathcal{E}\}$, deadline $\mathcal{D}_\mathcal{G}$, set of IaaSs $\mathcal{V}$
Ensure: Schedule of $\mathcal{G}$ in $\mathcal{V}$

1: Call relaxed ILP solver
2: **while** $\exists\ u \in \mathcal{U}$ such that $\sum_{t \in \mathcal{T}} \sum_{v \in \mathcal{V}} x_{u,t,v} \neq 1$ **do**
3: Choose a node $u_i \in \mathcal{U}$ according to the heuristic
4: Choose a resulting variable x_{u_i,t_a,v_x} according to the heuristic
5: Add new constraint $x_{u_i,t_a,v_x} = 1$ to the ILP
6: Call relaxed solver for the new ILP
7: **end while**
8: Return the solution from the last solver call

Fig. 3. The iterative method for obtaining an integer solution.

The first heuristic initially tosses a coin to decide if it should start from the first or from the last node of the DAG. If the first node (u_0) is selected, then the heuristic sets the variable $x_{u_0,t_a,v_x} = 1$ such that t_a is *minimum*, thus stipulating that node u_0 will execute in VM v_x, finishing at the minimum time t_a returned in the relaxed solution. If the last node u_n is selected, the heuristic sets the variable $x_{u_n,t_b,v_z} = 1$ such that t_b is *maximum*, thus stipulating that node u_n will execute in VM v_z, finishing at the maximum time t_b returned in the relaxed solution. After this first round, the heuristic keeps taking not scheduled tasks from the beginning and from the end of the DAG in turns, until an integer solution is reached or an unfeasible instance is found. First level tie-breaks are made by taking the larger x, and randomly at second level. We call this heuristic *begin-minimum end-maximum times* (BMEMT).

The second heuristic randomly selects at each iteration a ready node u_r with all its predecessors already scheduled and sets the variable $x_{u_r,t,v} = 1$ such that t is *minimum*. Ties are broken firstly by selecting the VM v which has minimum cost, and secondly by taking the larger x. If the tie persists, it is

randomly broken. We call this heuristic *begin-minimum time* (BMT).

VI. Evaluation

We implemented the presented integer linear program in JAVA and conducted simulations using the IBM ILOG CPLEX Optimizer[2] with default configuration. We evaluated the performance of the presented linear program when finding the *optimal solution*, the *first solution*, as well as BMEMT and BMT heuristics over the relaxed solution, as explained in Section V-B and Section V-C. The evaluated metrics are the monetary cost of the schedule, the workflow makespan, the solve time of the algorithm, and the number of times no solution was found (number of unfeasible solutions).

A. Simulation Configurations

We used 3 IaaS providers in our simulations, each one with its own prices for reserved and on-demand VMs. Table I shows VM options for IaaS A, Table II for IaaS B, and III for IaaS C. Moreover, Table IV shows the SaaS SLA contracts for reserved resources. The maximum number of VMs that can be leased from each IaaS was set $\delta_A = 4$, $\delta_B = 7$, $\delta_C = 2$.

Public clouds usually do not provide information about the quality of service for communication among internal nodes or external links. We assume that the bandwidth of links between VMs within the same IaaS is larger than the external links (between IaaS providers), which is a reasonable assumption in real environments. This is reflected in our simulation by randomly generating a $\mathcal{L}$ in the $[2, 3]$ interval for links between two different clouds, while for links between VMs inside the same cloud $\mathcal{L}$ is taken from the $[0.1, 0.2]$ interval. Our evaluation comprises simulations with DAGs of real world applications such as Montage [11] and fork-join DAG with 20 nodes of Figure 1. Simulations were run in an Intel® Core™ 2 Quad CPU Q6700 2.66GHz and $8GB$ of RAM.

B. Results

We have run 20 simulations for each DAG with the IaaS configurations shown in Tables I to IV. In each simulation, the computation cost for each node and the communication cost for each dependency were taken from the $[1, 3]$ interval. Each simulation was carried out in the following sequence:

1) Run BMT and BMEMT with time limit of 1800 seconds on each solver iteration and store the running times RT_{BMT} and RT_{BMEMT}.
2) Run *optimal solution* and *first solution* with time limit equal to RT_{BMT} (curves OPT-vs-BMT and FS-vs-BMT in the graphs).
3) Run *optimal solution* and *first solution* with time limit equal to RT_{BMEMT} (curves OPT-vs-BMEMT and FS-vs-BMEMT in the graphs).

To represent possible deadlines requested by the user, we have run simulations with $\mathcal{D}_\mathcal{G}$ varying from $T_{max} \times 2/7$ to $T_{max} \times 6/7$ in $1/7$ steps, where T_{max} is the makespan of the cheapest sequential execution of all nodes in the DAG on a single resource. Deadlines of $T_{max} \times 1/7$ showed only unfeasible solutions for all algorithms, while scheduling for deadlines of $T_{max} \times 7/7$ can be achieved by putting all tasks in the cheapest resource. It is important to note that the divisor 7 was chosen only to assess the evolution of the approaches (*optimal solution*, *first solution*, BMEMT, and BMT) with increasing deadline, so other dividers could be used.

[2] http://www.ibm.com/software/integration/optimization/cplex-optimizer/

TABLE V
Unfeasible solutions (%).

DAG	$\frac{\mathcal{D}_\mathcal{G}}{T_{max}}$	FS vs BMT	FS vs BMEMT	OPT vs BMT	OPT vs BMEMT	BMT	BMEMT
FJ-20	2/7	5	5	0	0	44	38
	3/7	0	57	4	57	0	9
	4/7	28	85	23	85	0	0
	5/7	19	71	19	71	4	0
	6/7	44	83	44	83	11	5
Mon.	2/7	100	100	94	94	100	100
	3/7	42	42	4	4	47	38
	4/7	4	4	0	0	9	19
	5/7	23	23	4	14	14	28
	6/7	50	50	33	33	44	50

Figure 4 shows results for simulations with 20-node fork-join DAGs. BMT and BMEMT performs better with larger deadlines, being able to find solutions with average costs from 40% ($\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 3/7$) to 53% ($\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$) lower with the same average solve times. Moreover, the FS-vs-BMEMT and OPT-vs-BMEMT approaches were not able to find feasible solutions in nearly all cases (Table V), which reflects in a lower average makespan for the feasible solutions when $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$.

Similar results were found when we change the DAG topology. Results for Montage DAGs are shown in Figure 5, with BMT and BMEMT being from 50% to 72% better than FS when regarding to costs, although with higher solve times, and getting up to 50% better than OPT with higher deadlines with the same solve times. However, we note that all algorithms tended to increase the number of unfeasible solutions with higher deadlines due to time limits.

Results presented in this section suggest that the BMT and BMEMT heuristics can be effective for reducing costs when having the same running times as the optimal approach mainly with higher deadlines. This can be explained by the discretization of time needed for the integer linear formulation: higher deadlines increase the number of discrete intervals in $\mathcal{T} = \{1, \ldots, \mathcal{D}_\mathcal{G}\}$, also increasing time complexity thus leading the solver to have difficulties in finding a good feasible integer solution. Moreover, in a general view, higher costs do not imply in proportionally lower makespans: wrong VM choices can make the schedule costly without makespan improvements, as can be concluded from Figures 4(a) and 4(b) for instance. In addition, the presented algorithms can provide basis for the SaaS provider to negotiate its SLA contracts with its customers.

TABLE I
IAAS PROVIDER A

Type	Core	Performance Per Core	On-demand Prices	Reserved Prices
Small	1	1.5	$0.13	$0.045
Medium	2	1.5	$0.20	$0.070

TABLE II
IAAS PROVIDER B

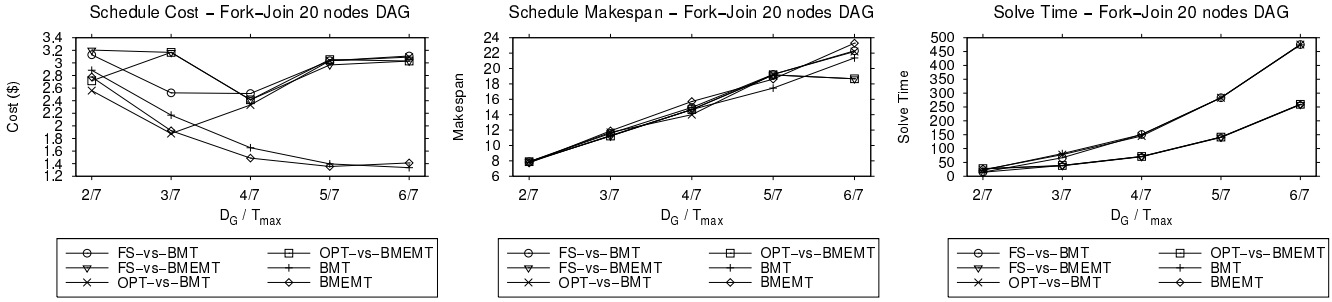
Type	Core	Performance Per Core	On-demand Prices	Reserved Prices
Small	1	2	$0.17	$0.045
Medium	2	2	$0.30	$0.059
Large	3	2	$0.40	$0.140
Extra-Large	4	2	$0.52	$0.183
Double Extra-Large	8	2	$0.90	$0.316

TABLE III
IAAS PROVIDER C

Type	Core	Performance Per Core	On-demand Prices	Reserved Prices
Small	1	2	$0.15	$0.052
Medium	2	2	$0.25	$0.088
Large	4	2.5	$0.50	$0.176
Extra-large	8	2.5	$0.80	$0.281

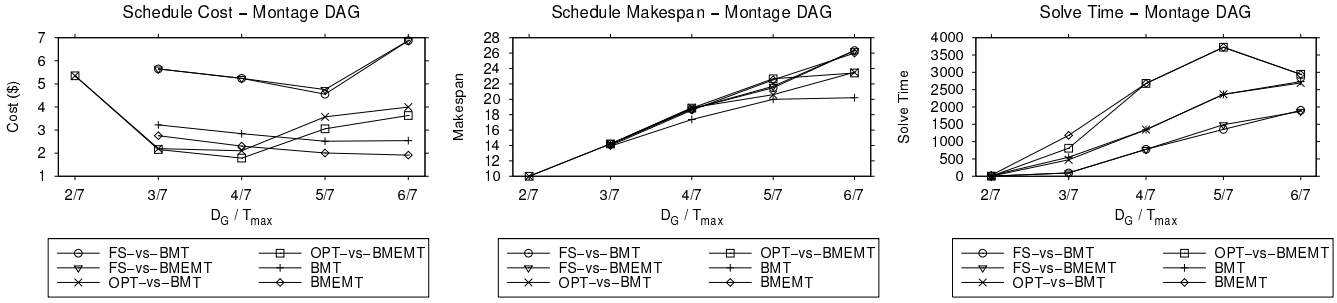
TABLE IV
SAAS SLAS FOR RESERVED VMS

Type	IaaS	VM	Number
Reserved	A	Small	1
Reserved	A	Medium	1
Reserved	B	Small	1
Reserved	B	Medium	1



(a) Cost (b) Makespan (c) Solve Time

Fig. 4. Results for the Fork-Join DAG with 20 nodes.



(a) Cost (b) Makespan (c) Solve Time

Fig. 5. Results for the Montage DAG.

VII. CONCLUSION

While the customer wants its jobs to be executed within an expected response time (deadline), the SaaS or PaaS clouds wants to maximize its profit. In this paper we presented an integer linear program (ILP) to solve the workflow scheduling problem in SaaS or PaaS clouds with two levels of SLA: the first one with its customers and the second one with its IaaS providers. In addition, we present two heuristics, namely BMT and BMEMT, to find feasible integer solutions over the relaxed runs of the proposed ILP. Simulation results shown that the optimal run of the proposed ILP is able to find low-cost solutions with shorter deadlines, while the proposed relaxed heuristics are effective to find low-cost solutions for larger deadlines. In-development works include non-iterative heuristics. In addition, extending the proposed ILP to consider (i) multiple workflow scheduling in the same set of resources, and (ii) fault tolerance mechanisms are considered for future work.

ACKNOWLEDGMENT

The authors would like to thank CAPES CNPQ, and FAPESP (2010/14433-8 and 2009/15008-1) for the financial support, CENAPAD-SP and Intel Manycore Testing Lab for the computational resources, and IBM for providing the CPLEX tools.

REFERENCES

[1] C. Hfer and G. Karagiannis, "Cloud computing services: taxonomy and comparison," *Journal of Internet Services and Applications*, vol. 2, pp. 81–94, 2011.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, pp. 50–58, apr 2010.

[3] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[4] L. Wu, S. K. Garg, and R. Buyya, "SLA-based admission control for a software-as-a-service provider in cloud computing environments," Uni. of Melbourne, Australia, Tech. Rep. CLOUDS-TR-2010-7, sep. 2010.

[5] Q. Li and Y. Guo, "Optimization of resource scheduling in cloud computing," in *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, sep. 2010, pp. 315 –320.

[6] G. Reig, J. Alonso, and J. Guitart, "Prediction of job resource requirements for deadline schedulers to manage high-level slas on the cloud," in *9th IEEE International Symposium on Network Computing and Applications (NCA)*, july 2010, pp. 162 –167.

[7] L. F. Bittencourt and E. R. M. Madeira, "HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, pp. 207–227, 2011.

[8] 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 *IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 400–407.

[9] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *10th IEEE International Conference on High Performance Computing and Communications (HPCC 2008)*, sept. 2008, pp. 5 –13.

[10] M. Alhamad, T. Dillon, and E. Chang, "Conceptual SLA framework for cloud computing," in *4th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, april 2010, pp. 606 –610.

[11] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.*, vol. 13, pp. 219–237, July 2005.