Using Time Discretization to Schedule Scientific Workflows in Multiple Cloud Providers

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, bit, edmundo}@ic.unicamp.br

***Abstract*—With the ever-increasing application demands, the execution of applications may require more resources than locally available. In this scenario, resources from multiple infrastructure-as-a-service (IaaS) cloud providers can be leased to fulfill application requirements. In this paper we deal with the problem of scheduling workflow applications in multiple IaaS providers, where the workflow scheduler must determine on which computational resource each component of a workflow should be allocated in order to minimize the involved monetary costs. We propose the use of different levels of discrete-time intervals in linear programming to schedule workflows with deadline constraints in multiple cloud providers. Simulations shown that increasing the granularity level of time-discretization decreases the scheduler running time, although yet achieving good solutions.**

***Keywords*-cloud computing; scheduling; workflow; ILP**

I. Introduction

E-Science and e-business applications are nowadays requiring an increasing computational power [1], [2], which has been exceeding the capacity available within the premises of a single organization. These complex applications comprise workflows, which can be decomposed into sets of smaller jobs that must be processed in a well-defined order. These data dependencies between jobs are commonly represented by a directed acyclic graph (DAG), where its nodes represent the service of a workflow, while its arcs represent data dependencies. These applications can be scheduled on the cloud providers, but optimal scheduling algorithms take too long to run for larger DAG instances due to the NP-Completeness of the scheduling problem.

We consider a scheduler modeled as integer linear program (ILP) for the scenario where the cloud customer submits his/her scientific workflow to be executed and managed by a workflow execution service (SaaS or PaaS, for instance) along with a response time (deadline) to be obeyed. Nevertheless, the input size of this scheduling problem increases with the DAG size and with the established deadline, since the ILP discrete-timeline becomes larger, making the scheduler running time increase considerably [3].

In this paper we propose the λ-*granularity* technique in order to modify the granularity of the schedule timeline when using ILP, aiming to quickly yield feasible scheduling solutions. By using different levels of discrete-time intervals in the timeline of the ILP, the input size of the schedule problem can be reduced, and the solver can be able to achieve solutions faster. The proposed technique is applied to a previously developed workflow scheduler [3], which was limited to small DAGs. Simulation results shown that increasing the granularity of the discrete-timeline is a useful mechanism to make the integer linear program to find feasible solutions for larger DAGs on a timely manner.

The next section shows related work. In Section III we present the considered workflow scheduling problem and the scheduling algorithm utilized in our evaluation, while Section IV shows the proposed time-discretization approach. Simulation results are discussed in Section V, while the final remarks and conclusions are presented in Section VI.

II. Related Work

Stefansson et al. propose in [4] a case study for large production scheduling. They present a comparison between two mathematical formulations for a pharmaceutical production scheduling problem as a mixed integer linear programming (MILP), one using a discrete-timeline and the other one a continuous-timeline. They concluded that using a continuous-time better represents the real-world decision problem and the solutions are more accurate. On the other hand, using a discrete-time makes the structure model simpler and complex constraints can be added without much effort. Floudas et al. overview in [1] the advances in MILP for the scheduling problem of chemical processing systems. They state that the solution accuracy would increase by reducing the length of the time intervals (slots), but not only the number of variables and constraints should be higher, but also the solver running time. Although these works present studies on time discretization of scheduling problems, none of them consider the cloud computing context.

Tordsson et al. describe in [5] an ILP formulation in order to optimize the placement of independent tasks in multiple cloud providers according to user-specified criteria. They consider a static cloud brokering scenario and also use IBM CPLEX solver[1] to evaluate simulations, in which execution times increase considerably as the number of tasks increases.

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

Van den Bossche et al. propose in [6] an ILP formulation of the independent task-scheduling to minimize the cost of external provisioning in hybrid clouds. In order to quickly obtain feasible solutions, they had to simulate the scheduling problem with low ($\approx$ 20) numbers of tasks. Although these works present advances on scheduling problems in cloud computing environment, none of them considers the execution of workflows. Moreover, the related work also shows limitations regarding to the number of tasks versus the scheduler execution times.

III. Background

The workflow scheduling problem in cloud environment can be tackled using either a static or a dynamic approach [2], [5]. The static approach is suitable for situations where the cloud provider has high resource availability and its conditions (resource prices, resource configurations, etc.) do not constantly change throughout the workflow execution life-cycle. Thus, the workflow scheduling can be performed off-line in advance to workflow deployment. On the other hand, the dynamic approach is more appropriate in scenarios where the conditions of uncertainty are high, such as variable resource prices, dynamic resource availability, variable bandwidth in network links, among others. In this manner, the scheduler algorithm should be run on-line to update the cloud environment information. Currently, cloud provider conditions tend to be static [5] because some parameters are rarely modified, such as virtual machine (VM) prices unusually change and resources availability is unknown but assumed to be high[2]. In this paper we focus on the static approach, which requires some data to be known a priori, while the on-line approach is already under investigation.

In this section we describe the ILP of the scheduler from [3]. It determines the VMs to be leased by a SaaS (or PaaS) provider from multiple IaaS providers to execute workflows meeting the customer's deadline and minimizing the VMs leasing costs. This scheduler is able to provide low-cost schedules, but its execution time is a drawback for larger workflows. Our proposal is incorporating the λ-granularity approach to this ILP, aiming to provide faster scheduling solutions. This proposal can be applied to other ILPs that use discrete variables to represent the schedule timeline.

This paper considers workflows whose services are dominated by computer processing time instead of I/O or memory access time. However, it is possible to modify the ILP to also consider workflows that can be I/O- or memory-bound rather than CPU-bound, because in those cases using only a faster VM can not yield an improvement in the scheduling. To achieve this, we would need to know information a priori about I/O and memory access of the workflow's services and the disk and memory speeds of the VMs. With such details, we can evaluate the time spent with I/O or memory operations, and select appropriate virtual machines. In short, these considerations are being evaluated as future work.

[2]Amazon EC2 make efforts to be available at least 99.95% during the Service Year, more details in http://aws.amazon.com/ec2-sla/

A. Notation and Problem Modeling

The following notations are used by the integer linear programming formulation of the problem [3]:

- 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 $u_i \in \mathcal{U}$, expressed as the number of instructions to be processed ($w_i \in \mathbb{R}^+$);
- $f_{i,j}$: number of data units to be transmitted between node n_i and n_j ($f_{i,j} \in \mathbb{R}^+$);
- $\mathcal{H}(j) = \{ij : i < j$, there exists an arc from vertex i to vertex j in the DAG$\}$: the set of immediate predecessors of n_j;
- $\mathcal{D}_\mathcal{G}$: 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 that compose the computing infrastructure of the problem;
- 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 ($\delta_i \in \mathbb{N}$).

Let $\mathcal{S}_i = \sigma_i^r \cup \sigma_i^o$ be the set of all service level agreements (SLAs) that have been signed between the software-as-a-service (SaaS) provider and the IaaS provider $i \in \mathcal{I}$. The $\mathcal{S}_i$ set is composed of two disjoint subsets: (i) the subset σ_i^r that includes SLAs for reserved VMs readily available, and (ii) the subset σ_i^o that 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 contains only VMs associated with prices for on-demand resources from IaaS $i \in \mathcal{I}$, i.e., $\mathcal{V}_i = \mathcal{V}_i^r \cup \mathcal{V}_i^o$ represents the set of VMs that can be leased from IaaS provider $i \in \mathcal{I}$. Each VM $v \in \mathcal{V}_i$ is associated with an element v_o of $\mathcal{V}_i^o$ and with an element v_r of $\in \mathcal{V}_i^r$. With this, the corresponding v_r and v_o of v have the same hardware configuration, but are billed at different price points. In general, the price per time unit of v_o is greater than or equal to v_r, being v_o and v_r related to the v.

Each SLA $s_r \in \sigma_i^r$ is related to only one type of VM $v \in \mathcal{V}_i^r$, and each SLA $s_o \in \sigma_i^o$ is related to only one type of VM $v \in \mathcal{V}_i^o$. One of the parameters defined in each contract agreement $s \in \mathcal{S}_i$ is the number $\alpha_s \in \mathbb{N}^+$ of VMs requested by the SaaS provider. Another SLA parameter indicates the duration time $t_s \in \mathbb{N}^+$ of the term commitment. The resources leased by the SaaS provider are presented in the set $\mathcal{V}$, which is defined as:

$$\mathcal{V} = \left\{ \bigcup_{i=1}^{m} \mathcal{V}_i \,, \quad \forall\, i \in \mathcal{I} : \mathcal{S}_i \neq \varnothing \right\} \tag{1}$$

where $m = |\,\mathcal{I}\,|$ and $\mathcal{V}_i = \mathcal{V}_i^r \cup \mathcal{V}_i^o$. In other words, according to the SLAs signed with each provider IaaS $i \in \mathcal{I}$, the

set $\mathcal{V}$ is composed. Moreover, we define the following IaaS characteristics that are also intrinsic for the ILP formulation:

- m: number of IaaS providers, $m \in \mathbb{N}$;
- $\mathcal{P}_v$: number of processing cores of $v \in \mathcal{V}$, $\mathcal{P}_v \in \mathbb{N}^+$;
- $\mathcal{J}_{v_q}$: 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 $\mathcal{L}_{v_g,v_h} \in \mathbb{R}^+$. If $v_g = v_h$, then $\mathcal{L}_{v_g,v_g} = \mathcal{L}_{v_h,v_h} = 0$;
- $\mathcal{B}_{i,v}$: binary variable that assumes a value 1 if v belongs to the IaaS provider $i \in \mathcal{I}$ (i.e. $v \in \mathcal{V}_i$), and 0 otherwise.
- $\mathcal{C}_v$: price to lease $v \in \mathcal{V}$ for 1 time unit, with $\mathcal{C}_v \in \mathbb{R}^+$.

Vendor lock-in problem is still an open issue in cloud computing environments [7]. This problem restricts users from migrating their applications to a different cloud provider. Although this work considers multiple clouds, we assume that all providers use an open-standard interface, i.e., open cloud computing interface (OCCI)[3], to describe applications that are compatible across multiple cloud providers, thus we consider the cloud lock-in problem to be beyond the scope of this paper. Let $\zeta = \{\sigma_1^r, \ldots, \sigma_m^r\}$ be the set that contains all SLAs associated with reserved VMs. We define the variable $\mathcal{K}_{s,v}$ that assumes the value 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 $v \in \mathcal{V}_i^r$.

B. Formulation of the Integer Linear Program

The ILP solves the scheduling problem through the binary variables x and y and the constant $\mathcal{C}_v$ [3]:

- $x_{u,t,v}$: 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.

Before formulating a mathematical model for any scheduling problem, it is necessary to take into consideration the *timeline* representation. Time can be discrete or continuous values [1]. Workflow scheduling usually requires the use of discrete variables to represent discrete decisions. For instance, resource assignment and atomic tasks allocation over time are some activities that represent discrete decisions involved in the scheduling problem [4]. Moreover, as in *Amazon EC2*, partial VM-hours consumed are billed as full hours. Therefore, the utilized ILP formulation only considers discrete-time representation. In Section IV we discuss how we can contour these problems to make the ILP applicable in practice. 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 $\mathcal{D}_\mathcal{G}$. The ILP is formulated as follows [3]:

[3]More details in http://occi-wg.org

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

$$\text{(C1)} \quad \sum_{s=1}^{t-\lceil w_z\times\mathcal{J}_r + f_{u,z}\times\mathcal{L}_{i,j}\rceil} x_{u,s,v} \geq \sum_{s=1}^{t} x_{z,s,r}$$
$$\forall\, z \in \mathcal{U}, \quad \forall\, u \in \mathcal{H}(z), \quad \forall\, r, v \in \mathcal{V},$$
$$\forall\, t \in \mathcal{T}, \quad \forall\, i, j \in \mathcal{I} \mid \mathcal{B}_{i,v} = 1, \quad \mathcal{B}_{j,r} = 1$$

$$\text{(C2)} \quad \sum_{u\in\mathcal{U}} \sum_{s=t:t\leq\mathcal{D}_\mathcal{G}-\lceil w_u\times\mathcal{J}_v\rceil}^{t+\lceil w_u\times\mathcal{J}_v\rceil-1} x_{u,s,v} \leq \mathcal{P}_v$$
$$\forall\, v \in \mathcal{V}, \quad \forall\, t \in \mathcal{T}$$

$$\text{(C3)} \quad \sum_{s=t-\lceil w_u\times\mathcal{J}_v\rceil+1}^{t} y_{s,v} \geq x_{u,t,v} \times \left(\lceil w_u \times \mathcal{J}_v\rceil\right)$$
$$\forall\, u \in \mathcal{U}, \quad \forall\, v \in \mathcal{V}, \quad \forall\, t \in \{\lceil w_u \times \mathcal{J}_v\rceil, \ldots, \mathcal{D}_\mathcal{G}\}$$

$$\text{(C4)} \quad \sum_{p=t+1}^{s-\lceil w_z\times\mathcal{J}_r\rceil} \left(y_{p,v} + y_{p,r}\right) \geq 2 \times \left(x_{u,t,v} + x_{z,s,r} - 1\right) \times \pi$$
$$\forall\, z \in \mathcal{U}, \quad \forall\, u \in \mathcal{H}(z), \quad \forall\, r, v \in \mathcal{V}, \quad \forall\, s, t \in \mathcal{T},$$
$$\text{s.t.} \quad \pi = \left(s - t - \lceil w_z \times \mathcal{J}_r\rceil\right) \geq \lceil f_{u,z} \times \mathcal{L}_{i,j}\rceil,$$
$$i, j \in \mathcal{I}, \quad \mathcal{B}_{i,v} = 1, \quad \mathcal{B}_{j,r} = 1$$

$$\text{(C5)} \quad \sum_{t\in\mathcal{T}}\sum_{v\in\mathcal{V}} x_{u,t,v} = 1 \quad \forall\, u \in \mathcal{U};$$

$$\text{(C6)} \quad \sum_{u\in\mathcal{U}}\sum_{v\in\mathcal{V}} \sum_{t=1}^{\lceil w_u\times\mathcal{J}_v\rceil} x_{u,t,v} = 0$$

$$\text{(C7)} \quad \sum_{v\in\mathcal{V}} y_{t,v} \leq \delta_i \quad \forall\, i \in \mathcal{I}, \forall\, t \in \mathcal{T}, \text{ s.t. } \mathcal{B}_{i,v} = 1$$

$$\text{(C8)} \quad \sum_{v\in\mathcal{V}} y_{t,v} \leq \alpha_s \quad \forall\, s \in \zeta, \forall\, t \in \mathcal{T}, \text{ s.t. } \mathcal{K}_{s,v} = 1$$

$$\text{(C9)} \quad x_{u,t,v} \in \{0,1\} \quad \forall\, u \in \mathcal{U}, \forall\, t \in \mathcal{T}, \forall\, v \in \mathcal{V}$$

$$\text{(C10)} \quad y_{t,v} \in \{0,1\} \quad \forall\, t \in \mathcal{T}, \forall\, v \in \mathcal{V}$$

The constraint (C1) 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 (C2) 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 (C3) 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 (C4) determines that a VM must stay active while it is sending (or receiving) data generated by precedent nodes. This constraint was added to the original formulation from [3] in order to charge the VMs usage during data transfers between workflow components. The constraint (C5) specifies that a service must be executed only once and in a single VM. The constraint (C6) establishes that a DAG node u cannot be set as finished until it has been executed in a VM v. The constraint (C7) specifies that the number of reserved VMs plus the number of on-demand VMs cannot exceed

the maximum number allowed by each IaaS provider [8]. The constraint (C8) establishes that the amount of virtual machines being used cannot exceed the limit stipulated in the SLA. The last two constraints, (C9) and (C10), specify that the variables of this integer linear program, namely x and y, will only assume binary values.

In the next section we discuss the representation of time, proposing the use of wider time intervals in the set $\mathcal{T}$ to reduce the size of the search space when solving the ILP.

IV. Representation of Time

To formulate a mathematical model for any scheduling problem, one major issue that arises is how to represent the timeline of the application execution. In general, time can be classified into two main categories [1], [4]: discrete-time and continuous-time. In the first one, schedulers are formulated through the approach of time-discretization, i.e., the timeline is divided into a number of equally spaced intervals. Thereupon, the events (scheduler decisions) can only take place at the beginning (or the end) of those intervals. On the other hand, with a continuous-time, the scheduling decisions are made at any instant of time within the timeline. Although the continuous-time increases accuracy of the scheduler, i.e., it presents a finer granularity, the mathematical ILP model becomes more complex [9], as well as it takes more time to obtain optimal solutions.

A. The λ-granularity Approach

The scheduler (ILP solver) execution time can be enlarged by the granularity level of the timeline, by the user desired response time (deadline), as well as by the DAG size.

When small lengths for the time intervals need to be used and/or the timeline under consideration is too long, the number of discretized-time intervals in the set $\mathcal{T}$ can be very large [10]. Thereby, the workflow scheduler leads to very large combinatorial problems, which are computationally very expensive to solve or even intractable in a timely manner. The parameter (or dimension) *time* is the main bottleneck that prevents scalability in the scheduler presented in [3]. Due to the precedence constraints between the DAG nodes and the NP-Completeness of the scheduling problem, both the solver execution time and the time set $\mathcal{T}$ grow quickly when we try to schedule DAGs with large number of nodes and dependencies. In other words, the main variable of the ILP is a three-dimensional binary-matrix ($x_{u,t,v}$: node u, time t, and virtual machine v), and therefore the larger the binary matrix, the higher the solver execution time. In this paper we propose *increasing the granularity of the time-discretization* in order to reduce the size of the binary-matrix $x_{u,t,v}$. With time discretized into larger units, the size of the time set $\mathcal{T}$ is reduced and consequently the size of the binary matrix is also reduced. Therefore, with a smaller $\mathcal{T}$, the solver execution time tends to decrease. We define the proposed λ-granularity approach as the *multiplicative factor* that determines the size of discrete-time intervals, and the set $\mathcal{T}$ is rewritten as in equation (2).

$$\mathcal{T} = \left\{ \lambda, 2\lambda, 3\lambda, 4\lambda, \ldots, \Lambda \right\} \quad \Big| \quad \Lambda \leq \mathcal{D}_{\mathcal{G}} \quad \text{and} \quad \lambda \in \mathbb{N}^{+} \quad (2)$$

Short λ-granularity (fine-grained time intervals) or long time-horizon (high deadline) can considerably increase the amount of discrete-time intervals in the set $\mathcal{T}$. In other words, if the discretization intervals are fine enough to capture all significant events, then the number of ILP constraints can become very large and the workflow scheduling problem can be computationally expensive to solve. Thus, the finer the λ-granularity, the greater the solution search space and, consequently, more time is spent by the scheduler to find a feasible solution with reduced monetary costs involved.

However, large λ-granularity (coarse-grained time intervals) presents a drawback: they can substantially decrease the size of $\mathcal{T}$ and, thus, the scheduling may become unfeasible because the scheduler will not find sufficient discrete-time units for all DAG nodes. Furthermore, the coarse λ-granularity may increase the monetary cost of feasible solutions, because as the size of the set $\mathcal{T}$ is reduced, the scheduler is forced to choose faster virtual machines (and consequently more expensive) to schedule all DAG nodes before the deadline, leading to an expensive workflow execution and resources (or money) wastefulness. Therefore, there is a clear trade-off between using small discrete-time intervals to obtain a more precise scheduling or use long discrete-time intervals to reduce the solver execution time. We aim to find solutions faster with monetary costs reduced.

B. Correcting Estimation Errors Introduced by The Time-discretization Approach

Reducing the amount of discrete-time intervals in the time set $\mathcal{T}$ to reduce the solver execution time produces a penalty in the monetary execution cost of the schedule. When we increase the λ-granularity, we force the solver to pick faster (and more expensive) VMs. Thus, the principle of the time-discretization approach goes in the opposite direction of the ILP objective function, which tries to minimize the monetary cost involved in the workflow scheduling. For instance, one discrete-time interval when $\lambda = 2$ is equivalent to two discrete-time intervals when $\lambda = 1$; i.e., if the solver schedules a DAG node u to be executed in only one discrete-time interval when $\lambda = 1$, then this same DAG node u will also be executed in only one discrete-time interval when $\lambda = 2$. With that, there will be an additional monetary execution cost of one time unit unused in the VM where DAG node u was scheduled when $\lambda = 2$, increasing the total monetary execution cost involved. Thus, for each $\lambda > 1$ value, the solver will provide an estimate (upper bound) of the monetary cost of the actual workflow execution.

The solver introduces empty spaces in the schedule that would be charged regardless of its use, since the VMs

Table I
IAAS PROVIDERS A, B, AND C

Provider	Type	Core	Core Performance	On-demand Price	Reserved Price
A	Small	1	1.5	$0.13	$0.045
	Medium	2	1.5	$0.20	$0.070
B	Small	1	2	$0.17	$0.045
	Medium	2	2	$0.30	$0.059
	Large	3	2	$0.40	$0.140
	XLarge	4	2	$0.52	$0.183
	Dbl. X-Large	8	2	$0.90	$0.316
C	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

would remain leased during these times. This issue occurs because the solver "believes" that one discrete-time interval, in $\lambda = 2$ for instance, will be entirely utilized by the node that is assigned to it. Thus, given a feasible workflow scheduling solution in $\lambda > 1$, we need to (without DAG rescheduling) recompute (i) the finish time for each DAG node (makespan); and (ii) the total monetary computation cost value for $\lambda = 1$ in order to reduce the monetary cost error embedded by the λ-discretization approach.

The general λ-discretization algorithm is shown in Figure 1. In the first line of the algorithm, the solver is called with a $\lambda > 1$ value to reduce the size of $\mathcal{T}$. If a solution is found, the algorithm then starts the recomputation of the schedule in order to compute the final makespan and cost results. Line 3 stores the schedule given by the solver call, and line 4 calls the solver again, but now with $\lambda = 1$ and with the schedule already fixed as stored in line 3. At the end, this will result in the recomputation of the schedule by suppressing empty spaces left when the solver was first called with $\lambda > 1$. Note that we do not change the mapping-task from the first solver call to the second solver call because we would be returning to the initial problem with $\lambda = 1$.

Require: DAG $\mathcal{G} = \{\mathcal{U}, \mathcal{E}\}$, deadline $\mathcal{D}_\mathcal{G}$, set of IaaSs $\mathcal{V}$
Ensure: Schedule of $\mathcal{G}$ in $\mathcal{V}$ or *Null* for unfeasible solution

1: Call ILP solver with $\lambda > 1$
2: **if** There is any feasible schedule of $\mathcal{G}$ in $\mathcal{V}$ **then**
3: Save DAG schedule data (for each node $u \in \mathcal{U}$, save only which VM $v \in \mathcal{V}$ the node u was scheduled).
4: Recall ILP solver with DAG schedule data (step 3) and with $\lambda = 1$ only to recompute (i) the finish execution time for each node $u \in \mathcal{U}$ and (ii) the total monetary execution cost involved
5: **end if**
6: Return the solution from the last solver call

Figure 1. The time-discretization technique.

Although the monetary execution cost can be reduced in the second solver call, it will still be an upper bound of the optimal solution. Therefore, as in other sub-optimal approaches following the NP-completeness of the workflow scheduling problem, this is a price we must pay to try to get a feasible solution quickly instead of (indefinitely) waiting for the solver to find the optimal solution. Here, we aim at presenting the calibration of the time-discretization as a tool to achieve faster feasible solutions using ILP.

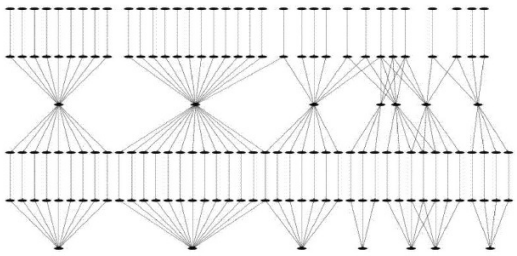


Figure 2. LIGO DAG topology

V. EVALUATION

We implemented the ILP in JAVA and conducted simulations using the IBM ILOG CPLEX with default configuration. Our evaluation comprises simulations with DAGs of real world applications such as fork-join and LIGO [11] (Figure 2); moreover, we use three different IaaS providers. This evaluation was also performed for Montage and fork-join DAGs, but we focus on a more comprehensive evaluation of the LIGO DAG in this section.

A. Simulation Configuration

We used 3 IaaS providers in our simulations, each one with its own prices for reserved and on-demand VMs. Table I shows virtual machines options respectively for IaaS providers A, B and C. We assume that the existing SLA contracts for reserved VMs (i.e., the elements of the set ζ) comprise 4 VMs: 1 *Small* and 1 *Medium* instance from both providers A and B. The maximum number of virtual machines that can be leased from each IaaS provider was set as follows: $\delta_A = 4$, $\delta_B = 7$, $\delta_C = 2$.

The public clouds usually do not provide information about the quality of service of internal links (between virtual machines) and external links (between IaaS providers). We assume that the bandwidth of internal links is larger than the bandwidth of external links, which is a reasonable assumption in real environments. This is reflected in our simulation by randomly generating a $\mathcal{L}$ in the $[2, 3]$ real interval for external links, while for internal links the set $\mathcal{L}$ is taken from the $[0.1, 0.2]$ real interval. Therefore, through the VMs already reserved and the ones that can be leased on-demand, our goal is that the overall solver execution time (solver time of workflow scheduling plus solver time of workflow cost recomputation) is small and, moreover, with lowest possible workflow monetary execution cost.

B. Results

Simulations were run on an Intel® Core™ Xeon $X5650$ CPU 2.67GHz and 16GB of RAM. We have run 30 sim-

ulations for each DAG with the IaaS provider configurations shown in Table I, plus the 4 reserved VMs. In each simulation, the computation cost for each DAG node and the communication cost for each DAG node dependency were randomly taken from the $[1, 3]$ real interval. We varied the λ-granularity to evaluate the relation between *workflow monetary execution cost* and *solver execution time* metrics. Meanwhile, we also provide data metrics of *workflow makespan* and *number of unfeasible solutions* to show the effect of the variation of λ on these metrics. Furthermore, in order to obtain a scheduling solution, the solver had its running time limited to 10 minutes for each simulation. After that time, the solver returned the best solution so far, if any. The time limit of 10 minutes was chosen because the majority of feasible solutions for the LIGO and 50-nodes fork-join DAGs with $\lambda = 1$ start to be found in the first 9 minutes of the solver execution.

To represent possible deadlines, 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 DAG nodes on a single resource. Deadlines of $T_{max} \times 1/7$ showed only unfeasible solutions for all solver executions, while $\mathcal{D}_\mathcal{G}$ of $T_{max} \times 7/7$ can be achieved by putting all tasks in the cheapest resource. The divisor 7 was chosen only to assess the evolution of the λ value with increasing $\mathcal{D}_\mathcal{G}$, so other dividers could be used.

Figure 4 shows the evolution of the solver execution time for different DAGs with the original scheduler from [3]. When we increased the size of the DAG (number of nodes and dependencies between nodes), we indirectly also increased (i) the number of the discretized-time intervals on set $\mathcal{T}$; (ii) the number of possible workflow scheduling solutions; and (iii) the solver execution time. For instance, for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$, the solver took, on average, 1.15, 141.43, 1652.92 e 3456.34 seconds to find a feasible solution to the DAG *fork-join* with 10, 20, 30 nodes and the DAG *Montage* [3], respectively. In these simulations, the solver also had its running time limited (to more than 10 minutes; details in [3]), and therefore the majority of the solutions were non-optimal. Therefore, increasing discrete-time intervals aims to reduce the size of set $\mathcal{T}$ and the solver execution time.

The averages shown in the following graphs are over 30 DAG-simulations for each deadline $\mathcal{D}_G$. The λ-granularity proposal is compared with the original scheduler from [3], shown as $\lambda = 1$ in the result graphs.

1) Fork-Join DAG with 50*-nodes:* Figure 3 shows results for simulations with 50-nodes fork-join DAGs. In these simulations, the integer values used for λ are in the set $\{1, 2, \ldots, 9, 10\}$, where $\lambda = 1$ corresponds to the results for the original scheduler. The solver running time started to decrease only when $\lambda \geq 4$. Without fixing the monetary execution cost error (*1st Call ILP solver* in the graphs), the higher the λ value, the lower the solve running time and the greater the workflow monetary execution cost is. For

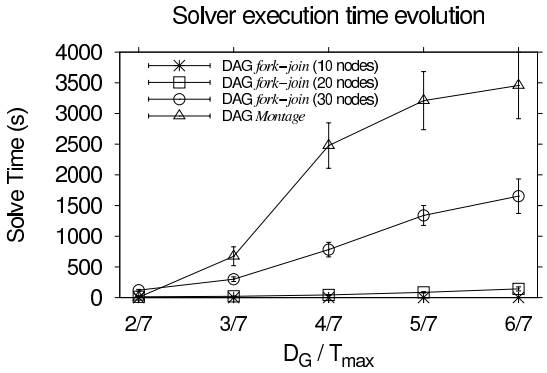
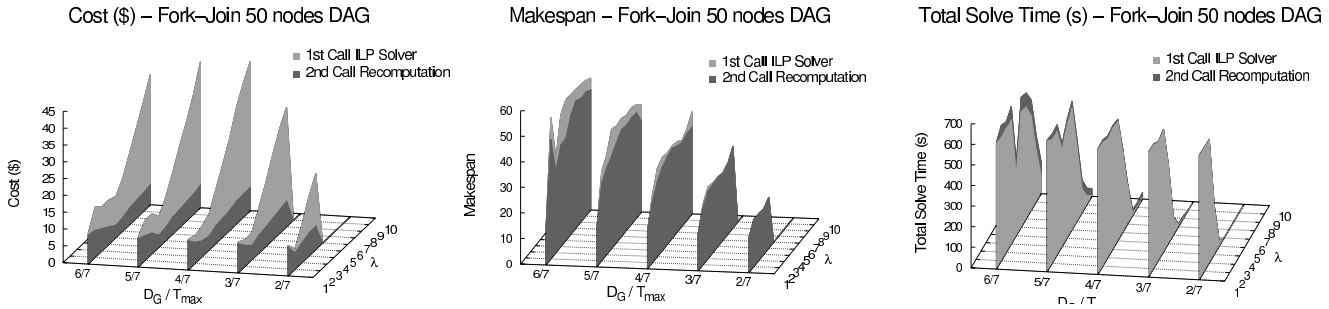


Figure 4. Solver running time with increasing DAG size.

instance, comparing the feasible solutions between $\lambda = 1$ and $\lambda = 5$, we had a reduction of approximately from 50% (for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$) to 96% (for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$) in the average solver execution time and, on the other hand, an increase of approximately from 35% (for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$) to 149% (for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$) in the average monetary computation cost. For these feasible solutions, when the solver recomputes (*2nd Call Recomputation* in the graphs) the monetary execution costs using $\lambda = 1$ instead of $\lambda = 5$, it was able to reduce significantly the monetary cost error embedded by the time-discretization approach. For instance, it was able to reduce the average monetary costs from 68% (for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$) to 70% (for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$). Nevertheless, this DAG-recomputation increased the total solver running time from only 18% (for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$) to 20% (for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$).

When we compare the solutions found with $\lambda = 10$ (2nd solver call) with the solutions found in $\lambda = 1$ (original ILP scheduler), we were able to get quick solutions and with reasonable monetary costs. For instance, for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$ and $\lambda = 10$ (with recomputation), the solver was able to find solutions with an average cost 34% lower, and with the solver execution time reduced by 75% when compared to the solutions of the original scheduler for the same $\mathcal{D}_\mathcal{G}$. Thus, increasing the λ-granularity made it possible to reduce both the monetary execution cost and the solver running times. Note that for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$, the solver was able to find solutions only when $\lambda \leq 5$, due to the short deadline. Increasing the λ value, followed by deadline reduction, reduces the timeline-horizon (i.e., reduces the size of the time set $\mathcal{T}$), and therefore increases the number of unfeasible solutions, as shown in Figure 6[4]. In short, the solver does not find sufficient discrete-time intervals for all DAG nodes. Note that when the percentage of unfeasible solutions is 100%, the values of the monetary cost and the

[4]Figure 6 shows only unfeasible solutions when we call ILP solver with $\lambda = 1$ to schedule the DAG $\mathcal{G}$ (step 1 of the algorithm), not when we recall ILP solver for the DAG-recomputation (step 4 of the algorithm). Furthermore, for all feasible solutions found in step 1, the solver was able to find a feasible solution with reduced monetary cost in step 4. Thus, the unfeasible solutions graph of step 4 is the same as step 1.



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

Figure 3. Results for the Fork-Join DAG with 50 nodes.

makespan are zero because no VMs have been used.

The DAG-recomputation does not increase the original makespan value found in the first call of the solver. Due to fine-grain used in the second call of the solver, in some cases, there could be a decrease in the makespan, because the solver can be able to suppress empty spaces in the DAG nodes allocation over the timeline.

2) LIGO DAG: When scheduling the LIGO DAGs, with 168 nodes, the ILP solver was able to find feasible solutions in a timely manner *only* when increasing the λ-granularity, as shown in Figure 5. We simulated the LIGO DAG as follows:

- For $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$ and $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 3/7$, we use the following integer values for λ: $\{5, 6, \ldots, 14, 15\}$;
- For $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 4/7$, we use the following integer values for λ: $\{10, 11, \ldots, 19, 20\}$;
- For $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 5/7$ and $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$, we use the following integer values for λ: $\{15, 16, \ldots, 24, 25\}$

For each deadline, these λ values were chosen according to the first simulations of the LIGO DAG, and as shown in Figure 6, no deadline had 100% viable solutions for all λ values. Furthermore, by using short discrete-time intervals (fine-grained, i.e. $\lambda \leq 5$), the solver could not find feasible solutions in a timely manner for any deadline. Feasible solutions to the LIGO DAG were possible only with a significant reduction in the size of the time set $\mathcal{T}$ (or a significant increase in the discrete-time intervals, with high λ values). For instance, for the short deadline $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$, the LIGO DAG scheduling was only possible for $\lambda \in \{6, \ldots, 9\}$, and the ILP solver has yet been aborted by the timeout of 10 minutes for all those λ values. Regardless of the deadline $\mathcal{D}_\mathcal{G}$, due to the size of the LIGO DAG, even increasing the λ-granularity , the solver was aborted by the the 10 minutes timeout in all simulations, returning the best feasible solution found so far.

For the recomputed solutions in $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$, all the simulations had similar solver running times (already considering the overhead added by the DAG-recomputation). Furthermore, for $\lambda = 9$, we had solutions with the lowest average monetary cost, which was approximately 1.8%, 17% 18% lower than the average monetary costs of the solutions found for λ values equal to 8, 7 and 6, respectively. The

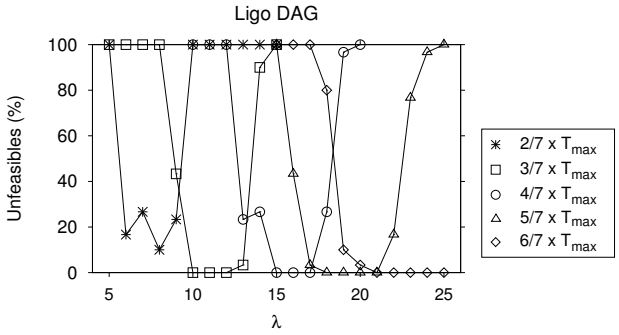
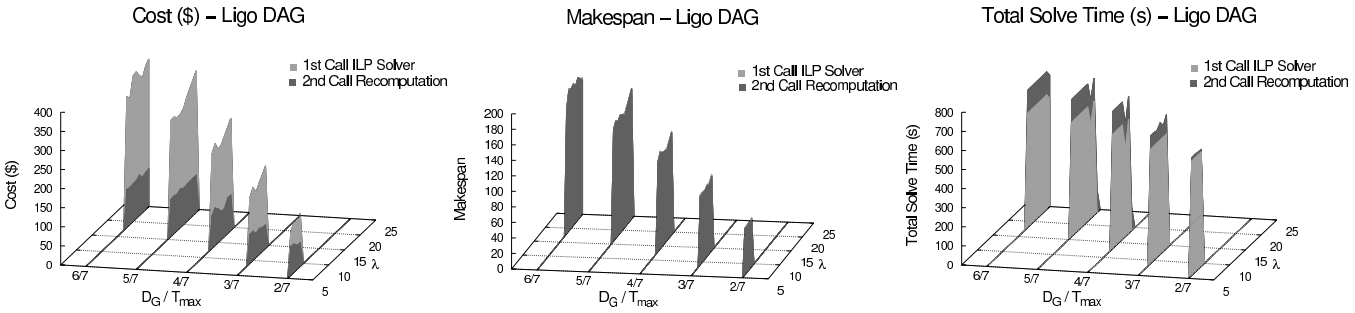


Figure 6. Unfeasible solutions.

recomputation of the estimated costs significantly reduces the error embedded by the λ-granularity approach without significant increase in the solve time, i.e., the correction in the average monetary cost was from 35% (for $\lambda = 6$) to 56% (for $\lambda = 9$) of the feasible solutions initially found.

For $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 3/7$, it was possible to find feasible solutions only when $\lambda \in \{9, \ldots, 14\}$, and for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$ only when $\lambda \in \{18, \ldots, 25\}$. In other words, we need high integer values for λ in order to get a feasible solution in a timely manner. Regarding the deadline $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$ and independently of the λ value, all the simulations had a similar overhead added by the DAG-recomputation in the average solver running time (approximately 20% of the timeout). On the other hand, we have had a significant reduction in the monetary cost. For instance, the solver was able to reduce the average monetary execution cost from 70% (for $\lambda = 18$) to 75% (for $\lambda = 25$) of the feasible solutions initially found. Besides that, the different levels of time-discretization can help us choose a specific value for λ that generates the lowest monetary cost for a given deadline $\mathcal{D}_\mathcal{G}$, e.g., $\lambda = 17$ for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$ and $\lambda = 23$ for $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$.

Results presented for both LIGO and fork-join DAGs suggest that depending on the size of the DAG and the deadline, we need to increase the discrete-time intervals in order to quickly get a feasible solution. In addition, a particular λ value may be chosen from a historical of



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

Figure 5. Results for the LIGO DAG.

workflows executions to return a scheduling solution with a reasonable monetary cost, based on parameters such as workflow size (number of nodes and dependencies between nodes) and deadline. Moreover, similar results were obtained for the Montage and for 30-nodes fork-join DAGs.

VI. Conclusion

In this paper we presented λ-granularity, a discretization-time approach to decrease the scheduler running time by using different levels of discrete time fragmentation in the timeline of the ILP. We performed simulations with DAGs that represent real-world applications. With the original ILP scheduler ($\lambda = 1$) and large deadlines, the size of the set of discrete time intervals $\mathcal{T}$ can be large enough to prevent a viable solution to be found in a timely manner in the cloud computing context. The results of our proposal suggest that increasing the λ-granularity ($\lambda > 1$) can be effective in reducing the solver execution time, yet achieving good (low-cost) solutions. Besides that, the discretization-time approach can help the service providers of workflow execution to negotiate SLAs with its customers. Future work includes the determination of an ideal λ-granularity according to the scheduler input, on-line dynamic scheduling approach and take the I/O- or memory-bound workflows into consideration.

Acknowledgment

The authors would like to thank CAPES, CNPq, and FAPESP (2012/02778-6) for the financial support, and IBM for providing the CPLEX tools.

References

[1] C. Floudas and X. Lin, "Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications," *Annals of Operations Research*, vol. 139, pp. 131–162, 2005.

[2] L. F. Bittencourt, E. R. M. Madeira, and N. L. S. da Fonseca, "Scheduling in hybrid clouds," *Communications Magazine, IEEE*, vol. 50, no. 9, pp. 42 –47, september 2012.

[3] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, "Workflow scheduling for SaaS / PaaS cloud providers considering two SLA levels," in *Network Operations and Management Symposyum (NOMS)*, april 2012, pp. 906 –912.

[4] H. Stefansson, S. Sigmarsdottir, P. Jensson, and N. Shah, "Discrete and continuous time representations and mathematical models for large production scheduling problems: A case study from the pharmaceutical industry," *European Journal of Operational Research*, vol. 215, no. 2, pp. 383 – 392, 2011.

[5] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Journal of Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, feb 2012.

[6] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads," in *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, july 2010, pp. 228 – 235.

[7] P. Harsh, F. Dudouet, R. Cascella, Y. Jegou, and C. Morin, "Using open standards for interoperability issues, solutions, and challenges facing cloud computing," in *8th International Conference on Network and Service Management (CNSM)*, oct. 2012, pp. 435 –440.

[8] L. Wu, S. K. Garg, and R. Buyya, "SLA-based admission control for a software-as-a-service provider in cloud computing environments," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1280 – 1299, 2012.

[9] K. L. Yee and N. Shah, "Improving the efficiency of discrete time scheduling formulation," *Journal of Computers Chemical Engineering*, vol. 22, Supl. 1, pp. S403 – S410, 1998.

[10] C. A. Floudas and X. Lin, "Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review," *Computers & Chemical Engineering*, vol. 28, no. 11, pp. 2109 – 2129, 2004.

[11] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi, "Scheduling data-intensive workflows onto storage-constrained distributed resources," in *Intl. Symposium on Cluster Computing and the Grid (CCGRID)*, Rio de Janeiro, Brazil, 2007, pp. 401–409.