Discretização do Tempo na Utilização de Programação Linear para o Problema de Escalonamento de Workflows em Múltiplos Provedores de Nuvem

Thiago A. L. Genez[1], **Luiz F. Bittencourt**[1], **Edmundo R. M. Madeira**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Av. Albert Einstein, 1252 – 13.083-852 – Campinas – SP – Brasil

{thiagogenez, bit, edmundo}@ic.unicamp.br

Abstract. *Complex applications in e-science and e-business are usually modeled as workflows which execution in clouds can minimize the upfront investment in computational resources. Therefore, the development of algorithms to schedule workflows in clouds is of major interest. In this paper we propose and evaluate the use of different levels of timeline discretization utilizing linear programming to solve the problem of scheduling workflows with deadline constraints in multiple cloud providers. Simulations show that increasing the granularity of time discretization decreases the scheduler execution time, making it possible to achieve solutions faster when scheduling workflows that present larger execution times.*

Resumo. *Aplicações complexas de e-Ciência e e-Business são geralmente modeladas como workflows cujas execuções em nuvens podem minimizar investimentos em infraestrutura computacional. Então, o desenvolvimento de algoritmos para escalonamento de workflows em nuvens é de grande interesse. Neste trabalho propomos e avaliamos o uso de diferentes níveis de discretização da linha do tempo por meio da programação linear para resolver o problema de escalonamento de workflows com deadline em múltiplos provedores de nuvem. Experimentos mostram que o aumento da granularidade da discretização do tempo diminui o tempo de execução do escalonador, possibilitando alcançar soluções mais rápidas no escalonamento de workflows que apresentam tempos de execução maiores.*

1. Introdução

Computação em nuvem emergiu como um novo paradigma, onde hardware e software são entregues como serviços de utilidade geral e disponibilizados para os usuários através da Internet. Graças à camada de virtualização, construída sobre os recursos computacionais físicos, é possível otimizar o uso da infraestrutura e, principalmente, oferecer diferentes tipos de hardware e software aos usuários da nuvem. Dessa maneira, a computação em nuvem permite que os recursos (serviços) sejam alugados e liberados de acordo com a necessidade dos usuários e tarifados por meio do modelo "pago-pelo-uso" (do inglês, *pay-as-you-go*) [Zhang et al. 2010].

Várias aplicações complexas de e-Ciência e *e-Business* podem ser modeladas como *workflow*, compondo um conjunto de serviços a serem processados em uma ordem bem-definida [Papuzzo and Spezzano 2011]. Além disso, cada serviço pode depender de dados computados por outros serviços que o antecedem. Dessa forma, *workflows*

são geralmente representados por um grafo acíclico direcionado (*directed acyclic graph* - DAG) para representar os serviços e suas dependências. Escalonamento de *workflow* é um problema NP-Completo e, por consequência disso, aprimoramentos em algoritmos e heurísticas são fundamentais para melhorar o desempenho de sua execução em nuvens.

Desenvolvemos uma formulação linear inteira (PLI) para escalonar *workflows* em nuvens [Genez et al. 2012] cuja execução é realizada por um provedor de software como serviço (*Software as a Service* – SaaS). Nesse trabalho, o cliente envia seu *workflow* para ser executado pelo provedor de SaaS, juntamente com um tempo de resposta (*deadline*) a ser obedecido. Entretanto, dependendo do tamanho do DAG e do *deadline*, a linha temporal discreta da PLI torna-se grande e, portanto, aumenta o tempo de execução do escalonador. Assim, esse escalonador fica limitado somente a DAGs pequenos (≈ 25 nós), pois não é possível encontrar soluções viáveis em tempo hábil para DAGs maiores.

A contribuição deste trabalho está na proposta e avaliação do uso de diferentes níveis de granularização da linha do tempo no programa linear inteiro que resolve o problema de escalonamento de *workflow* na computação em nuvem. O principal objetivo é, portanto, mostrar que o aumento da granularidade da discretização do tempo permite ao escalonador encontrar rapidamente soluções viáveis ao DAGs com grande número de nós e dependências; e, ao mesmo tempo, minimizar os custos monetários com a terceirização de recursos computacionais. Por uma questão de clareza, utilizamos apenas SaaS para referenciar ao provedor de nuvem, porém este trabalho também aplica-se aos provedores de plataforma como serviço (*Plataform as a Service* – PaaS).

O restante deste artigo está organizado da seguinte maneira. Conceitos básicos e o problema do escalonamento são descritos na Seção 2, enquanto trabalhos relacionados são expostos na Seção 3. Uma visão geral do algoritmo usado no escalonador é apresentado na Seção 4. A Seção 5 descreve a discretização da linha do tempo proposta a esse escalonador, enquanto resultados experimentais são analisados na Seção 6. A Seção 7 finaliza este artigo com as conclusões e trabalhos futuros.

2. Conceitos Básicos

Nesta seção introduzimos brevemente os conceitos básicos sobre a representação de *workflows* e o cenário do escalonamento de *workflows* em nuvens deste trabalho.

2.1. Representação de *Workflow*

Um *workflow* é geralmente representado por um grafo acíclico direcionado (DAG) $\mathcal{G} = \{\mathcal{U}, \mathcal{E}\}$, onde cada nó $n_i \in \mathcal{U}$ representa um serviço a ser executado e cada aresta $e_{i,j} \in \mathcal{E}$ representa uma dependência de dados entre o serviço i e j, ou seja, $e_{i,j}$ são os dados produzidos por n_i e consumidos por n_j. Isto significa que n_j pode iniciar a sua execução somente após n_i finalizar e enviar todos os dados necessários a n_j.

A Figura 1 mostra a submissão de um DAG que será executado pelo provedor de SaaS. Esse DAG é denominado *fork-join* e pode representar um aplicativo de processamento do filtro de mediana de imagens [Bittencourt and Madeira 2011]. Nessa figura, os rótulos dos nós e das arestas representam, respectivamente, custos de computação (número de instruções, por exemplo) e comunicação (bytes para transmitir, por exemplo). Esse DAG é composto por 14 nós, onde o nó 1 representa a operação de divisão

(*fork*) da imagem a ser processada e é o primeiro nó a ser executado. Os nós $2 - 13$ representam a função de processamento do filtro de mediana e somente podem iniciar suas execuções depois que receberem os dados do nó 1. Após o término das execuções e o envio dos dados dos nós $2 - 13$, o nó 14 pode começar sua execução que representa a operação de união (*join*) das imagens inicialmente particionadas.

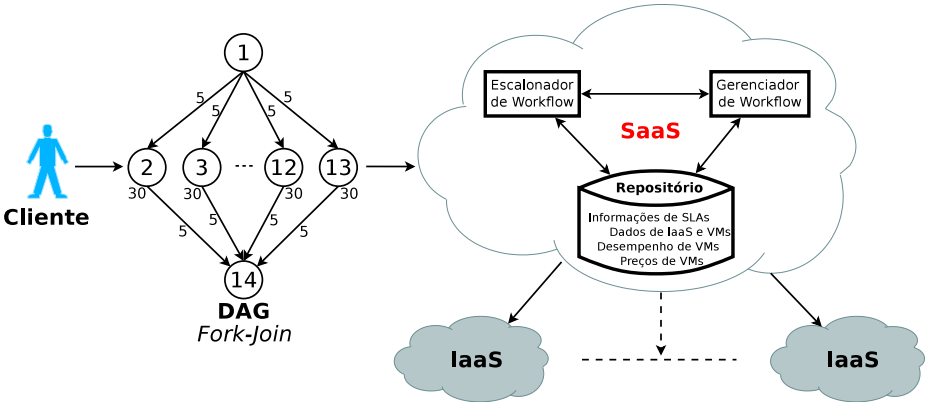


Figura 1. Submissão do DAG *fork-join* com 14 nós a ser executado pelo SaaS

Assumimos, sem perda de generalidade, que todo DAG tem apenas um nó de entrada (primeiro nó) e um nó de saída (último nó). Também assumimos que cada nó $n_i \in \mathcal{U}$ é indivisível e é executado em um único núcleo de processamento virtualizado. Todos os *workflows* considerados neste trabalho são estáticos, isto é, a topologia do DAG não é alterada durante o escalonamento.

O problema de escalonamento de *workflow* consiste em, dado um conjunto de tarefas/serviços dependentes e suas dependências, escolher em qual recurso computacional cada tarefa/serviço irá executar. Geralmente, um escalonador possui uma função objetivo, como por exemplo, minimizar o tempo de execução (*makespan*), maximizar o uso de recursos computacionais ou minimizar os custos monetários da execução. Em sua forma geral, o problema de escalonamento de tarefas/*workflow* é NP-Completo e o problema de otimização associado é NP-Difícil. Isto é, não conhecemos nenhum algoritmo que gere soluções determinísticas ótimas em tempo polinomial. Portanto, algoritmos ótimos para realizar escalonamento de grandes instâncias de um problema levam muito tempo para serem executados, tornando, assim, as heurísticas e outras abordagens sub-ótimas uma boa opção para o problema de escalonamento.

2.2. Cenário

O provedor de SaaS pode alugar instâncias de máquinas virtuais do provedor de infraestrutura como serviço (*Infrastructure as a Service* – IaaS) por meio de dois tipos de acordo de nível de serviço (SLA – *Service Level Agreements*): sob-demanda ou reservado. No SLA sob-demanda, o provedor de SaaS alugará máquinas virtuais sem compromissos de longo prazo e irá pagar somente pelas unidades de tempo (normalmente por hora) utilizadas. Em contrapartida, no SLA reservado, o provedor de SaaS irá alugar os recursos virtualizados com compromissos de longo prazo (1 a 3 anos, por exemplo) e pagará uma taxa antecipada para reservar cada máquina virtual. Entretanto, o provedor de SaaS receberá um desconto significativo sobre a tarifa das unidades de tempo utilizadas de cada

instância reservada. O conceito de SLA adotado neste trabalho será semelhante ao conceito usado na Amazon EC2[1].

Com intuito de minimizar investimentos em recursos computacionais, o usuário pode executar seus DAGs através do serviço de execução de *workflow* disponibilizado pelo provedor de SaaS, como mostra a Figura 1. Ademais, o usuário pode desejar que a execução de seu *workflow* termine antes de um determinado tempo (*deadline*). Para alcançar isso, consideramos nesse cenário duas camadas de SLA. A primeira camada inclui todos os SLAs estabelecidos entre o provedor de SaaS e cada um dos seus clientes, enquanto a segunda camanda contém todos os SLAs acordados entre o provedor de SaaS e cada provedor de IaaS cujos serviços são contratados. Portanto, o provedor de SaaS tem como objetivo executar o *workflow* do cliente dentro do *deadline* (estipulado no SLA da primeira camada) utilizando recursos alugados a partir de vários provedores de IaaS.

Por outro lado, o provedor de SaaS necessita minimizar o custo monetário dessa execução, para ser capaz de maximizar seu lucro. Quando sua infraestrutura está inteiramente ocupada, o provedor de SaaS deve alugar recursos virtualizados a fim de não recusar nenhuma solicitação de seus clientes. Portanto, o escalonamento de *workflow* em nuvem consiste basicamente em um processo de decisão cujo principal objetivo é determinar os seguintes aspectos: (i) a quantidade e o tipo de máquinas virtuais que serão necessárias para serem alugadas, (ii) qual provedor de IaaS deve ser utilizado para minimizar os custos totais monetários para executar o *workflow* e (iii) quais recursos podem satisfazer o *deadline* estipulado pelo usuário.

3. Trabalhos Relacionados

Alguns trabalhos propõem soluções para a execução de tarefas independentes na computação em nuvem, mas apenas poucos trabalhos consideram a relação custo-benefício do uso de recursos virtualizados, que são alugados a partir de várias nuvens públicas, para oferecer serviços de execução de *workflows* focados em parâmetros de qualidade de serviço (QoS – *Quality of Service*) definidos nos SLAs.

O trabalho em [Wu et al. 2010] descreve um algoritmo de alocação de recursos para os provedores de SaaS com o propósito de minimizar o custo da infraestrutura alugada e da violações de SLAs. Os autores apresentam políticas de custo-benefício para o mapeamento e escalonamento de tarefas, com a intenção de maximizar o lucro do provedor de SaaS através do uso de vários provedores de IaaS. Em [Van den Bossche et al. 2010] os autores propõem um escalonador de tarefas que é formulado por meio da PLI. São apresentadas possíveis formas de avaliar o custo computacional com intuito de maximizar o uso do *datacenter* privado e de minimizar o custo da execução de tarefas nas nuvens públicas, enquanto satisfaz as restrições de QoS das aplicações. Embora esses trabalhos apresentem avanços no campo de escalonamento de tarefas nas nuvens, nenhum deles considera execução de *workflows*.

Em [Yao et al. 2010], os autores propõem um algoritmo de escalonamento de *workflow* em grades computacionais que minimiza o custo da execução, enquanto satisfaz o *deadline* estipulado pelo usuário. O algoritmo é formulado através da PLI e realiza o escalonamento em duas etapas: (i) decompõe a restrição do tempo de execução

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

do *workflow* (*deadline global*) em janelas de tempo para todas as tarefas (isto é, determina um *deadline local* para cada tarefa) e (ii) seleciona o melhor serviço que satisfaça a restrição da janela de tempo local. Embora esse escalonador minimize o custo monetário da execução *workflow* respeitando um *deadline*, os autores consideram apenas *workflows* sequenciais e afirmam que todo *workflow* pode ser reduzido a um *workflow* sequencial equivalente, por meio do agrupamentos dos nós paralelos em apenas um nó. O uso do *workflow* sequencial facilita o escalonamento, mas diminui a qualidade da solução.

Um estudo de caso sobre o problema de escalonamento é apresentado em [Stefansson et al. 2011]. Os autores apresentam uma comparação entre dois modelos matemáticos para esse problema, sendo um formulado com o uso do tempo discreto e outro com tempo contínuo. Em [Floudas and Lin 2005], os autores também comparam o uso do tempo discreto e contínuo na formulação do problema de escalonamento como uma programação linear inteira. Embora o tempo contínuo solucione casos de testes maiores e apresente uma melhor precisão nos escalonamentos quando comparado ao uso do tempo discreto, nenhum desses trabalhos considera ambientes da computação em nuvem.

4. Visão Geral do Algoritmo de Escalonamento

Nesta seção vamos descrever a formulação do escalonador baseada em [Genez et al. 2012] cuja função objetivo é minimizar os custos monetários da execução do DAG, enquanto satisfaz o *deadline* estipulado no SLA do usuário. Esse escalonador leva em consideração o custo de computação de cada nó do DAG, o custo de comunicação das arestas do DAG, o poder de processamento dos recursos, os preços das máquinas virtuais e a largura de banda dos enlaces que conectam esses recursos. Ao considerar essas variáveis, o escalonador é capaz de decidir qual é o melhor recurso para executar cada nó do DAG. Portanto, a PLI decide entre (i) alugar máquinas virtuais caras e poderosas para acelerar a execução; ou (ii) alugar máquinas virtuais baratas para não desperdiçar recursos (ou dinheiro) e, ao mesmo tempo, cumprir os *deadlines* estipulados.

4.1. Notações e Modelagem do Problema de Escalonamento

O algoritmo de escalonamento é formulado através da PLI e utiliza as seguintes notações:

- n: número de nós do DAG ($n \in \mathbb{N}$);
- $\mathcal{W} = \{w_1, \ldots, w_n\}$: conjunto de demandas de processamento de cada nó $n_i \in \mathcal{U}$, expressa como o número de instruções a serem processadas ($w_k \in \mathbb{R}^+$);
- $f_{i,j}$: quantidade de unidades de dados transmitidos entre n_i e n_j ($f_{i,j} \in \mathbb{R}^+$);
- $\mathcal{H}(j)$ = $\{ij : i < j$, existe um arco do vértice i para o vértice j no DAG$\}$, é o conjunto de predecessores imediatos de n_j;
- $\mathcal{D}_\mathcal{G}$: tempo de término (*deadline*) da execução do DAG $\mathcal{G}$ desejado pelo cliente do provedor de SaaS;
- $\mathcal{I} = \{i_1, \ldots, i_m\}$: conjunto de provedores de IaaS que compõem a infraestrutura;
- δ_i: número máximo de máquinas virtuais que um cliente pode alugar do provedor de IaaS $i \in \mathcal{I}$ em um determinado tempo t ($\delta_i \in \mathbb{N}$).

Seja $\mathcal{S}_i = \sigma_i^r \cup \sigma_i^o$ o conjunto composto por todos os SLAs que foram assinados entre o provedor de SaaS e o provedor de IaaS $i \in \mathcal{I}$. Esse conjunto é formado por dois subconjuntos disjuntos: (i) σ_i^r que contem os SLAs destinados a máquinas virtuais reservadas e que estão prontamente disponíveis e (ii) σ_i^o que inclui apenas os SLAs para

máquinas virtuais que são alugadas sob demanda (*on-the-fly*). Sejam $\mathcal{V}_i^r$ e $\mathcal{V}_i^o$ os conjuntos disjuntos que contêm, respectivamente, máquinas virtuais associadas com os preços para os recursos reservados e sob-demanda do provedor de IaaS $i \in \mathcal{I}$. Assim, $\mathcal{V}_i = \mathcal{V}_i^r \cup \mathcal{V}_i^o$ representa o conjunto de máquinas virtuais disponíveis para serem alugadas do IaaS i.

Cada SLA $s_r \in \sigma_i^r$ está relacionado apenas com uma máquina virtual $v \in \mathcal{V}_i^r$, e cada SLA $s_o \in \sigma_i^o$ está associado somente com uma máquina virtual $v \in \mathcal{V}_i^o$. Um dos parâmetros definidos em cada contrato $s \in \mathcal{S}_i$ é o número $\alpha_s \in \mathbb{N}^+$ de máquinas virtuais disponibilizadas para o provedor de SaaS. Outro parâmetro definido em $s \in \mathcal{S}_i$ é o tempo de duração $t_s \in \mathbb{N}^+$ desse acordo, que é normalmente estabelecido a longo prazo para os contratos $s_r \in \sigma_i^r$. Aliás, o fornecimento de máquinas virtuais é limitado para cada cliente do provedor de IaaS [Wu et al. 2010] e, dessa forma, a restrição $\sum_{s \in \mathcal{S}_i} \alpha_s \leq \delta_i, \ \forall \ i \in \mathcal{I}$ deve ser rigorosamente obedecida para um determinado tempo t.

A seguir definimos as características dos provedores de IaaS para a PLI.

- m: número de provedores de IaaS ($m \in \mathbb{N}$);
- $\mathcal{P}$: número de núcleos de processamento da máquina virtual $v \in \mathcal{V}$, onde $\mathcal{P} \in \mathbb{N}^+$;
- $\mathcal{J}_v$: unidades de tempo que a máquina virtual $v \in \mathcal{V}$ leva para executar uma instrução, com $\mathcal{J}_v \in \mathbb{R}^+$;
- $\mathcal{L}_{v_g,v_h}$: unidades de tempo necessários para transmitir uma unidade de dados sobre o enlace que conecta a máquina virtual $v_g \in \mathcal{V}$ e a máquina virtual $v_h \in \mathcal{V}$, com $\mathcal{L}_{v_g,v_g} = 0$ se $v_g = v_h$ e $\mathcal{L}_{v_g,v_h} \in \mathbb{R}^+$;
- $\mathcal{B}_{i,v}$: variável binária que assume um valor 1 se a máquina virtual v pertence ao provedor de IaaS $i \in \mathcal{I}$, ou 0 caso contrário;
- $\mathcal{C}_v$: preço para alugar a máquina virtual $v \in \mathcal{V}$ durante uma unidade de tempo, com $\mathcal{C}_v \in \mathbb{R}^+$.

Seja $\zeta = \{\sigma_1^r, \ldots, \sigma_m^r\}$ o conjunto composto por todos os SLAs associados a reservas de máquinas virtuais. Então, definimos a variável binária $\mathcal{K}_{s,v}$ que assume o valor de 1 se o SLA $s \in \zeta$ está relacionado com a máquina virtual reservada $v \in \mathcal{V}$, ou 0 caso contrário. Isso significa que, se $\mathcal{K}_{s,v} = 1$ e $\mathcal{B}_{i,v} = 1$, então $s \in \sigma_i^r$ e $v \in \mathcal{V}_i^r$.

4.2. Formulação da Programação Linear Inteira

A formulação da PLI leva em consideração a quantidade de núcleos de processamento da máquina virtual para tomar decisões no escalonamento, e também o número limitante (δ_i) de máquinas virtuais que podem ser instanciadas para cada usuário em um determinado tempo t. Dessa maneira, o problema da programação inteira fornece o escalonamento através das variáveis binárias x e y e da constante $\mathcal{C}_v$:

- $x_{u,t,v}$: variável binária que assume o valor 1 se o nó u termina sua execução no instante de tempo t na máquina virtual v, caso contrário, esta variável assume o valor 0;
- $y_{t,v}$: variável binária que assume o valor 1 se a máquina virtual v está sendo utilizada no instante de tempo t, caso contrário, esta variável assume o valor 0;
- $\mathcal{C}_v$: constante que assume o custo por unidade de tempo da máquina virtual v.

A formulação da PLI considera intervalos discretos de tempo. Por isso, a linha temporal discreta da execução do *workflow* é definida por $\mathcal{T} = \{1, \ldots, \mathcal{D}_\mathcal{G}\}$, que varia de 1 ao *deadline* $\mathcal{D}_\mathcal{G}$ desejado. Portanto, a PLI é formulada da seguinte forma.

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

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

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

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

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

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

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

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

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

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

As restrições em (C1) especificam que todo serviço deve ser executado em algum momento e em uma única máquina virtual. A restrição em (C2) determina que um nó u do DAG não pode terminar sua execução até que todas suas instruções tenham sido executadas na máquina virtual v. As restrições em (C3) estabelecem que um nó z do DAG não pode iniciar a sua execução até que todas os nós que o precedem tenham terminado seus processamentos e os dados resultantes tenham chegado à máquina virtual que executará z. As restrições em (C4) estipulam que o número de nós do DAG em execução em uma máquina virtual v, em um determinado tempo t, não pode exceder o número de núcleos de processamento de v.

As restrições em (C5) determinam que uma máquina virtual deve permanecer ativa (ou seja, com status *sendo usado* habilitado na variável y), enquanto ela estiver executando os nós que a exigem. As restrições em (C6) especificam que o número de máquinas virtuais reservadas somado com o número de máquinas virtuais alugadas sob demanda não pode exceder o número máximo permitido em cada provedor de IaaS. As restrições em (C7) estabelecem que a quantidade de máquinas virtuais sendo utilizadas não pode exceder o limite estipulado no SLA. As duas últimas restrições, (C8) e (C9), especificam que as variáveis deste programa linear inteiro (x e y) só irão assumir os valores binários.

5. Representação da Linha do Tempo

O problema de escalonamento é conhecido por ser um problema difícil de ser modelado e resolvido eficientemente [Stefansson et al. 2011]. A granularidade da linha do tempo é a questão-chave desse problema, quando utilizamos programação linear inteira. O tempo pode ser classificado em duas principais categorias [Floudas and Lin 2005]: discreto e contínuo. No primeiro modo, os escalonadores são formulados por meio da abordagem da discretização do tempo; isto é, a linha do tempo é divida em intervalos de tempo iguais. Dessa forma, as decisões do escalonador, tal como término de um evento, devem ser feitas somente no início (ou no final) desses intervalos. No modo contínuo, por outro

lado, as decisões do escalonador são realizadas em qualquer instante de tempo pertencente ao intervalo $[1, deadline]$. Embora a linha temporal contínua aumente a precisão do escalonador, ou seja, a linha do tempo apresenta uma granularidade mais fina, o modelo matemático torna-se mais complexo [Yee and Shah 1998], assim como torna-se mais complexa a obtenção de soluções ótimas.

Por outro lado, a natureza do problema de escalonamento requer, de modo geral, o uso de algumas variáveis discretas para representar decisões discretas, como, por exemplo, atribuições de recursos e alocações de tarefas ao longo do tempo [Stefansson et al. 2011]. Além disso, os provedores de IaaS normalmente contabilizam o uso das máquinas virtuais de acordo com as unidades de tempo inteiras utilizadas (modelo *pay-per-use*). Aliás, as unidades de tempo parcialmente consumidas são cobradas como se fossem unidades de tempo completas[2]. Portanto, devido à cobrança das máquinas virtuais serem feitas por unidades de tempo inteiras, o escalonador apresentado em [Genez et al. 2012] utiliza apenas tempo discreto para representar a execução do *workflow*. Entretanto, dependendo do nível de granularização da linha do tempo e do tamanho do *workflow*, o tempo de execução do escalonador pode ser alto para o contexto da computação em nuvem; resultando, então, em nenhuma solução viável em tempo hábil. Assim, neste trabalho vamos avaliar o uso de diferentes níveis de fragmentação do tempo discretizado no escalonamento de diferentes tipos de DAGs.

O uso de intervalos de tempo curtos (granularidade fina) e/ou linha temporal longa (*deadline* alto) pode aumentar consideravelmente a quantidade de intervalos de tempo discretizado no conjunto $\mathcal{T}$ usado no programa linear inteiro. Por exemplo, o aumento do *deadline* (e/ou do grafo) implica diretamente no aumento do tamanho do conjunto $\mathcal{T}$. Essas situações podem aumentar o espaço de busca das soluções e, consequentemente, aumentar o tempo de execução do escalonador. Em outras palavras, o escalonamento torna-se um problema computacionalmente caro de ser resolvido ou até mesmo em um problema sem solução em tempo hábil [Floudas and Lin 2005]. Portanto, definimos λ como o fator multiplicativo que determina a granularidade do tempo discreto no problema de escalonamento. Por esse motivo redefinimos o conjunto $\mathcal{T}$ da PLI, descrito na Seção 4, no seguinte conjunto: $\mathcal{T} = \{\lambda, 2\lambda, 3\lambda, 4\lambda, \ldots, \Lambda\}$, tal que $\Lambda \leq \mathcal{D}_{\mathcal{G}}$ e $\lambda \in \mathbb{N}^{+}$.

Por outro lado, o uso de intervalos de tempo longos (granularidade grossa) pode diminuir a quantidade de intervalos de tempo, de modo que o escalonamento torna-se inviável, pois não terá unidades de tempo o suficiente para representar todas as dependências dos nós do DAG. Além disso, outra consequência do uso da granularidade grossa são soluções encontradas com custos mais altos, pois nesse caso alguns núcleos de processamento não serão utilizados e isso poderá resultar em desperdício de recursos (ou dinheiro). Assim, há um claro *trade-off* entre utilizar intervalos de tempo curtos para obter um escalonamento mais preciso ou usar intervalos de tempo longos para diminuir o tempo de execução do escalonamento. Portanto, em nossos experimentos, iremos variar o fator multiplicativo λ para diminuir o tempo de execução no escalonamento de *workflows* com uma grande quantidade de nós e dependências, de modo que o escalonamento seja viável e, ainda, manter os custos monetários baixos.

[2]http://aws.amazon.com/ec2/pricing/

6. Avaliação

Implementamos o programa linear inteiro em JAVA e as simulações foram realizadas por meio da biblioteca IBM ILOG CPLEX Optimizer[3] com as configurações internas padrão. As métricas avaliadas foram o custo monetário do escalonamento, o *makespan* do *workflow*, o tempo de execução do escalonador e o número de vezes que nenhuma solução foi encontrada (número de soluções inviáveis). Nesta seção vamos apresentar o cenário dos experimentos, o ambiente das simulações e os resultados experimentais.

6.1. Configuração das Simulações

Três provedores de IaaS foram utilizados em nossas simulações, sendo que cada provedor define seus próprios preços das máquinas virtuais para os planos de reservas e sob-demanda. A Tabela 1 mostra as opções de máquinas virtuais para o provedor de IaaS A, a Tabela 2 para o provedor de IaaS B e a Tabela 3 para o provedor de IaaS C. Além disso, a Tabela 4 mostra todos os SLAs de máquinas virtuais reservadas pelo provedor de SaaS. O número máximo de máquinas virtuais que podem ser alugadas a partir de cada provedor de IaaS foi estipulado da seguinte maneira: $\delta_A = 4$, $\delta_B = 7$, $\delta_C = 2$.

Tabela 1. Provedor de IaaS A

Tipo	Núcleo	Instrução Por Núcleo	Preço Demanda	Preço Reserva
P	1	1.5	\$0.13	\$0.045
M	2	1.5	\$0.20	\$0.070

Tabela 2. Provedor de IaaS B

Tipo	Núcleo	Instrução Por Núcleo	Preço Demanda	Preço Reserva
P	1	2	\$0.17	\$0.045
M	2	2	\$0.30	\$0.059
G	3	2	\$0.40	\$0.140
EG	4	2	\$0.52	\$0.183
EG2	8	2	\$0.90	\$0.316

Tabela 3. Provedor de IaaS C

Tipo	Núcleo	Instrução Por Núcleo	Preço Demanda	Preço Reserva
P	1	2	\$0.15	\$0.052
M	2	2	\$0.25	\$0.088
G	4	2.5	\$0.50	\$0.176
EG	8	2.5	\$0.80	\$0.281

Tabela 4. Máquinas virtuais reservadas para provedor de SaaS

Tipo	IaaS	VM	Quantidade
Reservada	A	P	1
Reservada	A	M	1
Reservada	B	P	1
Reservada	B	M	1

Nuvens públicas geralmente não fornecem informações sobre a qualidade de serviço dos enlaces internos (entre as máquinas virtuais) e enlaces externos (entre os provedores de IaaS). Portanto, assumimos que a largura de banda dos enlaces internos (dentro do mesmo provedor de IaaS) é maior do que a largura de banda dos enlaces externos, pois é uma suposição razoável em ambientes reais. Isso se reflete em nossa simulação gerando aleatoriamente um valor para $\mathcal{L}$ no intervalo $[2, 3]$ para os enlaces entre nuvens diferentes, enquanto que para os enlaces entre máquinas virtuais dentro da mesma nuvem, o valor de $\mathcal{L}$ pertence ao intervalo $[0.1, 0.2]$.

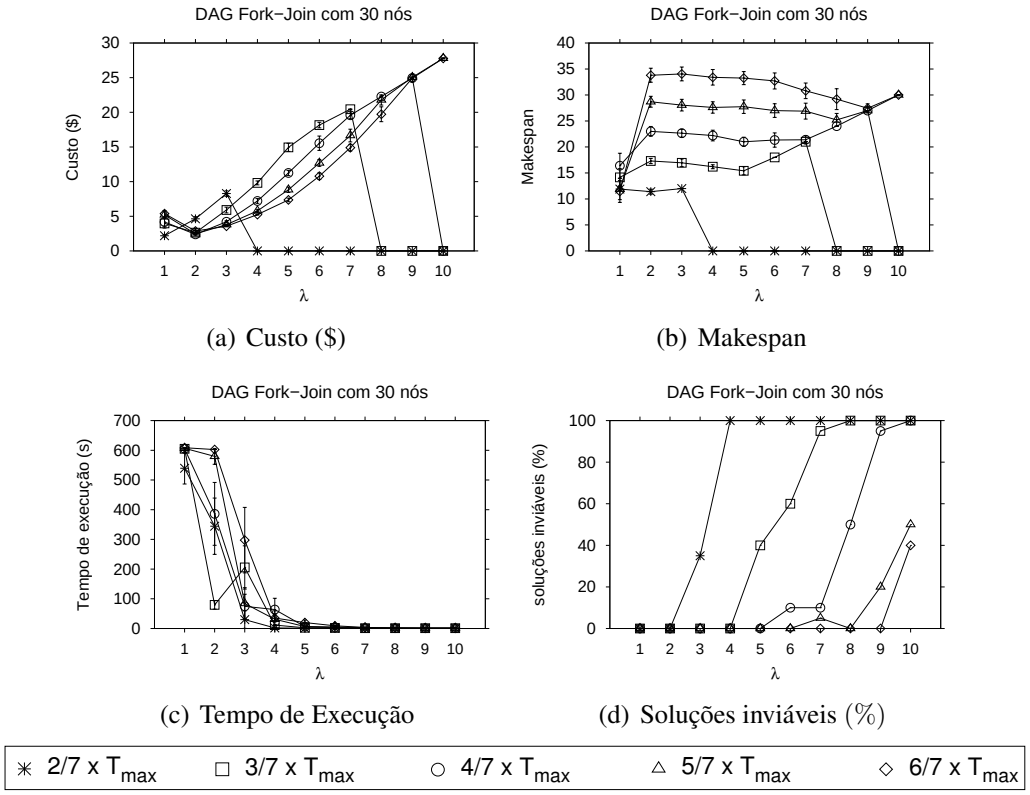
Nossa avaliação é composta por simulações de DAGs que representam aplicações do mundo real, tais como *Montage* [Deelman et al. 2005], *Ligo* [Ramakrishnan et al. 2007] e o DAG *fork-join* da Figura 1 com 30 nós. Simulações foram executadas em um processador Intel® Xeon $X5660$ CPU $2.80GHz$ com $16GB$ de RAM.

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

6.2. Resultados Experimentais

Realizamos 30 simulações para cada DAG com as configurações dos provedores de IaaS e dos SLAs presentes nas Tabelas 1 a 4. Em cada simulação, o custo de computação de cada nó e os custos de comunicação de cada dependência foram obtidos aleatoriamente do intervalo $[1, 3]$. Além disso, em cada simulação variamos o fator multiplicativo λ para avaliar a relação entre as métricas custo monetário e tempo de execução do escalonador. Executamos o PLI para cada simulação com tempo limite de 10 minutos.

Para representar os possíveis valores dos *deadlines* solicitados pelos usuários, executamos as simulações com $\mathcal{D}_\mathcal{G}$ variando de $\mathcal{T}_{max} \times 2/7$ à $\mathcal{T}_{max} \times 6/7$ em etapas de $1/7$, onde $\mathcal{T}_{max}$ é o *makespan* da execução sequencial de todos os nós do DAG em um único recurso cujo preço é o mais barato. *Deadlines* iguais à $\mathcal{T}_{max} \times 1/7$ apresentaram somente soluções inviáveis, enquanto *deadlines* iguais à $\mathcal{T}_{max} \times 7/7$ podem ser trivialmente alcançados colocando todas as tarefas no recurso mais barato. É importante frisar que o divisor 7 foi escolhido apenas para avaliar a evolução da discretização com o aumento do *deadline*, portanto outros divisores poderiam ser utilizados. Os gráficos foram plotados de acordo com a média das 30 simulações de cada DAG, com um intervalo de confiança de 95%. Em alguns pontos, o intervalo de confiança é pequeno e, consequentemente, a sua visualização torna-se imperceptível.



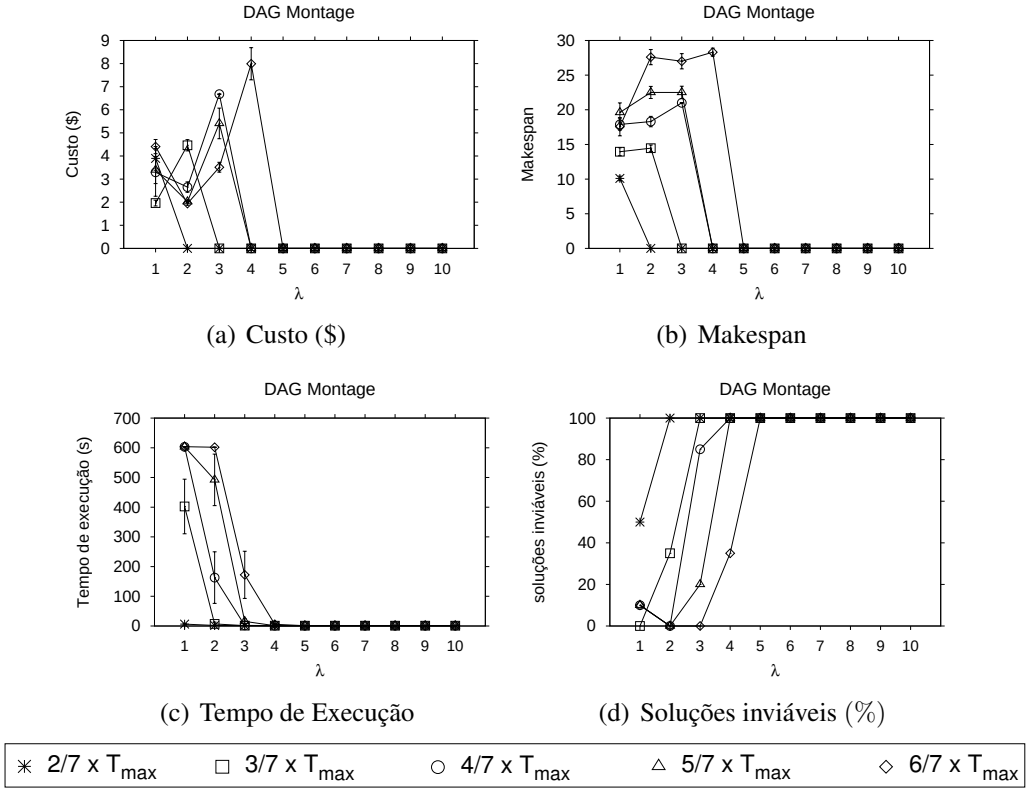
(a) Custo ($) (b) Makespan

(c) Tempo de Execução (d) Soluções inviáveis $(\%)$

Figura 2. Resultados para o DAG *fork-join* com 30 nós

A Figura 2 mostra resultados das simulações para o DAG *fork-join* com 30 nós. Para $\lambda = 1$ e $\lambda = 2$, o escalonador encontrou soluções para todos os $\mathcal{D}_\mathcal{G}$, ou seja, 0% de soluções inviáveis. Para $\lambda = 2$ e $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$, o escalonamento foi 1.57 vezes mais rápido, mas gerou um aumento de duas vezes no custo monetário, quando comparado às

soluções para o mesmo $\mathcal{D}_\mathcal{G}$ e $\lambda = 1$. No entanto, mantendo $\lambda = 2$ e aumentando o *deadline*, o escalonamento tornou-se mais rápido com custos mais baixos. Por exemplo, para $\lambda = 2$ e $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 3/7$ foi possível encontrar soluções com 50% do custo e 6 vezes mais rápido quando comparado às soluções para o mesmo $\mathcal{D}_\mathcal{G}$ e $\lambda = 1$. Embora haja um aumento nos custos para $\lambda \geq 3$ (devido ao desperdício de núcleos virtualizados alugados não usados), o tempo de execução do escalonamento diminuiu consideravelmente. Por exemplo, na comparação entre as simulações com $\lambda = 1$ e $\lambda = 4$, para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 4/7$, temos uma redução de aproximadamente 91% no tempo de execução e, em contrapartida, um aumento de aproximadamente 53% no custo monetário. Por outro lado, o aumento de λ, acompanhado de redução no $\mathcal{D}_\mathcal{G}$, diminui a linha temporal do escalonamento devido ao aumento da granularidade da discretização do tempo e, portanto, aumenta o número de soluções inviáveis. Isso ocorre pelo motivo de não existirem intervalos de tempo suficientes para todos os nós do DAG. Por exemplo, o aumento de soluções inviáveis para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$ ocorre apenas quando $\lambda = 3$, porém para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$ esse aumento acontece somente quando $\lambda = 9$.



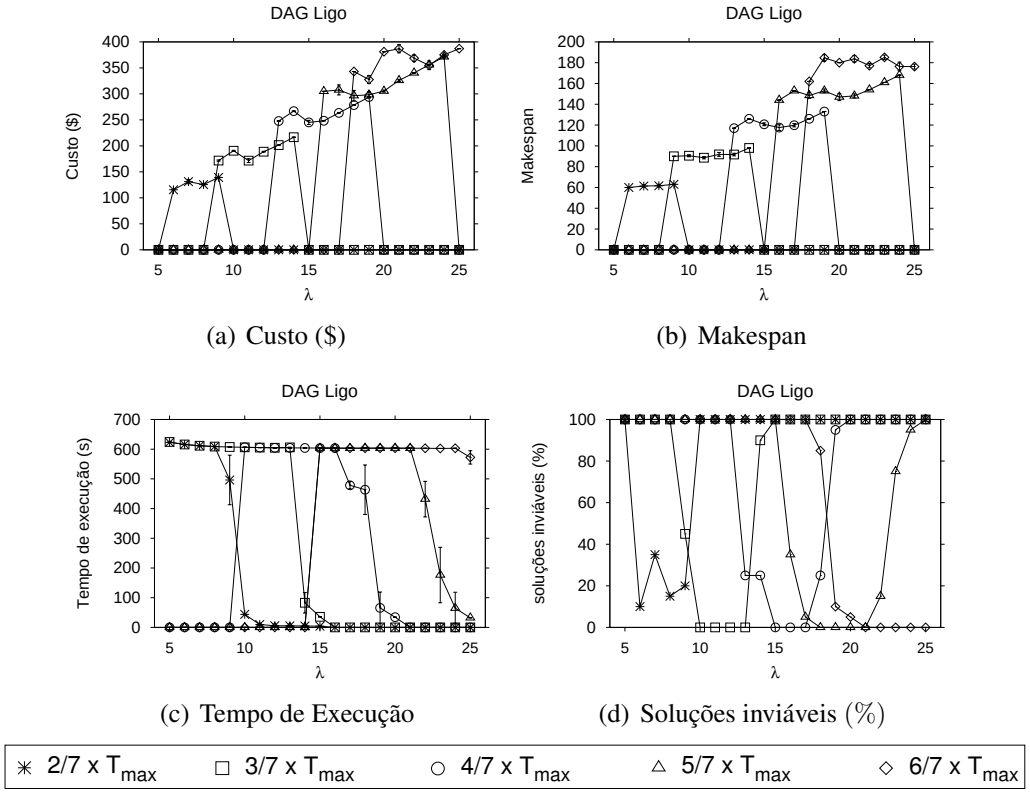
(a) Custo ($) (b) Makespan

(c) Tempo de Execução (d) Soluções inviáveis $(\%)$

Figura 3. Resultados para o DAG *Montage* (25 nós)

Os resultados das simulações para o DAG *Montage* são apresentados na Figura 3. O escalonador encontrou a solução ótima (custo mínimo) para para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 3/7$ e $\lambda = 1$, pois obteve a solução antes de atingir o limite de tempo de 10 minutos. No entanto, para as simulações com $\mathcal{D}_\mathcal{G} > \mathcal{T}_{max} \times 3/7$, o solucionador da PLI foi abortado por esse tempo limitante de 10 minutos e, consequentemente, teve que escolher a melhor solução obtida até esse momento. Aumentando a discretização do tempo para $\lambda = 2$, foi possível encontrar soluções rapidamente e, além disso, com custos menores. Por exemplo, para as simulações com $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 4/7$, o custo médio do escalonamento foi reduzido em

15% enquanto o tempo médio de execução foi reduzido em 70%, quando comparado às soluções para o mesmo $\mathcal{D}_\mathcal{G}$ e $\lambda = 1$. Mantendo $\lambda = 2$ e aumentando o *deadline*, a redução do custo ficou ainda maior; 43% para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 5/7$ e 45% para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$. Por outro lado, o tempo de execução reduziu 18% somente para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 5/7$, enquanto para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$, essa redução ocorreu apenas quando $\lambda = 3$ que, por sinal, foi de 67%. Além disso, notamos que o número de soluções inviáveis manteve-se baixo para $\lambda = 1$, $\lambda = 2$ e *deadline* altos, como mostra a Figura 3(d). Portanto, o aumento da granularidade da discretização do tempo permite ao solucionador da PLI encontrar soluções viáveis antes de atingir o limite de tempo estipulado nas simulações.



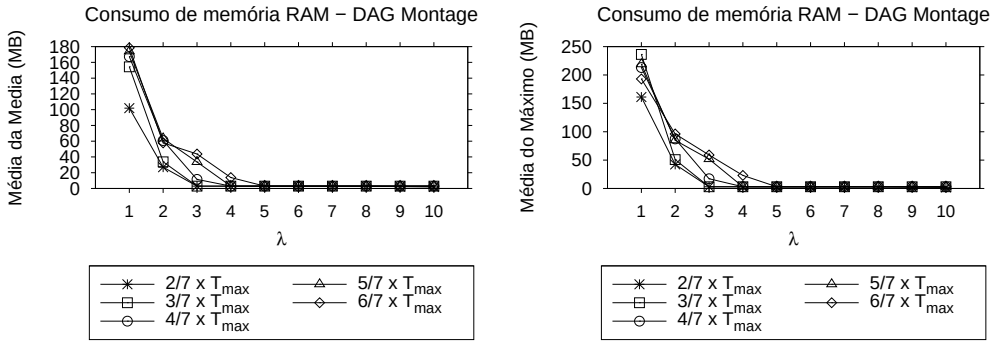
(a) Custo ($) (b) Makespan

(c) Tempo de Execução (d) Soluções inviáveis $(\%)$

Figura 4. Resultados para o DAG *Ligo* (168 nós)

Quando usamos DAGs com maior número de nós, o escalonamento possui uma quantidade menor de soluções inviáveis com aumento da discretização do tempo, como mostram os resultados das simulações do DAG *Ligo* na Figura 4. Assim, utilizando uma granularidade fina ($\lambda < 5$), o solucionador da PLI não consegue achar uma solução em tempo hábil, como mostra a Figura 4(d). Em outras palavras, através da PLI, as soluções viáveis para esses tipos de DAGs são possíveis apenas com aumento da granularidade do tempo. Por exemplo, para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 2/7$, o escalonamento foi possível apenas para $\lambda \in \{6, 9\}$. Entretanto, como o DAG *Ligo* é maior (168 nós), o aumento do multiplicador de $\mathcal{T}_{max}$ acarreta um aumento maior no conjunto $\mathcal{T}$ do que no caso de DAGs menores, e, portanto, o escalonamento torna-se computacionalmente caro de ser solucionado mais rapidamente. Assim, para diminuir o tamanho do conjunto $\mathcal{T}$, temos que aumentar o valor de λ. Por exemplo, para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 3/7$, foi possível encontrar soluções viávies somente quando $\lambda \in \{9, 4\}$, e para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$, apenas quando $\lambda \in \{18, 25\}$.

Portanto, para um determinado *deadline*, podemos escolher um λ que irá gerar um custo mínimo ao escalonamento, como, por exemplo, $\lambda = 19$ para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$.



(a) Consumo médio (b) Consumo máximo

Figura 5. Consumo de memória RAM nas simulações do DAG *Montage*

A Figura 5 apresenta o consumo médio (Figura 5(a)) e máximo (Figura 5(b)) de memória RAM das simulações do DAG *Montage*. Essas figuras mostram que quanto maior o valor de λ, menor será o consumo de memória RAM devido à diminuição da complexidade computacional do problema, que tem como consequência um menor tempo de execução do escalonador, como apresenta a Figura 3(d). Por exemplo, na comparação entre $\lambda = 1$ e $\lambda = 3$ para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$, o consumo de memória RAM e o tempo de execução do escalonador foram reduzidos aproximadamente em 73% e 67%, respectivamente, e com 100% das soluções viáveis . O consumo de memória RAM das simulações dos DAGs *Ligo* e *fork-join* (com 30 nós) apresentam comportamentos semelhantes.

Os resultados apresentados nesta seção sugerem que o aumento da granularidade da discretização do tempo pode ser eficaz na redução do tempo de execução de *workflows* com vários nós e várias dependências. Quando executados com $\lambda = 1$ e *deadlines* altos, esses *workflows* aumentam não só o número de intervalos discretos em $\mathcal{T} = \{1, \ldots, \mathcal{D}_\mathcal{G}\}$, como também aumentam a complexidade computacional do problema, dificultando o solucionador a encontrar uma boa solução inteira em tempo viável. Portanto, a abordagem da discretização do tempo apresentado neste trabalho pode fornecer base para o provedor de SaaS negociar seus contratos SLAs com seus clientes.

7. Conclusão

Neste trabalho apresentamos uma abordagem para reduzir o tempo de execução de *workflows* (com vários nós e várias dependências) através do uso de diferentes níveis de discretização da linha do tempo. Realizamos simulações com vários DAGs que representam aplicações do mundo real, tais como: *Montage*, *Ligo* e DAG *fork-join*. Os resultados mostram que, para DAGs grandes e *deadlines* altos, a estratégia de aumentar a granularidade da discretização do tempo permite ao escalonador encontrar rapidamente soluções viáveis. Como trabalhos futuros podemos apontar o relaxamento da PLI e o desenvolvimento de heurísticas não iterativas para encontrar soluções viáveis inteiras sobre as execuções relaxadas. Além disso, escalonamento de múltiplos DAGs e DAGs dinâmicos em nosso cenário também são considerados como trabalhos futuros.

Agradecimentos

Os autores agradecem à CAPES, à FAPESP (2010/14433-8 e 2009/15008-1) e ao CNPq pelo apoio financeiro, e à IBM por providenciar a ferramenta IBM ILOG CPLEX.

Referências

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

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

Floudas, C. and Lin, X. (2005). Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139:131–162.

Genez, T. A. L., Bittencourt, L. F., and Madeira, E. R. M. (2012). Workflow scheduling for SaaS / PaaS cloud providers considering two SLA levels. In *Network Operations and Management Symposyum (NOMS 2012) (accepted for publication)*.

Papuzzo, G. and Spezzano, G. (2011). Autonomic management of workflows on hybrid grid-cloud infrastructure. In *7th International Conference on Network and Service Management (CNSM 2011)*.

Ramakrishnan, A., Singh, G., Zhao, H., Deelman, E., Sakellariou, R., Vahi, K., Blackburn, K., Meyers, D., and Samidi, M. (2007). Scheduling data-intensiveworkflows onto storage-constrained distributed resources. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '07, pages 401–409, Washington, DC, USA. IEEE Computer Society.

Stefansson, H., Sigmarsdottir, S., Jensson, P., and Shah, N. (2011). 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*, 215(2):383 – 392.

Van den Bossche, R., Vanmechelen, K., and Broeckhove, J. (2010). Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 228 –235.

Wu, L., Garg, S. K., and Buyya, R. (2010). SLA-based admission control for a software-as-a-service providerin cloud computing environments. Technical Report CLOUDS-TR-2010-7, Uni. of Melbourne, Australia.

Yao, Y., Liu, J., and Ma, L. (2010). Efficient cost optimization for workflow scheduling on grids. In *International Conference on Management and Service Science (MASS 2010)*, pages 1 –4.

Yee, K. and Shah, N. (1998). Improving the efficiency of discrete time scheduling formulation. *Computers Chemical Engineering*, 22(0):S403 – S410.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, pages 7–18.