Escalonamento de Workflows com Uso Intensivo de Dados em Nuvens

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. *Cloud computing can be used to run e-Science applications, usually modeled as workflows. The execution of such applications can involve a response time (deadline) that must be fulfilled. However, to schedule data-intensive application workflows, the scheduler must consider task's hard-drive access. In this paper we evaluate the impact of scheduling I/O-intensive workflows using schedulers that are not prepared for this type of workflow, and propose a scheduler that considers the demands of I/O-operations. Results show that the proposed scheduler avoids the makespan increase, complying with deadlines.*

Resumo. *Computação em nuvem pode ser utilizada para executar aplicações de e-Ciência, geralmente modeladas como workflows. A execução de tais aplicações pode envolver um tempo de resposta (deadline) a ser cumprido. No entanto, para escalonar workflows com uso intensivo de dados, o escalonador deve considerar o tempo gasto com acesso ao disco. Neste artigo avaliamos o impacto do escalonamento de workflows com uso intensivo de dados em escalonadores não preparados para esse tipo de workflow, e propomos um escalonador que considera as demandas de I/O. Resultados mostram que o escalonador proposto evita o aumento do tempo de resposta, obedecendo os deadlines.*

1. Introdução

Várias aplicações complexas de e-Ciência podem ser modeladas como *workflows*, compondo um conjunto de serviços a serem processados em uma ordem bem definida. Para executar tais *workflows*, é desejável utilizar vários computadores com intuito de distribuir os serviços para execução em paralelo, a fim de reduzir o tempo de execução do *workflow* ou obedecer um tempo máximo estipulado. Recentemente, a nuvem vem sendo avaliada como alternativa de plataforma computacional para execução de *workflows*, pois os recursos podem ser alugados e liberados de acordo com a necessidade e tarifados por meio do modelo "pago-pelo-uso" (do inglês, *pay-as-you-go*) [Zhang et al. 2010].

O tempo de execução para *workflows* científicos com uso intensivo de dados pode ser composto por tempo de processamento (CPU) e tempo de leitura/escrita de informações. Esses *workflows* podem gerar grandes volumes de dados, os quais precisam ser analisados e disponibilizados para os cientistas a um ritmo cada vez mais acelerado. Assim, a execução de tais *workflows* pode envolver um tempo de resposta desejável (*deadline*). Devido ao tempo de processamento (CPU) ser geralmente maior que o tempo de acesso ao disco, quanto maior a demanda de operações de I/O do *workflow*, maior será o tempo que o recurso alocado ficará sendo utilizado e, consequentemente, maior o tempo

de execução do *workflow* (*makespan*). Adicionalmente, maior será o custo monetário do aluguel dos recursos em nuvens. Com intuito de cumprir o *deadline*, o escalonador precisa, no momento da escolha dos recursos, levar em consideração tanto o desempenho de processamento quanto o desempenho de acesso a disco.

Desenvolvemos um programa que usa programação linear inteira para escalonar *workflows* em nuvens [Genez et al. 2012b]. Nesse trabalho, o cliente envia seu *workflow* para ser executado por um provedor de *software-as-a-service* (SaaS), juntamente com um *deadline*. No entanto, esse escalonador não considera o tempo gasto com acesso ao disco nas máquinas virtuais alugadas e, com isso, *deadlines* podem não ser satisfeitos. Dessa forma, contratos de acordo de nível de serviços – *service level agreement*s (SLAs) – estabelecidos entre o provedor de SaaS e seus clientes podem acabar sendo quebrados, implicando em prejuízos ao provedor de SaaS.

A contribuição deste trabalho está na proposta e avaliação do escalonamento de *workflows* científicos com uso intensivo de dados, utilizando máquinas virtuais alugadas de múltiplos provedores de *infrastructure-as-a-service* (IaaS). Dessa forma, incluímos na modelagem do programa linear inteiro conceitos sobre o tempo gasto pelas operações de I/O na execução do *workflow*. O principal objetivo é, portanto, mostrar que utilizar um escalonamento ciente de I/O ajuda a (i) cumprir *deadlines* e (ii) diminuir o custo monetário do escalonamento.

O restante deste artigo está organizado da seguinte maneira. Trabalhos relacionados são expostos na Seção 2, enquanto conceitos básicos deste trabalho são descritos na Seção 3 . Uma visão geral do escalonador formulado como um programa linear inteiro é apresentado na Seção 4. Na Seção 5 descrevemos e analisamos os resultados experimentais, enquanto na Seção 6 finalizamos este artigo com as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Várias abordagens para escalonar *workflows* em nuvens são encontradas na literatura. O trabalho em [Wu et al. 2010] propõe um algoritmo de escalonamento de *workflows* em nuvens públicas que minimiza tanto o custo monetário quanto o tempo de execução (*makespan*) do *workflow*. O algoritmo é desenvolvido através da técnica de enxame de partículas discreto – *Revised Discrete Particle Swarm Optimization* (RDPSO). Uma estratégia para escalonar *workflows* de serviços em nuvens híbridas é proposto em [Bittencourt and Madeira 2011]. Os autores desenvolveram um algoritmo, denominado *Hybrid Cloud Optimized Cost* (HCOC), que decide quais recursos devem ser alugados da nuvem pública para aumentar o poder de processamento da nuvem privada. Em [Zhu et al. 2012], os autores descrevem uma heurística para escalonar *workflows* em nuvens que maximiza o uso dos recursos alugados, enquanto garante um tempo de resposta (*deadline*) para a execução do *workflow*. De modo geral, para realizar o escalonamento de *workflows*, esses trabalhos levam em consideração apenas os custos de processamento e de transmissão de dados das tarefas do *workflow*. Neste trabalho apresentamos uma avaliação com *workflows* de aplicações reais que também considera o tempo de acesso ao disco rígido (demanda de I/O) que cada tarefa executa no recurso que foi alocado.

Em [Juve et al. 2009, Juve et al. 2012], os autores analisaram o desempenho e

o custo monetário das execuções de *workflows* científicos na Amazon EC2[1], e compararam os resultados com as execuções desses *workflows* em um sistema *High-performance computing* (HPC) típico. Eles concluíram que os *workflows* científicos que resultaram em custos maiores na Amazon EC2 foram os que utilizavam mais acesso ao disco rígido, pois ficavam mais tempo utilizando as máquinas virtuais, devido às operações de I/O serem mais lentas que as operações de processamento (CPU). Embora esse trabalho apresente avaliações das execuções de *workflows* científicos nas *virtual machine*s (VMs) alugadas da nuvem, considerando apenas execução seqüencial dos *workflows*, ele não realiza escalonamento de *workflows* em uma única ou múltiplas nuvens.

3. Conceitos Básicos e Definições

Nesta seção apresentamos uma visão geral sobre a representação de *workflows* e o cenário de escalonamento de *workflows* em nuvens utilizado neste trabalho.

3.1. Representação de *Workflow*

Um *workflow* é geralmente representado por um grafo acíclico direcionado – *directed acyclic graph* (DAG) – $\mathcal{G} = \{\mathcal{U}, \mathcal{E}\}$, onde cada nó $u_i \in \mathcal{U}$ representa um serviço a ser executado e cada aresta (ou arco) $e_{i,j} \in \mathcal{E}$ representa uma dependência de dados entre o serviço i e j , onde u_j pode iniciar a sua execução somente após u_i finalizar e enviar todos os dados necessários a u_j. As palavras *workflow* e *DAG* serão utilizadas como sinônimo no restante deste trabalho.

A Figura 1 mostra o DAG *Montage*, um aplicativo que cria mosaicos de imagens a partir de dados coletados por telescópios. Nessa figura, os rótulos dos nós representam o nível do estágio de processamento das imagens, dos quais alguns podem ser paralelizados[2], como, por exemplo, os estágios 1, 2 e 5. *Montage* é uma aplicação de uso intensivo de dados armazenados em disco rígido, pois pode gastar mais de 95% do seu tempo de execução esperando as operações de I/O serem executadas [Deelman et al. 2008, Berriman et al. 2010, Juve et al. 2012]. Assim, para esse tipo *workflow*, o algoritmo de escalonamento deve também levar em consideração o tempo gasto com operações de I/O.

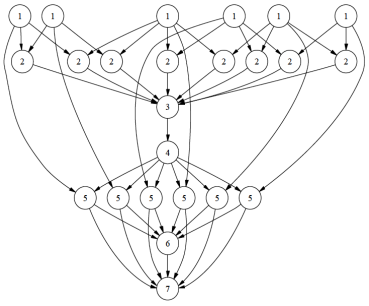


Figura 1. DAG que representa a aplicação *Montage* [Deelman et al. 2005]

3.2. Cenário do Escalonamento de Workflows em Nuvens

O provedor de SaaS pode alugar instâncias de máquinas virtuais do provedor de IaaS por meio de dois tipos de SLAs: sob-demanda ou reservado. No primeiro, o provedor de

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

[2]Maiores detalhes em `http://montage.ipac.caltech.edu`

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.

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. Aliás, 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 inclui todos os SLAs estabelecidos entre o provedor de SaaS e cada um dos seus clientes, enquanto a segunda contém todos os SLAs acordados entre o provedor de SaaS e cada provedor de IaaS cujos serviços são contratados. Logo, o provedor de SaaS tem como objetivo executar o *workflow* do cliente dentro do *deadline* utilizando recursos alugados a partir de vários provedores de IaaS.

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. Logo, o escalonamento de *workflow* em nuvem consiste basicamente em um processo de decisão cujo principal objetivo é determinar: (i) a quantidade e o tipo de VMs 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.

4. Visão Geral do Algoritmo de Escalonamento

Nesta seção vamos descrever a formulação do escalonador baseada em [Genez et al. 2012b] cuja função objetivo é minimizar os custos monetários da execução do DAG, enquanto satisfaz o *deadline*. Esse escalonador é capaz de providenciar escalonamentos com custos monetários baixos, mas considera somente DAGs cujos serviços são dominados pelo tempo de processamento em vez do tempo de acesso ao disco. Nossa proposta é incorporar no programa linear inteiro (PLI), o custo das operações de I/O de cada nó do DAG, a fim de providenciar soluções melhores e mais realistas ao provedor de SaaS.

4.1. Notação e Modelagem do Problema de Escalonamento

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

- $n = |\mathcal{U}|$: número de nós do DAG $\mathcal{G}$ ($n \in \mathbb{N}$);
- $\mathcal{W} = \{w_1, \ldots, w_n\}$: conjunto de demandas de processamento de cada nó $u_i \in \mathcal{U}$, expressa como o número de instruções a serem processadas ($w_i \in \mathbb{R}^+$);
- $\mathcal{O} = \{o_1, \ldots, o_n\}$: conjunto de demandas de operações de I/O de cada nó $u_i \in \mathcal{U}$, expressa como bloco de dados a serem lidas/gravadas no disco rígido ($o_i \in \mathbb{R}^+$);
- $f_{i,j}$: quantidade de unidades de dados a serem transmitidos entre os nós $u_i \in \mathcal{U}$ e $u_j \in \mathcal{U}$ ($f_{i,j} \in \mathbb{R}^+$);
- $\mathcal{H}(j) = \left\{ij : i < j, \text{existe um arco do vértice } i \text{ ao vértice } j \text{ no DAG } \mathcal{G}\right\}$: conjunto de predecessores imediatos de $u_j \in \mathcal{U}$;

- $\mathcal{D}_{\mathcal{G}}$: tempo de término (*deadline*) da execução do DAG $\mathcal{G}$ estipulado pelo cliente do provedor de SaaS;
- $\mathcal{I} = \{i_1, \ldots, i_m\}$: conjunto de provedores de IaaS que compõem a infraestrutura computacional do problema;
- δ_i: número máximo de máquinas virtuais que um cliente pode alugar do provedor de IaaS $i \in \mathcal{I}$ em qualquer instante de 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. Neste trabalho, vamos assumir que cada máquina virtual $v \in \mathcal{V}_i$ está associada com um elemento v_o de $\mathcal{V}_i^o$ e com um elemento v_r de $\in \mathcal{V}_i^r$. Dessa forma, os respectivos v_r e v_o de v possuem a mesma configuração de hardware, mas são tarifados com preços diferentes, de modo geral, o preço por unidade de tempo de v_o é maior ou igual ao de v_r.

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 acordo $s \in \mathcal{S}_i$ é o número $\alpha_s \in \mathbb{N}^+$ de máquinas virtuais solicitadas pelo provedor de SaaS. Outro parâmetro do SLA é o tempo de duração $t_s \in \mathbb{N}^+$ desse acordo. Portanto, os recursos alugados pelo provedor de SaaS estão presentes no conjunto $\mathcal{V}$, o qual é definido como: $\mathcal{V} = \left\{ \bigcup_{i=1}^{m} \mathcal{V}_i \quad \forall\, i \in \mathcal{I} : \mathcal{S}_i \neq \varnothing \right\}$ onde $m = |\,\mathcal{I}\,|$ e $\mathcal{V}_i = \mathcal{V}_i^r \cup \mathcal{V}_i^o$. Em outras palavras, de acordo com os SLAs assinados com cada provedor de IaaS $i \in \mathcal{I}$, ou seja, com os SLAs presentes em $\mathcal{S}_i$, o conjunto $\mathcal{V}$ é construído. A seguir, definimos algumas características importantes dos provedores de IaaS que também são intrínsecas à formulação do PLI:

- $m = |\,\mathcal{I}\,|$: número de provedores de IaaS, onde $m \in \mathbb{N}$;
- $\mathcal{P}_v$: número de núcleos de processamento da VM $v \in \mathcal{V}$, onde $\mathcal{P}_v \in \mathbb{N}^+$;
- $\mathcal{J}_v$: unidades de tempo que a máquina virtual $v \in \mathcal{V}$ leva para executar uma instrução, com $\mathcal{J}_v \in \mathbb{R}^+$;
- $\mathcal{F}_v$: unidades de tempo que a máquina virtual $v \in \mathcal{V}$ leva para ler/escrever um bloco de dados no disco rígido, com $\mathcal{F}_v \in \mathbb{R}^+$
- $\mathcal{L}_{v_g,v_h}$: unidades de tempo necessárias 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 $\mathcal{L}_{v_g,v_h} \in \mathbb{R}^+$. Se $v_g = v_h$, então $\mathcal{L}_{v_g,v_g} = 0$;
- $\mathcal{B}_{i,v}$: variável binária que assume valor 1 se a máquina virtual $v \in \mathcal{V}$ pertence ao provedor de IaaS $i \in \mathcal{I}$; caso contrário, assume o valor 0;
- $\mathcal{C}_v$: preço para alugar a máquina virtual $v \in \mathcal{V}$ durante uma unidade de tempo, onde $\mathcal{C}_v \in \mathbb{R}^+$.

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 VM 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

O PLI soluciona o problema de escalonamento através das seguintes variáveis e constante:

- $x_{u,t,v}$: 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 granularidade da linha do tempo é a questão-chave desse problema, quando utilizamos programação linear inteira. O tempo pode ser representado por valores discretos ou contínuos. Na Amazon EC2, as horas parciais consumidas das VMs são cobradas como horas completas. Portanto, o escalonador apresentado neste trabalho utiliza apenas tempo discreto para representar a execução do *workflow*. Dependendo do nível de granularidade 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. Portanto, definimos λ como um *fator multiplicativo* que determina a granularidade do tempo discreto no problema de escalonamento. Seja $\mathcal{T} = \{\lambda, 2\lambda, 3\lambda, 4\lambda, \ldots, \Lambda\}$ a linha temporal discreta da execução do *workflow*, que varia de 1 ao *deadline* desejado ($1 \leq \Lambda \leq \mathcal{D}_\mathcal{G}$). O fator multiplicativo $\lambda \in \mathbb{N}^+$ foi apresentado em [Genez et al. 2012a]. O programa linear é formulado da seguinte maneira:

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

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

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

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

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

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

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

$$\text{(C7)}\ \sum_{v\in\mathcal{V}} y_{t,v} \leq \delta_i,\ \forall\, i\in\mathcal{I},\ \forall\, t\in\mathcal{T},\ \text{s.t. } \mathcal{B}_{i,v}=1 \qquad \text{(C8)}\ \sum_{v\in\mathcal{V}} y_{t,v} \leq \alpha_s,\ \forall\, s\in\zeta,\ \forall\, t\in\mathcal{T},\ \text{s.t. } \mathcal{K}_{s,v}=1$$

$$\text{(C9)}\ x_{u,t,v}\in\{0,1\},\ \forall\, u\in\mathcal{U},\ \forall\, t\in\mathcal{T},\ \forall\, v\in\mathcal{V} \qquad \text{(C10)}\ y_{t,v}\in\{0,1\},\ \forall\, t\in\mathcal{T},\ \forall\, v\in\mathcal{V}$$

As restrições em (C1) 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 (C2) estipulam que o número de nós do DAG em execução em uma VM v, em um determinado tempo t, não pode exceder o número de núcleos de processamento de v. As restrições em (C3) determinam que uma VM 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 (C4) determinam que a VM deve permanecer ativa durante o envio (ou recebimento) de dados gerados por nós precedentes. As restrições em (C5) especificam que toda tarefa deve ser executada em algum momento e em uma única VM. A restrição em (C6) determina que um nó u do DAG não pode terminar sua execução até que todas suas instruções de processamento tenham sido executadas e as operações de I/O tenham sido lidas/gravadas na VM v. As restrições em (C7) especificam que o número de VMs reservadas somado com o número de máquinas virtuais alugadas sob demanda não pode exceder o número máximo permitido para cada provedor de IaaS. As restrições em (C8) estabelecem que a quantidade de máquinas virtuais sendo utilizadas não pode exceder o limite estipulado no SLA. As duas últimas restrições, (C9) e (C10), especificam que as variáveis deste programa linear inteiro (x e y) só irão assumir os valores binários.

5. Experimentos

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 médio do escalonamento, o *makespan*[4] médio do *workflow* e o tempo de execução do *solver*. O simulador CloudSim[5] não foi utilizado neste trabalho pois não oferece suporte para execução de tarefas dependentes.

5.1. Configurações da Simulação

As simulações foram conduzidas utilizando três provedores de IaaS, sendo que cada um define seus próprios preços para as máquinas virtuais nos planos de reservas e sob-demanda. A Tabela 1 mostra as opções de máquinas virtuais, para os provedores IaaS A, B e C. Assumimos que os SLA existentes para as VMs reservadas (isto é, os elementos de ζ) compreendem quatro máquinas: uma instância *P* e uma *M* de ambos os provedores A e B. O número máximo de máquinas virtuais que podem ser alugadas de cada provedor de IaaS foi estipulado como: $\delta_A = 4$, $\delta_B = 7$, $\delta_C = 2$. 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

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

[4]Tempo de término do último nó do *workflow*

[5]https://code.google.com/p/cloudsim/

Tabela 1. Provedores de IaaS A, B e C

Provedor	Tipo de Máquinas Virtuais	Núcleos	Desempenho do núcleo	Desempenho de I/O	Preço demanda	Preço Reserva
A	P	1	2	baixo (0.1)	\$0.13	\$0.045
	M	2	2	moderado (0.2)	\$0.20	\$0.070
B	P	1	2	baixo (0.1)	\$0.238	\$0.083
	M	2	2.5	moderado (0.2)	\$0.276	\$0.097
	L	2	2.5	moderado (0.2)	\$0.401	\$0.141
	XL	4	2.5	alto (0.4)	\$0.552	\$0.194
	XXL-I/O	8	2.5	alto (0.4)	\$1.104	\$0.388
C	P	1	3	alto (0.4)	\$0.15	\$0.052
	M	2	3	alto (0.4)	\$0.25	\$0.088
	L-I/O	4	4	muito alto (1.0)	\$0.50	\$0.176
	XXL-I/O	8	4	muito alto (1.0)	\$0.80	\$0.281

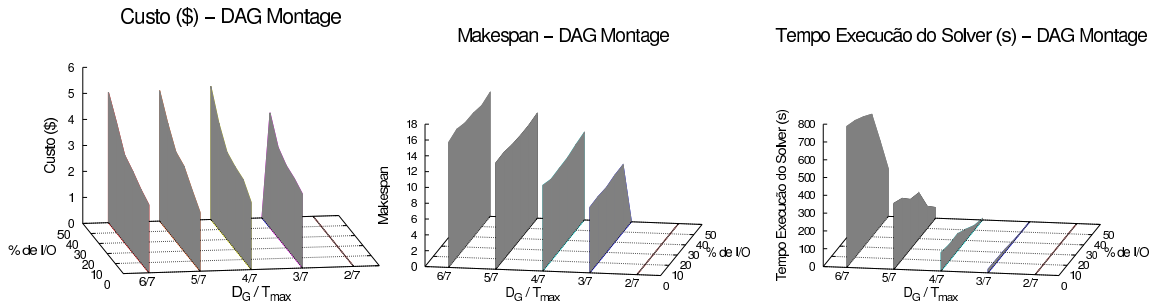
$\mathcal{L}$ no intervalo real $[2, 3]$ para os enlaces externos, enquanto para os enlaces internos o valor de $\mathcal{L}$ é obtido do intervalo real $[0.1, 0.2]$. Nosso objetivo é mostrar que ao escalonar *workflows* com uso intensivo de dados em um escalonador que não considera o tempo de operações de I/O pode não cumprir *deadlines* estipulados.

Nas simulações utilizamos apenas o DAG *Montage* [Deelman et al. 2005] por ser uma aplicação de uso intensivo de dados, ou seja, realiza muitas operações de acesso ao disco. Como dito anteriormente, 95% do tempo de execução do *Montage* pode ser gasto em operações de I/O. Neste trabalho, vamos mostrar que o escalonamento do *Montage* no escalonador original em [Genez et al. 2012b], que não considera operações de I/O, pode resultar em quebras de SLAs pelo não cumprimento de *deadlines*. Simulações foram executadas em um processador Intel® Xeon $X5660$ CPU $2.80GHz$ com $16GB$ de RAM.

5.2. Resultados

Realizamos 30 simulações para o DAG *Montage* (Figura 1) com as configurações dos provedores de IaaS presentes na Tabela 1, além das quatros máquinas virtuais inicialmente reservadas. Em cada simulação, o custo de computação de cada nó e os custos de comunicação de cada dependência foram obtidos aleatoriamente do intervalo real $[1, 3]$, enquanto os custos das operações de I/O foram obtidos do intervalo $[1, 15]$. Segundo [Deelman et al. 2008, Berriman et al. 2010, Juve et al. 2012], como 95% do tempo total de execução do *Montage* pode ser gasto em operações de I/O, o intervalo $[1, 15]$ foi definido da seguinte maneira: calculamos o tempo de execução de processamento de dois nós, um com custo de computação mínimo (igual a 1) e outro com custo máximo (igual a 3), na VM mais lenta (P do provedor A, por exemplo) e na mais rápida (XXL-I/O do provedor C, por exemplo) da Tabela 1. Essas máquinas foram escolhidas porque elas têm o pior e o melhor desempenho de acesso ao disco, respectivamente. O tempo de execução de processamento calculado representa 5% do tempo total da execução de cada nó; portanto, encontrar os 95% do restante do tempo, que corresponde ao tempo de acesso ao disco, foi trivial. Ao dividir cada tempo de acesso ao disco encontrado pelo desempenho de disco das respectivas VMs, encontramos o números de operações de I/O dos nós acima.

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 seqüencial 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 so-



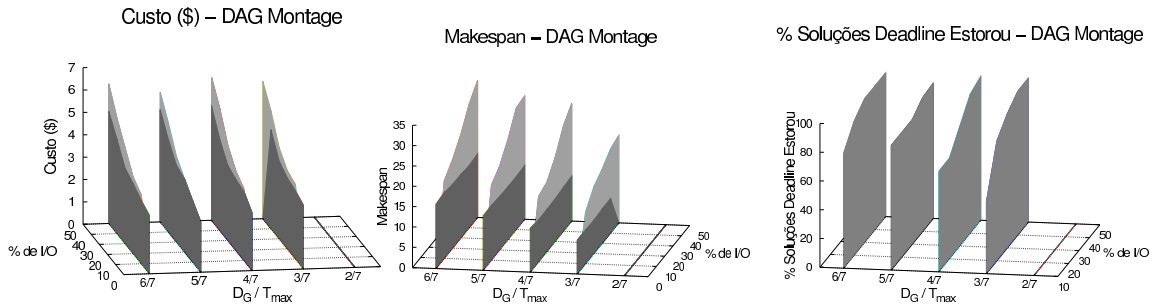
(a) Custo ($) (b) Makespan (c) Tempo de Execução (s)

Figura 2. Resultados para o DAG *Montage* considerando porcentagens de operações de I/O.

mente soluções inviáveis, enquanto *deadlines* iguais à $\mathcal{T}_{max} \times 7/7$ podem ser trivialmente alcançados colocando todas as tarefas no recurso mais barato. O divisor 7 foi escolhido apenas para avaliar a evolução do aumento do *deadline*, portanto outros divisores poderiam ser utilizados. Os gráficos foram plotados de acordo com a média das métricas das 30 simulações de cada porcentagem de instruções de I/O. Com intuito de obter um escalonamento, o *solver* teve o seu tempo de execução limitado a 15 minutos para cada simulação. Após esse tempo, o solver retornou a melhor solução até o momento, se encontrada.

A Figura 2 mostra os resultados das simulações para o DAG *Montage* com $\lambda = 1$ considerando as operações de I/O. Para mostrar a conseqüência do aumento do número de acessos ao disco rígido, variamos a porcentagem de operações de I/O de 0% a 50% do valor máximo do intervalo $[1, 15]$, onde 0% corresponde aos resultados do escalonador original. Variações acima de 50% serão detalhadas em seguida. Para $\mathcal{D}_{\mathcal{G}} = \mathcal{T}_{max} \times 2/7$, o *solver* rapidamente descobriu que todos os escalonamentos eram inviáveis, pois não tinham intervalos de tempos suficientes para escalonar todos os nós do DAG, mesmo utilizando uma granularidade fina ($\lambda = 1$) no conjunto de tempo $\mathcal{T}$. Por esse motivo que os tempos de execução para esse *deadline* foram baixos, como mostra a Figura 2(c). Note que, quando há soluções inviáveis, os valores do custo monetário e do *makespan* são zero, pois não houve uso de máquinas virtuais. O escalonamento do *Montage* com altas porcentagens de operações de I/O tornou-se ainda mais caro quando aumentamos o *deadline*. Por exemplo, em relação ao custo monetário médio do escalonamento sem considerar acesso ao disco rígido, o custo monetário médio teve um aumento de 21% (DAGs com 10% de operações de I/O) a 123% (DAGs com 50% de operações de I/O) para $\mathcal{D}_{\mathcal{G}} = \mathcal{T}_{max} \times 5/7$, e um aumento de 8% (DAGs com 10% de operações de I/O) a 95% (DAGs com 50% de operações de I/O) para $\mathcal{D}_{\mathcal{G}} = \mathcal{T}_{max} \times 6/7$. Conforme ilustra a Figura 2(b), o *makespan* médio manteve-se semelhante para todas as simulações, independentemente da porcentagem de I/O. Essa semelhança ocorre pois o *solver* procura um escalonamento viável com custos monetários mínimos e que respeita o *deadline* do DAG.

A Figura 3 ilustra os resultados das simulações que reutilizam o escalonamento original, que não considera o tempo gasto com acesso ao disco rígido, para escalonar os DAGs que contêm operações de I/O. Como mostram as Figuras 3(a) e 3(b), na curva em cinza escuro encontram-se os resultados da Figura 2, enquanto nas curvas em cinza claro encontram-se os resultados dos cálculos dos valores médios do custo monetário (Figura 3(a)) e do *makespan* (Figura 3(b)) quando o escalonamento original foi reutilizado. Como



(a) Custo ($) (b) Makespan (c) Quebra do *deadline* (%)

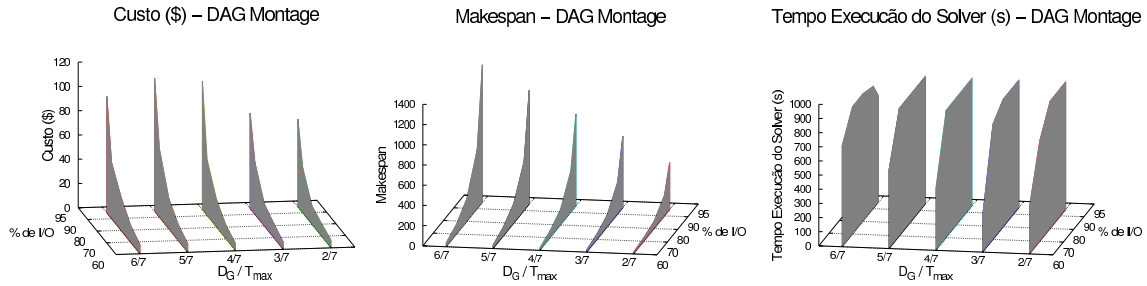
Figura 3. Reutilizando o escalonamento original (0% de operações de I/O) para escalonar os DAGs com operações de I/O.

o *solver* calculou rapidamente esses valores (levou aproximadamente 1 minuto para cada simulação), não apresentamos o gráfico da métrica *tempo de execução do solver*. Portanto, conforme esperado, o reaproveitamento do escalonamento original para escalonar DAGs que contêm demandas de I/O provavelmente quebraria o *deadline* estipulado pelo usuário. Dessa maneira, a Figura 3(c) apresenta a porcentagem das simulações onde o *makespan* dos escalonamentos foi maior que o *deadline* $\mathcal{D}_\mathcal{G}$; isto é, situações onde ocorreria uma possível quebra de SLA, o qual é estipulado entre o provedor de SaaS e seu cliente.

Independentemente do valor do *deadline* e da porcentagem de demanda de I/O do DAG, todas as simulações da Figura 3 tiveram características semelhantes. Por exemplo, para $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$, o custo monetário médio teve um aumento de 17% a 140% em comparação com o custo monetário médio dos escalonamentos originais (DAGs com 0% de operações de I/O) com os DAGs de 10% e 50% de operações de I/O. Além disso, o *makespan* médio também teve um aumento considerável de 20% a 114%, respectivamente. Em contrapartida, ao reaproveitar o escalonamento original, 80% das simulações com os DAGs de 10% de demanda de I/O quebraram o *deadline* estipulado pelo usuário, enquanto para os DAGs com 50% de operações de I/O esse valor foi de 100%, ou seja, todas as simulações quebraram o *deadline* $\mathcal{D}_\mathcal{G}$.

As curvas da Figura 2 foram adicionados na Figura 3 para ilustrar a comparação entre o escalonamento deste trabalho com o reaproveitamento do escalonamento original. Por exemplo, para DAGs com 50% de operações de I/O e $\mathcal{D}_\mathcal{G} = \mathcal{T}_{max} \times 6/7$, as soluções obtidas pelo *solver* deste trabalho foram 25% mais baratas que a reutilização do escalonamento original, enquanto o *makespan* foi reduzido em 53%. Entretanto, diferentemente das soluções onde os escalonamentos foram reaproveitados, todas as soluções encontradas pelo *solver* deste trabalho foram viáveis e cumpriram o *deadline* $\mathcal{D}_\mathcal{G}$. Embora a diferença dos custos monetários médios da Figura 3(a) aparente ser insignificante, o *solver* deste trabalho conseguiu encontrar um conjunto de VMs que minimizam o custo monetário do escalonamento do *workflow*, enquanto satisfaz o *deadline* desejado pelo usuário. Note que, ao reutilizar o escalonamento original, o custo monetário do escalonamento tende a valores baixos, porém o *makespan* do *workflow* tende a valores altos.

Para os DAGs com demandas de I/O maiores que 50%, tivemos que utilizar $\lambda > 1$ para reduzir o tamanho de $\mathcal{T}$ com intuito de encontrar soluções viáveis em tempo hábil. A Figura 4 mostra os resultados das simulações para os DAGs com 60% a 95% de operações de I/O. A escolha de λ foi feita da seguinte maneira: observamos que o tamanho médio



(a) Custo ($) (b) Makespan (c) Tempo de Execução (s)

Figura 4. Resultados considerando porcentagens de operações de I/O.

do conjunto $\mathcal{T}$ nas simulações da Figura 2 foi de 40 unidades de tempo em $\lambda = 1$. Dessa forma, antes de realizar o escalonamento, precisavamos que $\frac{\mathcal{D}_G}{\lambda} \leq 40$, $\forall \lambda \in \mathbb{N}^+$. Resumidamente, os gráficos da Figura 4 seguem a mesma tendência dos gráficos da Figura 2, ou seja, quanto maior a porcentagem gasto com tempo de acesso ao disco rígido, maior o tempo do consumo da máquina virtual e, portanto, maior o custo monetário do escalonamento. No entanto, o uso da granularidade grossa na linha do tempo gera uma penalidade no escalonamento, como, por exemplo, aumento no custo monetário e do *makespan*. Essa penalidade ocorre pois o *solver* "acredita" que uma unidade de tempo discretizada em $\lambda > 1$ será totalmente utilizada pelo nó que for atribuído. Dessa forma, um mecanismo para corrigir o erro gerado por essa penalidade já está sendo considerado.

Os resultados obtidos sugerem que *workflows* com baixa porcentagem de operações de I/O (10% e 20%, por exemplo) aumentam levemente o custo monetário do escalonamento. No entanto, acima de 30% de operações de I/O, o aumento do custo monetário começa a se tornar significante, impactando, por exemplo, no orçamento do provedor de SaaS. Ao reutilizar escalonamentos que não consideram o tempo gasto com demandas de I/O das tarefas do *workflow*, o *makespan* do *workflow* aumenta consideravelmente, quebrando a maioria dos *deadlines* $\mathcal{D}_G$. Isso ocorre porque, no escalonamento original, o *solver* escolhe provavelmente VMs baratas cujo desempenho de acesso ao disco é lento. Portanto, escalonar aplicações com uso intensivo de dados e ignorar o tempo gasto com operações de I/O pode afetar significativamente o *makespan* real da aplicação e, além disso, não cumprir o *deadline* estipulado pelo usuário.

6. Conclusão

Neste trabalho apresentamos uma abordagem para escalonar *workflows* com uso intensivo de dados em nuvens. Realizamos simulações com o DAG *Montage* que representa uma aplicação do mundo real. Resultados mostraram que utilizar escalonadores que não consideram tempo de acesso ao disco do *workflow* pode resultar em aumento do *makespan* real da aplicação e, consequentemente, quebras de *deadlines*. Em ambientes de nuvens computacionais, o não cumprimento do *deadline* pode acarretar em quebras de contratos SLAs entre o provedor de serviços (SaaS, por exemplo) e seu cliente. Aliás, dependendo das cláusulas do contrato SLA, pode acarretar em prejuízos financeiros ao provedor de serviço devido à multa paga ao cliente por essa quebra de contrato. Além disso, o provedor de SaaS pode ter seu orçamento afetado pela maior utilização das máquinas virtuais pagas aos provedores de IaaS, enfatizando, portanto, a importância do desenvolvimento de escalonadores que consideram o tempo dos acessos ao disco realizados pelas tarefas do

workflow. Como trabalhos futuros apontamos a avaliação do escalonamento em condições de incerteza, onde as informações de entrada do escalonamento podem ser imprecisas.

Agradecimentos

Os autores agradecem à CAPES, à FAPESP (2012/02778-6) e ao CNPq pelo apoio financeiro, e à IBM por providenciar a ferramenta IBM ILOG CPLEX.

Referências

Berriman, G. B., Juve, G., Deelman, E., Regelson, M., and Plavchan, P. (2010). The application of cloud computing to astronomy: A study of cost and performance. In *6th IEEE International Conference on e-Science Workshops*, E-SCIENCEW '10, pages 1–7. IEEE Computer Society.

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., Livny, M., Berriman, B., and Good, J. (2008). The cost of doing science on the cloud: the montage example. In *ACM/IEEE conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA. IEEE Press.

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. *Journal of Scientific Programming*, 13:219–237.

Genez, T. A. L., Bittencourt, L. F., and Madeira, E. R. M. (2012a). Discretização do tempo na utilização de programação linear para o problema de escalonamento de workflows em múltiplos provedores de nuvem. In *XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, SBRC '12.

Genez, T. A. L., Bittencourt, L. F., and Madeira, E. R. M. (2012b). Workflow scheduling for SaaS / PaaS cloud providers considering two SLA levels. In *2012 IEEE/IFIP Network Operations and Management Symposyum*, NOMS '12, pages 906 –912.

Juve, G., Deelman, E., Berriman, G. B., Berman, B. P., and Maechling, P. (2012). An evaluation of the cost and performance of scientific workflows on amazon EC2. *Journal of Grid Computing*, 10(1):5–21.

Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B., and Maechling, P. (2009). Scientific workflow applications on amazon ec2. In *5th IEEE International Conference on E-Science Workshops*, E-SCIENCEW '09, pages 59–66.

Wu, Z., Ni, Z., Gu, L., and Liu, X. (2010). A revised discrete particle swarm optimization for cloud workflow scheduling. In *2010 International Conference on Computational Intelligence and Security*, CIS '10, pages 184 –188.

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.

Zhu, M., Wu, Q., and Zhao, Y. (2012). A cost-effective scheduling algorithm for scientific workflows in clouds. In *31st IEEE International Performance Computing and Communications Conference*, IPCCC'12.