`GraspCC-LB`: Dimensionamento de Recursos para Execução de *Workflows* em Ambientes de Computação de Alto Desempenho

Luis Carlos Ramos Alvarenga[1], **Yuri Frota**[2], **Daniel de Oliveira**[2], **Rafaelli Coutinho**[1]

[1]CEFET/RJ - Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

[2]Instituto de Computação - Universidade Federal Fluminense (IC/UFF)

luis.alvarenga@aluno.cefet-rj.br, {yuri,danielcmo}@ic.uff.br,

rafaelli.coutinho@cefet-rj.br

Resumo. *Com a crescente complexidade das simulações computacionais e o aumento do volume de dados processados, a execução de workflows científicos em ambientes HPC torna-se cada vez mais necessária. No entanto, dimensionar a quantidade necessária de recursos para essa execução pode ser uma tarefa desafiadora, uma vez que implica considerar a estrutura do workflow e as características do ambiente. Este artigo apresenta a heurística* `GraspCC-LB`*, baseada no procedimento de busca adaptativa randomizada gulosa (GRASP), para o dimensionamento de recursos em ambientes HPC. A* `GraspCC-LB` *considera a estrutura do workflow em layers para realizar o dimensionamento, o que a difere das abordagens existentes. A* `GraspCC-LB` *foi avaliada utilizando traces reais de workflows das áreas de bioinformática e astronomia, demonstrando resultados promissores.*

Abstract. *With the increasing complexity of computational simulations and the growing volume of processed data, the execution of scientific workflows in HPC environments becomes more necessary. However, dimensioning the required amount of resources for this execution can be a challenging task, as it involves considering the workflow structure and the environment characteristics. This paper introduces the* `GraspCC-LB` *heuristic, based on the Greedy Randomized Adaptive Search Procedure (GRASP), for resource estimation in HPC environments.* `GraspCC-LB` *considers the workflow structure in layers to perform the estimation, making it different from existing approaches. The* `GraspCC-LB` *was evaluated using real traces of workflows from the fields of bioinformatics and astronomy, and the results are promising.*

1. Introdução

Muitos experimentos científicos no campo da Ciência Computacional e Engenharia (CSE, de *Computer Science and Engineering*) dependem de simulações computacionais em larga escala, as quais frequentemente exigem recursos de Computação de Alto Desempenho (HPC, de *High-Performance Computing*) para obter resultados em um tempo aceitável [Gil et al., 2007]. Esses experimentos podem ser representados de diversas formas (*e.g.*, *scripts*, aplicações monolíticas), mas são comumente modelados como *Workflows*

Científicos (deste ponto em diante referenciados apenas como *workflows*) [de Oliveira et al., 2019]. Os *workflows* representam um conjunto de tarefas (que são invocações de um ou mais programas) e dependências de dados que representam as relações de produção e consumo de dados entre as tarefas no fluxo. Por conta das dependências de dados, os *workflows* comumente possuem múltiplos *Níveis* (*i.e.*, *Layers*). Cada *layer* representa um conjunto de tarefas que não possuem dependências entre si, conforme exemplificado na Figura 1, e que podem (*a priori*) ser executadas em paralelo.

Por conta da demanda por HPC das aplicações que compõem o *workflow*, diferentes ambientes computacionais podem ser usados para sua execução, *e.g.*, nuvens de computadores, *clusters* e supercomputadores. Experimentos científicos modelados como *workflows* têm sido amplamente executados nesses ambientes [Deelman et al., 2018], uma vez que os sistemas de *workflow* existentes já proveem mecanismos otimizados para escalonamento de tarefas e transferência de dados nesses ambientes. Apesar de tais sistemas representarem um avanço no apoio à execução eficiente de *workflows*, eles ainda carecem de soluções no que se refere ao dimensionamento de recursos computacionais necessários para a execução de *workflows*. O dimensionamento eficiente pode determinar o seu sucesso ou fracasso. Por exemplo, se os recursos reservados para a execução forem subdimensionados, o *workflow* pode apresentar um tempo de execução elevado, e caso os recursos sejam superdimensionados, o custo financeiro pode se tornar inviável (caso a execução seja em uma nuvem pública, por exemplo).

Diversas pesquisas já vem sendo realizadas para prover dimensionamento eficiente de *workflows* em ambientes HPC [Coutinho et al., 2015; Moschakis and Karatza, 2015; Mohammadi et al., 2018; Rosa et al., 2021]. Entretanto, tais abordagens assumem que as tarefas de um *workflow* ou são completamente independentes (o que comumente gera um subdimensionamento de recursos, uma vez que dependências de dados não são consideradas), ou que as mesmas podem ser agrupadas *a priori*, e que as tarefas dentro de um mesmo agrupamento podem ser executadas como *Bag-of-tasks* (BoT). Dependendo da estrutura do *workflow*, agrupar tarefas *a priori* assumindo uma semântica pré-definida pode não produzir o melhor dimensionamento possível [Coutinho et al., 2016]. Tarefas em um mesmo *layer* do *workflow* podem executar aplicações diferentes, mas eventualmente podem apresentar um melhor desempenho caso sejam executadas em um mesmo recurso computacional (*e.g.*, uma máquina virtual específica). Isso acontece porque o programa conserva seu contexto de execução, propicia a reutilização de *cache* e reduz sobrecargas de comunicação e inicialização.

De forma a permitir que o dimensionamento do ambiente seja realizado considerando as dependências do *workflow* e sem a necessidade de um agrupamento realizado *a priori*, este artigo propõe a heurística `GraspCC-LB`. A `GraspCC-LB` foi desenvolvida como um procedimento de busca adaptativa gulosa (GRASP) que considera a especificação do *workflow* orientada a *layers* e, em cada *layer*, as tarefas do *workflow* são agrupadas em *buckets*. Os *buckets* são criados de acordo com as tarefas que podem ser executadas no momento (cuja dependência de dados já foi satisfeita), *i.e.*, o usuário não necessita definir a semântica do agrupamento. Essa técnica foi intitulada *Layered-Bucket* (LB). No que se refere a execução, os *buckets* são executados de forma paralela, enquanto as *layers* são executadas de forma sequencial. A `GraspCC-LB` foi avaliada utilizando *traces* de *workflows* reais e os resultados se mostraram promissores.

O artigo está organizado em cinco seções, além da Introdução. Na Seção 2, são apresentados os trabalhos relacionados. Na Seção 3, é apresentado o modelo matemático proposto para resolver o problema de dimensionamento *Layered-Bucket*. Na Seção 4, é apresentada a heurística `GraspCC-LB`. Na Seção 5, são relatados os resultados experimentais, enquanto que a Seção 6 conclui o presente artigo.

2. Trabalhos Relacionados

Diversos trabalhos encontrados na literatura propõem soluções para dimensionar recursos para a execução de aplicações em ambientes HPC [Moschakis and Karatza, 2015; Coutinho et al., 2015; Lin et al., 2016; Deldari et al., 2017; Abdi et al., 2018; Zhou et al., 2019; Song et al., 2020; Rosa et al., 2021]. No entanto, algumas dessas abordagens focam apenas no dimensionamento do ambiente para aplicações que não apresentam dependências de dados. Por exemplo, Abdi et al. [2018] propõem um modelo matemático e uma abordagem baseada na metaheurística GRASP, chamada GRASP-FC, que considera restrições de prazo para o dimensionamento de recursos de aplicações BoT. Já Coutinho et al. [2013] também apresentam uma abordagem baseada em GRASP para o problema de gerenciamento de recursos em nuvem para aplicações paralelas, visando reduzir o custo financeiro e o tempo de execução das aplicações. No entanto, é importante destacar que ambos os trabalhos não estão conectados ao conceito de *workflows*.

Por outro lado, diversos trabalhos já se concentram no dimensionamento de recursos para a execução de *workflows* em ambientes HPC [Coutinho et al., 2015; Malawski et al., 2015; Mohammadi et al., 2018; Rosa et al., 2021]. Malawski et al. [2015] e Mohammadi et al. [2018] utilizam programação linear inteira para minimizar os custos financeiros, levando em conta os *deadlines* e as dependências entre as tarefas. Coutinho et al. [2015], por sua vez, propõem um modelo matemático combinado com uma abordagem heurística para o dimensionamento de recursos em nuvens de computadores, buscando reduzir tanto os custos quanto o tempo de execução, levando em consideração a quantidade de recursos computacionais disponíveis, o custo financeiro e o *deadline*.

Os trabalhos mencionados anteriormente [Coutinho et al., 2015; Malawski et al., 2015; Mohammadi et al., 2018; Rosa et al., 2021] realizam um agrupamento das tarefas de acordo com o programa que elas executam (ou seja, a técnica *First-Activity-First* [Ogasawara et al., 2011]). No entanto, essa abordagem comumente não permite definir o melhor dimensionamento. Rosa et al. [2021] apresentam um serviço de previsão de recursos computacionais e custos financeiros (CRCPs), que estima a quantidade necessária de recursos computacionais para a execução do *workflow*. Essa previsão é realizada por meio da combinação de uma heurística e do uso de regressão linear, baseando-se em um histórico de execuções. No entanto, nem sempre esse histórico de execuções se encontra disponível para utilização.

Algumas abordagens não se concentram em técnicas de dimensionamento, mas sim em estimativas da quantidade de recursos necessários para o escalonamento de *workflows*. Por exemplo, Lin et al. [2016] e Deldari et al. [2017] apresentam heurísticas para minimizar o custo financeiro da execução do *workflow*, determinando quais tipos e quantidades de máquinas virtuais devem ser instanciados em um ambiente de nuvem. De forma semelhante, Liu et al. [2016] propõem um modelo que leva em conta o tempo de execução e os custos financeiros, considerando restrições de recursos e prazos para o escalonamento

de *workflows* em nuvens. No entanto, assim como o trabalho de Rosa et al. [2021], essas abordagens dependem de uma base de dados histórica de execuções do *workflow*, o que nem sempre está disponível. Portanto, para abordar as lacunas deixadas pelos trabalhos anteriormente citados, neste artigo, foi proposto o dimensionamento de recursos para a execução de *workflows* em ambientes HPC realizando um agrupamento por *layer*. Ou seja, o agrupamento é gerado de acordo com as tarefas disponíveis para a execução, sem que o usuário precise definir a semântica do agrupamento *a priori* e sem que dados históricos sejam necessários. Para isso, as próximas seções apresentam um modelo matemático exato intitulado `CC-LB` e uma heurística chamada `GraspCC-LB`.

3. Definição e Modelagem do Problema

Um *workflow* pode ser representado por $W(T, E)$, onde T é um conjunto de tarefas denotado por $T = \{T_1, T_2, ..., T_k\}$, e E é um conjunto de arestas direcionadas que definem as dependências de dados. A aresta $E_{gh}(T_g, T_h)$ indica que a tarefa T_g é considerada predecessora da tarefa T_h, enquanto T_h é a sucessora de T_g, e que $(T_g, T_h) \in T$. Uma tarefa de entrada no *workflow* não possui predecessores, e as tarefas de saída são aquelas que não têm sucessores. Esta relação de dependência é crucial para garantir a execução correta das tarefas, *i.e.*, garantir que as tarefas predecessoras sejam sempre concluídas antes do início da execução das tarefas sucessoras a fim de evitar erros na execução do *workflow*. Um exemplo de *workflow* é visto na Figura 1, onde é possível observar que $\{T_1, T_2, T_3, T_4, T_6\}$ são tarefas de entrada, $\{T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\}$ são tarefas de saída, T_9 é uma tarefa sucessora de $\{T_5, T_6\}$, e $\{T_1, T_2, T_3, T_4\}$ são tarefas predecessoras de T_5.

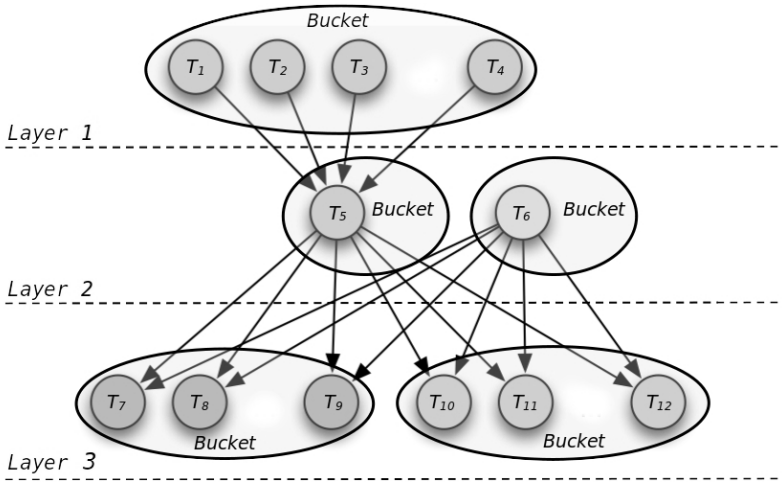

Figura 1. *Workflow* de acordo com a abordagem *Layered-Bucket*

As tarefas que não possuem dependências de dados entre si podem ser divididas em *Layers* de processamento, permitindo que sejam executadas em paralelo dentro de um mesmo *layer*. Na Figura 1 é possível ver a divisão de um *workflow* em três *layers*: *Layer 1* composto pelas tarefas $\{T_1, T_2, T_3, T_4\}$; *Layer 2*, pelas tarefas $\{T_5, T_6\}$; e, por fim, o *Layer 3*, por $\{T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\}$. As tarefas pertencentes a um *layer* podem ser agrupadas em *buckets*, seguindo uma abordagem chamada *Layered-Bucket*. Um *bucket* pode ser formado apenas por uma tarefa (*e.g.,* os *buckets* que compreendem separadamente as tarefas T_5 e T_6 na Figura 1) ou por um conjunto de tarefas independentes entre si que podem ser agrupadas e executadas como um BoT (*e.g.,* o *bucket* do *layer* 1). A

divisão do *workflow* em *layers* e *buckets* tem como objetivo oferecer maior flexibilidade para abordagem de dimensionamento, uma vez que permite que múltiplas formas de agrupamentos sejam avaliadas. No que se refere ao processamento, os *layers* são executados de forma sequencial enquanto que os *buckets* são executados em paralelo.

Uma vez que o *workflow W* está formalizado, é necessário modelar o problema de dimensionamento. O dimensionamento de recursos em ambientes HPC é uma tarefa complexa devido à variedade de tipos de máquinas disponíveis nesses ambientes e pelos diferentes modelos de cobrança pelo uso. Por exemplo, o supercomputador Santos Dumont (SDumont)[1] possui uma arquitetura heterogênea com 36.472 núcleos de CPU com arquitetura multi-core, GPUs e um nó MESCA2, com um número elevado de núcleos (240) e arquitetura de memória compartilhada. Os usuários do SDumont recebem cotas de Unidade de Alocação (UA) para executar suas tarefas por hora. Outro exemplo são as nuvens de computadores, que oferecem diferentes tipos de máquinas virtuais para instanciação, onde cada tipo de máquina virtual possui configurações específicas de CPU, memória, armazenamento e custos financeiros. O custo da execução em nuvem comumente é definido em dólares de acordo com a janela de tempo de uso. Dessa forma, os usuários precisam definir quais tipos e quantidades de máquinas são adequados para executar seus *workflows* em um determinado ambiente HPC, de modo a minimizar um objetivo específico, como o tempo de execução e/ou custo financeiro.

Para formalizar o problema de dimensionamento, é necessário introduzir algumas notações. Seja P o conjunto de tipos de máquinas fornecidos por uma infraestrutura computacional, como nuvens computacionais ou *clusters*, durante um período de tempo. Cada máquina $p \in P$ tem um custo financeiro C_p associado por período de tempo (que pode ser medido em qualquer unidade, *e.g.*, dólar ou em UAs, como no SDumont) e recursos computacionais como armazenamento em disco DS_p, capacidade de memória MC_p e processamento GF_p, medido em GFLOPs por período de tempo. Além disso, não é incomum que determinados ambientes definam um limite máximo de máquinas v_{max} alocadas para cada usuário por período de tempo.

Devem também ser definidos o custo financeiro máximo c_{max} do *workflow*, o tempo máximo de execução t_{max} (*i.e.*, *deadline*) do *workflow*, o conjunto de $layers\ L$, a quantidade de $buckets\ B_l$ em cada *layer* $l \in L$. Já os requisitos como armazenamento em disco ds_b^l, a capacidade de memória mc_b^l, a demanda por processamento gf_b^l em GFLOPs, e o número máximo de máquinas disponíveis v_b^l são definidos por $bucket$ $b \in B_l$ do $layer\ l \in L$, para cada período de tempo. A variável binária x_{pitbl} assume o valor 1 se e somente se a máquina $i \in \{1, \cdots, v_{max}\}$ do tipo $p \in P$ estiver alocada no tempo $t \in T = \{1, \cdots, t_{max}\}$ para o $bucket\ b \in B_l$ no $layer\ l \in L$, senão $x_{pitbl} = 0$. A variável inteira z_{pi} representa o último período de tempo em que a máquina $i \in \{1, \cdots, v_{max}\}$ do tipo $p \in P$ foi alocada. Para finalizar, a variável z, também inteira, contém o último período de tempo em que uma máquina foi alocada pelo usuário. A variável z representa o $makespan$ (duração da execução do *workflow*) estimado. O modelo, intitulado `CC-LB`, pode ser formulado conforme apresentado a seguir:

$$\text{(CC-LB)} \qquad \min(\alpha_1 \sum_{p \in P} \sum_{i=1}^{v_{max}} \sum_{t \in T} \sum_{l \in L} \sum_{b \in B_l} \frac{C_p\ x_{pitbl}}{c_{max}} + \alpha_2\ \frac{z}{t_{max}}) \tag{1}$$

[1] `https://sdumont.lncc.br/`

sujeito a

$$\sum_{p\in P}\sum_{i=1}^{v_{max}} DS_p \cdot x_{pitbl} \geq ds_b^l \cdot x_{p'i'tbl}, \qquad \forall l \in L, \forall b \in B_l, \forall t \in T, \forall p' \in P, \forall i' \in \{1,\ldots,v_{max}\} \quad (2)$$

$$\sum_{p\in P}\sum_{i=1}^{v_{max}} MC_p \cdot x_{pitbl} \geq mc_b^l \cdot x_{p'i'tbl}, \qquad \forall l \in L, \forall b \in B_l, \forall t \in T, \forall p' \in P, \forall i' \in \{1,\ldots,v_{max}\} \quad (3)$$

$$\sum_{p\in P}\sum_{i=1}^{v_{max}}\sum_{t\in T} GF_p \cdot x_{pitbl} \geq gf_b^l \qquad \forall l \in L, \forall b \in B_l \quad (4)$$

$$\sum_{\substack{t'\in T\\ t'\leq t}}\sum_{\substack{b'\in B_l\\ b'>b}} x_{pit'b'l} \leq (1 - x_{pitbl})\mathbf{M}, \qquad \forall l \in L, \forall b \in B_l, \forall t \in T, \forall p \in P, \forall i \in \{1,\ldots,v_{max}\} \quad (5)$$

$$\sum_{l\in L}\sum_{b\in B_l} x_{pi+1tbl} \leq \sum_{l\in L}\sum_{b\in B_l} x_{pitbl}, \qquad \forall t \in T, \forall p \in P, \forall i \in \{1,\ldots,v_{max}-1\} \quad (6)$$

$$\sum_{p'\in P}\sum_{i'=1}^{v_{max}}\sum_{\substack{t'\in T\\ t'\leq t}}\sum_{\substack{l'\in L\\ l'>l}}\sum_{b'\in B_{l'}} x_{p'i't'b'l'} \leq (1 - x_{pitbl})\mathbf{M}, \qquad \forall l \in L, \forall b \in B_l, \forall t \in T, \forall p \in P, \forall i \in \{1,\ldots,v_{max}\} \quad (7)$$

$$\sum_{p\in P}\sum_{i=1}^{v_{max}}\sum_{l\in L}\sum_{b\in B_l} x_{pitbl} \leq v_{max}, \qquad \forall t \in T \quad (8)$$

$$\sum_{p\in P}\sum_{i=1}^{v_{max}} x_{pitbl} \leq v_b^l, \qquad \forall l \in L, \forall b \in B, \forall t \in T \quad (9)$$

$$z_{pi} \geq t\sum_{l\in L}\sum_{b\in B_l} x_{pitbl}, \qquad \forall t \in T, \forall p \in P, \forall i \in \{1,\ldots,v_{max}\} \quad (10)$$

$$z \geq z_{pi} \qquad \forall p \in P, \forall i \in \{1,\ldots,v_{max}\} \quad (11)$$

$$\sum_{p\in P}\sum_{i=1}^{v_{max}} C_p \cdot z_{pi} \leq c_{max} \quad (12)$$

$$x_{pitbl} \in \{0,1\}, \qquad \forall l \in L, \forall b \in B_l, \forall t \in T, \forall p \in P, \forall i \in \{1,\ldots,v_{max}\} \quad (13)$$

$$z, z_{pi} \in \mathbb{Z} \qquad \forall p \in P, \forall i \in \{1,\ldots,v_{max}\} \quad (14)$$

onde $(\alpha_1 + \alpha_2) = 1$.

No modelo *CC-LB* apresentado anteriormente, a função objetivo 1 busca a minimização do custo financeiro e do tempo de processamento total do *workflow*. Os pesos α_1 e α_2 atribuem a importância dos objetivos de custo e de tempo de processamento definidos pelo usuário, respectivamente. Note que as duas parcelas da função objetivo estão normalizadas, usando c_{max} para o custo e t_{max} para o tempo. As restrições 2 e 3 garantem que as quantidades de armazenamento e memória contratadas (*i.e.*, reservadas) sejam maiores que as demandas de cada $bucket$ dos $layers$, em cada unidade de tempo, respec-

tivamente. De forma semelhante, a restrição 4 impõe que a capacidade de processamento seja suficiente para atender a demanda de cada $bucket$ dos $layers$. A ordem de processamento dos $buckets$ de um $layer$ é imposta pela restrição 5, e a restrição 6 foi incluída para estabelecer uma ordem entre a alocação das máquinas no $bucket$ e eliminar soluções simétricas (*i.e.*, soluções diferentes, mas com a mesma configuração). Ainda em relação ao processamento, a restrição 7 garante a ordem de processamento entre os $layers$.

A desigualdade 8 garante que o número de máquinas alocadas simultaneamente não exceda o limite máximo disponibilizado pelo ambiente computacional escolhido. Já a restrição 9 garante o limite de máquinas disponibilizadas por $bucket$ em cada período de tempo. A restrição 10 assegura a correta interpretação da variável z_{pi}, ou seja, o último período de tempo em que uma determinada máquina foi alocada. A desigualdade 11 determina o maior tempo de processamento dentre todas as máquinas alocadas, isto é, o tempo total de processamento do *workflow*. A restrição 12 garante que o custo de alocação das máquinas não ultrapasse o custo máximo disponível (seja ele em dólar ou UAs). Finalmente, as restrições 13 e 14 definem os requisitos de integralidade e não-negatividade das variáveis.

4. `GraspCC-LB`: Uma Heurística para Dimensionamento de Recursos

A heurística `GraspCC-LB` é constituída por duas etapas principais: *(i)* a etapa de pré-processamento do *workflow* para a definição dos $layers$ e dos $buckets$; e *(ii)* a etapa de processamento, onde é realizado de fato o dimensionamento dos recursos para execução de cada $bucket$ dos $layers$. A heurística `GraspCC-LB`, apresentada no Algoritmo 1, recebe como parâmetros de entrada o $workflow$ a ser dimensionado com os requisitos de disco, memória e processamento; o conjunto de máquinas P que possui suas características de disco, memória e capacidade de processamento; as restrições de custo c_{max} e tempo t_{max} do usuário; e os pesos atribuídos ao custo α_1 e ao tempo α_2. O procedimento inicia com uma solução global s^* vazia de custo infinito e realiza a etapa de pré-processamento para definição dos *layers* e dos *buckets* de acordo com a especificação do $workflow$ recebida. Para isso, o procedimento $LayeredBucket$ é invocado para definir os *layers* do *workflow* e os requisitos de disco ds, memória mc e processamento gf de cada *bucket* identificado, baseados nas exigências computacionais das tarefas do $workflow$.

Algoritmo 1: GraspCC-LB ($workflow, P, c_{max}, t_{max}, \alpha_1, \alpha_2$)

```
1  s* ← ∅; F(s*) ← ∞;
2  layers ← LayeredBucket(workflow);
3  iter ← 0;
4  while (iter ≤ maxIter) do
5      s ← ∅;
6      for l ∈ layers do
7          for b ∈ l do
8              s̄ ← GraspCC(P, c_max, t_max, ds_b^l, mc_b^l, gf_b^l, α1, α2);
9              s ← s ∪ s̄;
10     iter ← iter + 1;
11     if ((s is feasible) & (F(s) < F(s*))) then
12         s* ← s; iter ← 0;
13 return s*;
```

O procedimento *LayeredBucket* é detalhado no Algoritmo 2, e estrutura o $workflow$ recebido em *layers* e *buckets*, oferecendo um agrupamento das tarefas em tempo real. Cada tarefa t do $workflow$ possui um conjunto de metadados associados como possíveis classificações da atividade act_t, requisitos de disco d_t, requisitos de memória m_t, requisito de processamento g_t, conjuntos de tarefas sucessoras e predecessoras, $succ_t$ e $pred_t$, respectivamente, e a ordem $order_t$ que lhe será atribuída. Todas as tarefas do $workflow$ possuem inicialmente ordem zero ($order_t = 0$). O procedimento *LayeredBucket* estabelece, inicialmente, a estrutura de dados $layers$ como um conjunto vazio. Esta estrutura é definida como uma lista ordenada, na qual cada elemento (*i.e.*, um *layer*) é constituído por um conjunto de *buckets*. Cada $bucket$, por sua vez, é estruturado considerando o somatório dos requisitos computacionais de disco ($\sum d_t$), memória ($\sum m_t$) e processamento ($\sum g_t$) do conjunto de tarefas que possuem a mesma classificação de atividade definida anteriormente. É importante ressaltar que a classificação da atividade act_t pode possuir múltiplas semânticas (*e.g.*, mesmo programa invocado, atividades que executam em paralelo com MPI, *etc*.), permitindo que a `GraspCC-LB` explore diferentes formas de agrupamento de tarefas.

Algoritmo 2: LayeredBucket ($workflow$)

```
1  layers ← ∅;
2  for t ∈ workflow do
3      for k ∈ succ_t do
4          if order_k ≤ order_t then
5              order_k ← order_t + 1;
6  for t ∈ workflow do
7      ds ← d_t; mc ← m_t; gf ← g_t;
8      for t' ∈ workflow | t ≠ t' do
9          if order_t = order_t' ∧ act_t = act_t' then
10             ds ← ds + d_t'; mc ← mc + m_t'; gf ← gf + g_t';
11             workflow ← workflow − t';
12     bucket ← (ds, mc, gf);
13     layers[order_t] ← layers[order_t] + bucket;
14 return (layers);
```

A seguir, o procedimento *LayeredBucket* insere uma tarefa artificial t, que passa a ser a única tarefa de entrada do *workflow*. Esta tarefa artificial é a predecessora de todas as tarefas de entrada reais do *workflow*. A partir dela, cada tarefa t do $workflow$ é processada iterativamente. As tarefas sucessoras de t são examinadas, comparando suas ordens com a de t. Caso a ordem da tarefa sucessora k não seja superior à ordem da tarefa t, então a ordem da tarefa k é atualizada para a ordem subsequente a da tarefa t. O processo continua até que todas as tarefas tenham sido processadas. As ordens finais de cada tarefa representam a identificação do seu *layer*. Após a definição das ordens das tarefas, para cada ordem, os $buckets$ são criados nos $layers$. Cada $bucket$ é construído por meio do somatório de cada requisito de processamento das tarefas que compartilham as mesmas características act_t (classificação de atividade) em um mesmo *layer*. Ao final, a estrutura de $layers$ é retornada pelo procedimento *LayeredBucket*.

Após a definição dos *layers*, o procedimento apresentado no Algoritmo 1 inicia

a etapa de dimensionamento e adota a noção de melhoria sucessivas usando o parâmetro $maxIter$. Este parâmetro determina o número máximo de tentativas sem avanço na otimização da solução encontrada. Para cada iteração, inicia-se uma solução s vazia e processa-se, para cada *layer* l, seus *buckets* de maneira individual. Para cada *bucket* $b \in l$, o procedimento `GraspCC` [Coutinho et al., 2015] é invocado para construir uma solução local s, o que é aceitável uma vez que o `GraspCC` é um método para dimensionamento de aplicações executadas como BoT, e as tarefas em um mesmo *bucket* são executadas dessa forma. O `GraspCC` recebe como parâmetro o conjunto de recursos P, as restrições c_{max} e t_{max}, os requisitos computacionais ds_b^l, mc_b^l e gf_b^l, e os pesos atribuídos ao custo α_1 e ao tempo α_2.

O `GraspCC` também adota a noção de melhorias sucessivas e possui dois sub-procedimentos: a construção da solução inicial *coCC* e a busca local *lsCC*, conforme [Coutinho et al., 2015]. O sub-procedimento *coCC* começa com uma solução vazia e adiciona máquinas à solução de forma aleatória e gulosa. Uma máquina p com melhor custo financeiro e poder de processamento $(\alpha_1 C_p + \alpha_2 GF)$ é adicionada para o primeiro período de tempo da solução. O processo se repete até que a solução satisfaça os requisitos de armazenamento ds_b^l e memória mc_b^l, e o número de máquinas selecionadas não exceda o limite máximo v_b^l. No entanto, não há garantia que o sub-procedimento *coCC* alcance uma solução viável ou localmente ótima em relação a alguma vizinhança. Assim, o sub-procedimento $lsCC$ inicia com a solução fornecida por *coCC* e, de maneira iterativa, substitui a solução atual pela solução de custo mínimo $F(s)$ na vizinhança. A busca local é interrompida quando nenhuma melhoria é identificada na vizinhança da solução atual.

A solução $\bar{s}$, gerada pelo `GraspCC` é incorporada à solução local s. Caso s seja viável, *i.e.*, atenda as restrições de custo financeiro c_{max} e tempo t_{max} do usuário, e apresente um custo inferior à melhor solução encontrada até o momento, s se torna a nova melhor solução global e o contador de iterações é reiniciado. O algoritmo é finalizado quando o número máximo de iterações for atingido, sem que haja melhoria na solução global, retornando, então, a melhor solução encontrada (s^*).

5. Avaliação Experimental

De forma a avaliar a heurística `GraspCC-LB`, foram realizados uma série de experimentos com *traces* de workflows reais. Os experimentos foram executados em um computador Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz, 130 GB de RAM, sistema operacional Linux Ubuntu v20.04.1. O modelo `CC-LB` foi implementado na linguagem Python 3.9 com uso do *solver* Gurobi versão 9.5, usando os parâmetros padrões e o limite de tempo de 86.400 segundos. A heurística `GraspCC-LB` foi implementada na linguagem ISO/IEC C++17.

Os experimentos consideraram um ambiente heterogêneo composto por máquinas com as seguintes características *(i)* Máquina *Small* com 7,5 GB de RAM e 38,4 GFLOPS de processamento, *(ii)* Máquina *Standard* com 64 GB de RAM e 153,6 GFLOPS, *(iii)* Máquina *Intermediate* com 192 GB de RAM e 76,8 GFLOPS, e *(iv)* Máquina *Large* com 2048 GB de RAM e 153,6 GFLOPS. Todos os tipos de máquina possuem 500 GB de armazenamento. Os custos financeiros para usar cada máquina por 1 segundo são U$ 0,0003, U$ 0,0007, U$ 0,0016 e U$ 0,0081, respectivamente. Foram considerados também os seguintes *workflows*: (i) *1000 Genomes* [Ferreira da Silva et al.,

2019], (ii) *Epigenomics* [Juve et al., 2013] e (iii) *Montage* [Rynge et al., 2014], e os *traces* reais de execução foram obtidos em `https://github.com/wfcommons/pegasus-instances`. A Tabela 1 apresenta as características dos *traces*, onde as colunas representam o nome da instância, os requisitos de memória RAM, armazenamento em disco e processamento em GFLOPs, o tempo máximo de processamento (em minutos) imposto pelo usuário. É importante ressaltar que a Tabela 1 também apresenta a quantidade de *layers* e de *buckets* obtidas pela abordagem proposta, e o conjunto de tipos de máquinas para execução, baseado no ambiente computacional usado nos *traces* reais. Nos experimentos realizados, o custo máximo (C_{max}) foi fixado em U$ 1.000,00.

Tabela 1. Descrição das Instâncias

Instância	*Layers*	*Buckets*	RAM (GB)	Armazenamento em Disco (GB)	Número de GFLOP	Tempo Máx. (min)	Máquinas
1000genome-2ch-100k	3	52	8,40	19,42	106417,72	60	*Large Intermediate*
epigenomics-hep-1seq-50k	9	41	2,17	1,21	20709,38	30	*Intermediate Small*
montage-dss-05d	8	58	0,58	9,25	214495,14	50	*Large Standard*
montage-dss-05d-b1			0,20	3,16	84868,80		
montage-dss-05d-b2	8	20	0,19	3,03	53940,41	50	*Large Standard*
montage-dss-05d-b3			0,19	3,06	75685,63		

A Tabela 2 apresenta os resultados obtidos tanto pelo modelo matemático `CC-LB` (*i.e.*, solução exata do problema), bem como a solução fornecida pela heurística `GraspCC-LB`. A primeira coluna refere-se ao nome da instância. Na coluna `CC-LB`, para cada instância, é reportado o valor do custo financeiro, o tempo em minutos e o tempo computacional obtido pelo modelo matemático. Na coluna `GraspCC-LB`, além das mesmas informações reportadas para o `CC-LB`, é apresentado o *Gap* entre as soluções geradas pela heurística e pelo modelo matemático. Por fim, os pesos α_1 e α_2 atribuem a importância dos objetivos de custo financeiro e de tempo de alocação do recurso, conforme definido pelo usuário, respectivamente. A variação das execuções estão relacionadas em como os pesos afetam os objetivos: *(i)* $\alpha_1 = 0,5$ e $\alpha_2 = 0,5$, considera de maneira igualitária o custo financeiro e tempo de execução; *(ii)* $\alpha_1 = 1$ e $\alpha_2 = 0$, privilegia o custo financeiro; e *(iii)* $\alpha_1 = 0$ e $\alpha_2 = 1$, privilegia o tempo de execução.

A partir dos resultados, é possível observar que o modelo `CC-LB` conseguiu gerar resultados ótimos para 78% das variações de execução. Os resultados marcados com (*) indicam que o modelo não conseguiu atingir a otimalidade dentro do limite de tempo estabelecido (86.400 segundos) para essas instâncias, *i.e.*, não encontrou as soluções ótimas no tempo limite definido. Para duas execuções das instâncias *montage-dss-05d*, o modelo também não conseguiu encontrar uma solução viável (indicado por "-"). Assim, considerando os resultados em que o modelo retornou uma solução ótima, foi realizada uma comparação com os resultados gerados pela heurística. A `GraspCC-LB` alcançou o mesmo resultado ótimo do modelo em 50% dos casos avaliados. Além disso, a `GraspCC-LB` encontrou soluções para instâncias que o modelo não foi capaz de encontrar.

Em relação ao tempo computacional, conforme esperado, o tempo computacional da heurística `GraspCC-LB` foi consideravelmente inferior ao do modelo `CC-LB` em todas as execuções. Observa-se também que a `GraspCC-LB` não atingiu a otimalidade

Tabela 2. Modelo matemático `CC-LB` *versus* Heurística `GraspCC-LB`

Instâncias	CC-LB			GraspCC-LB				α_1	α_2
	Valor da Solução		Tempo Total (s)	Valor da Solução		Tempo Total (s)	*Gap* (%)		
	Custo ($)	Tempo (min)		Custo ($)	Tempo (min)				
	15,13	26	2.728,13	15,13	26	4,91	0	0,5	0,5
1000genome-2ch-100k	4,99	60	173,05	4,99	52	6,45	0	1	0
	15,13	26	3.686,17	15,13	26	5,27	0	0	1
	1,99	25	357,32	1,99	25	3,94	0	0,5	0,5
epigenomics-hep-1seq-50k	1,59*	30*	86.400,00	1,59	30	4,82	0	1	0
	2,57	25	162,65	2,30	25	4.02	0	0	1
	-	-	86.400,00	17,44	45	6,16	-	0,5	0,5
montage-dss-05d	15,22*	50*	86.400,00	8,56	65	5,76	0	1	0
	-	-	86.400,00	20,11	45	6,03	-	0	1
	6,06	17	34,53	5,17	19	1,94	10,15 ↑	0,5	0,5
montage-dss-05d-b1	1,17	50	9,90	2,95	24	2,02	60,16 ↑	1	0
	8,28	17	24,80	7,83	19	1,94	10,52 ↑	0	1
	5,00	15	67,76	5,00	15	1,93	0	0,5	0,5
montage-dss-05d-b2	1,00	50	8,82	2,78	20	1,97	63,79 ↑	1	0
	6,33	15	37,97	5,00	15	1,93	0	0	1
	5,57	17	41,67	5,13	18	1,94	5,35 ↑	0,5	0,5
montage-dss-05d-b3	1,13	50	9,65	2,91	23	1,94	61,03 ↑	1	0
	7,83	15	39,51	7,79	18	1,96	5,56 ↑	0	1

em 7 das 12 variações do *workflow* Montage. Uma conclusão obtida foi que a heurística `GraspCC-LB` apresentou desempenho melhor em instâncias maiores do que nas pequenas. Além disso, o Montage apresenta uma fase custo de *Reduce*, que não é observada nos demais *workflows*.

6. Considerações Finais

Dimensionar recursos computacionais para a execução do *workflow* em ambientes HPC de larga escala pode não ser uma tarefa trivial, dada a complexidade da estrutura do *workflow* e a heterogeneidade do ambiente HPC. Este artigo aborda o dimensionamento adequado de recursos computacionais para a execução eficiente de *workflows* em ambientes HPC, destacando o impacto de sub ou superdimensionamento dos recursos. A heurística `GraspCC-LB` é proposta com o objetivo de minimizar custo e tempo de processamento, respeitando as restrições impostas pelo usuário, pelo ambiente HPC (*e.g.*, quantidade máxima de máquinas a serem reservadas) e a própria estrutura do *workflow*. Diferentemente dos trabalhos relacionados, a `GraspCC-LB` utiliza uma abordagem *Layered-Bucket* que explora agrupamentos de tarefas por nível do *workflow*, visando encontrar o melhor dimensionamento possível. A `GraspCC-LB` é avaliada utilizando *traces* de *workflows* reais e comparada com o modelo matemático `CC-LB` (*i.e.*, solução exata do problema). Experimentos mostraram que a `GraspCC-LB` gerou a solução ótima em 50% dos cenários avaliados e encontrou soluções para instâncias que o modelo não foi capaz de encontrar dentro do prazo estipulado (*i.e.*, 24 horas). Além disso, o tempo computacional da heurística `GraspCC-LB` foi consideravelmente menor que o modelo `CC-LB` em todas as execuções. Como trabalho futuro, pretende-se avaliar o impacto do dimensionamento gerado pela heurística `GraspCC-LB` em um ambiente real e comparar os resultados com adaptações de outros trabalhos da literatura.

Referências

Abdi, S., Pourkarimi, L., Ahmadi, M., and Zargari, F. (2018). Cost minimization for bag-of-tasks workflows in a federation of clouds. *J. Supercomput.*, 74(6):2801–2822.

Coutinho, R. et al. (2013). Optimization of a cloud resource management problem from a consumer perspective. In *Euro-Par 2013*, volume 8374 of *LNCS*, pages 218–227. Springer.

Coutinho, R. et al. (2015). Optimizing virtual machine allocation for parallel scientific workflows in federated clouds. *FGCS*, 46:51–68.

Coutinho, R. et al. (2016). A dynamic cloud dimensioning approach for parallel scientific workflows: a case study in the comparative genomics domain. *J. Grid Comput.*, 14(3):443–461.

de Oliveira, D. C. M., Liu, J., and Pacitti, E. (2019). *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Deelman, E. et al. (2018). The future of scientific workflows. *Int. J. High Perform. Comput. Appl.*, 32(1):159–175.

Deldari, A., Naghibzadeh, M., and Abrishami, S. (2017). Cca: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud. *J. of Supercomp.*, 73(2):756–781.

Ferreira da Silva, R. et al. (2019). Using simple pid-inspired controllers for online resilient resource management of distributed scientific workflows. *FGCS*, 95:615–628.

Gil, Y. et al. (2007). On the black art of designing computational workflows. In *Proc.s of the WORKS*, page 53–62, New York, NY, USA.

Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., and Vahi, K. (2013). Characterizing and profiling scientific workflows. *FGCS*, 29(3):682–692.

Lin, B., Guo, W., Xiong, N., Chen, G., Vasilakos, A., and Zhang, H. (2016). A pretreatment workflow scheduling approach for big data applications in multicloud environments. *IEEE Transactions on Network and Service Management*, 13(3):581–594.

Liu, J., Pacitti, E., Valduriez, P., de Oliveira, D., and Mattoso, M. (2016). Multi-objective scheduling of scientific workflows in multisite clouds. *FGCS*, 63:76–95.

Malawski, M. et al. (2015). Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization. *Scientific Programming*, 2015:5.

Mohammadi, S., Pedram, H., and PourKarimi, L. (2018). Integer linear programming-based cost optimization for scheduling scientific workflows in multi-cloud environments. *The Journal of Supercomputing*, 74:4717–4745.

Moschakis, I. and Karatza, H. (2015). Multi-criteria scheduling of bag-of-tasks applications on heterogeneous interlinked clouds with simulated annealing. *Journal of Systems and Software*, 101:1–14.

Ogasawara, E. S. et al. (2011). An algebraic approach for data-centric scientific workflows. *VLDB*, 4(12):1328–1339.

Rosa, M. J. et al. (2021). Computational resource and cost prediction service for scientific workflows in federated clouds. *FGCS*, 125:844–858.

Rynge, M. et al. (2014). Producing an infrared multiwavelength galactic plane atlas using montage, pegasus, and amazon web services. *Astronomical Data Analysis Software and Systems XXIII*, 485:211.

Song, A., Chen, W.-N., Luo, X., Zhan, Z.-H., and Zhang, J. (2020). Scheduling workflows with composite tasks: A nested particle swarm optimization approach. *IEEE Transactions on Services Computing*, 15(2):1074–1088.

Zhou, J., Wang, T., Cong, P., Lu, P., Wei, T., and Chen, M. (2019). Cost and makespan-aware workflow scheduling in hybrid clouds. *Journal of Systems Architecture*, 100:101631.