CEFET/RJ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO

DIMENSIONAMENTO DE RECURSOS PARA EXECUÇÃO DE *WORKFLOWS* CIENTÍFICOS EM AMBIENTES DE COMPUTAÇÃO DE ALTO DESEMPENHO

Luis Carlos Ramos Alvarenga

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação, Centro Federal de Educação Tecnológica Celso Suckow da Fonseca CEFET/RJ, como parte dos requisitos necessários à obtenção do grau de mestre.

Orientadores:
Rafaelli de Carvalho Coutinho
Daniel Cardoso de Moraes de Oliveira

Rio de Janeiro,
Outubro de 2024

Dimensionamento de recursos para execução de *workflows* científicos em ambientes de computação de alto desempenho

Dissertação de Mestrado em Ciência da Computação, Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ.

Luis Carlos Ramos Alvarenga

Aprovada por:

Presidente, Profa. Rafaelli de Carvalho Coutinho, D.Sc. (orientadora)

Daniel Cardoso de Moraes de Oliveira, D.Sc. (coorientador)

Marta Lima de Queirós Mattoso, D.Sc.

Pedro Henrique González Silva, D.Sc.

Yuri Abitbol de Menezes Frota, D.Sc.

Rio de Janeiro,

Outubro de 2024

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

A473 Alvarenga, Luis Carlos Ramos
Dimensionamento de recursos para execução de workflows científicos em ambientes de computação de alto desempenho / Luis Carlos Ramos Alvarenga. — 2024.
41f. : il. color. , enc.

Dissertação (Mestrado) Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, 2024.
Bibliografia : f. 37-41
Orientador: Rafaelli de Carvalho Coutinho
Coorientador: Daniel Cardoso de Moraes de Oliveira

1. Computação de alto desempenho. 2. Programação (Computadores). 3. Computação em nuvem. 4. Gerenciamento de recursos de informação. I. Coutinho, Rafaelli de Carvalho. (Orient.). II. Oliveira, Daniel Cardoso de Moraes de. (Coorient.). III. Título.

CDD 004.068

Elaborada pela bibliotecária Tania Mello – CRB/7 nº 5507/04

DEDICATÓRIA

A minha família.

AGRADECIMENTOS

Deus.

A minha esposa, Alana, por seu amor, incentivo e apoio inestimável.

Aos meus orientadores Rafaelli Coutinho e Daniel de Oliveira, pelos ensinamentos e suporte que foram essenciais na condução deste trabalho.

Ao professor Yuri Frota, por compartilhar seu conhecimento e experiência.

RESUMO

Dimensionamento de recursos para execução de *workflows* científicos em ambientes de computação de alto desempenho

Luis Carlos Ramos Alvarenga

Orientadores:
Rafaelli de Carvalho Coutinho
Daniel Cardoso de Moraes de Oliveira

Resumo da Dissertação submetida ao Programa de Pós-graduação em Ciência da Computação do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca CEFET/RJ como parte dos requisitos necessários à obtenção do grau de mestre.

Cientistas cada vez mais necessitam executar experimentos altamente exigentes em termos computacionais. Estes experimentos são frequentemente modelados como *workflows* científicos e são executados em ambientes de computação de alto desempenho (HPC). Tipicamente estes ambientes fornecem uma grande variedade de recursos aos usuários. O dimensionamento adequado dos recursos para a execução de *workflows* científicos nestes ambientes é uma tarefa crucial. Um ambiente sub ou superdimensionado pode afetar diretamente o desempenho do experimento, gerando impactos negativos relacionados ao tempo e ao custo financeiro da execução. Com isso, pesquisas envolvendo estimação de recursos para execução de experimentos em ambientes de HPC vêm sendo realizadas, como a heurística *GraspCC*, que usa o procedimento de busca adaptativa randomizada gulosa (GRASP). O objetivo deste estudo é investigar a execução de *workflows* científicos em ambientes de alto desempenho, tais como *clusters* e nuvens de computadores, com o intuito de estimar eficientemente os recursos necessários considerando o tempo e os custos financeiros associados. O problema foi definido a partir da proposta de estruturação do *workflow* em estágios ou níveis paralelos de tarefas semelhantes entre si e que operam de maneira independente, chamada *Layered-Bucket* (LB). Para resolver o problema, foram propostas uma formulação de programação matemática inteira e uma adaptação da heurística *GraspCC* para considerar a abordagem LB, chamada *GraspCC-LB*. A abordagem proposta foi avaliada utilizando dados históricos de *workflows* das áreas de bioinformática e astronomia. As estimativas de recursos produzidas pela *GraspCC-LB* foram comparadas com o uso real de recursos em um ambiente HPC do mundo real para avaliar sua eficácia. Os resultados mostram a eficácia do *GraspCC-LB* como uma abordagem robusta para otimização de recursos no contexto de *workflows* científicos de grande escala que requerem capacidades HPC, apresentando-se como uma importante ferramenta de apoio à tomada de decisões.

Palavras-chave:
Workflows científicos, Computação de Alto Desempenho, Computação em nuvem, Otimização, Sistemas de suporte à decisão.

Rio de Janeiro,

Outubro de 2024

ABSTRACT

Dimensionamento de recursos para execução de *workflows* científicos em ambientes de computação de alto desempenho

Luis Carlos Ramos Alvarenga

Advisors:
Rafaelli de Carvalho Coutinho
Daniel Cardoso de Moraes de Oliveira

Abstract of dissertation submitted to Programa de Pós-graduação em Ciência da Computação - Centro Federal de Educação Tecnológica Celso Suckow da Fonseca CEFET/RJ as partial fulfillment of the requirements for the degree of master.

Scientists increasingly need to run highly computationally demanding experiments. These experiments, often modeled as scientific workflows, are run in high-performance computing (HPC) environments. Typically, these environments provide a wide range of resources to users. The proper sizing of resources for running scientific workflows in these environments is a crucial task. An undersized or oversized environment can directly affect the performance of the experiment, leading to negative impacts on the time and financial cost of execution. Thus, research involving resource estimation for experiment execution in HPC environments has been conducted, such as the *GraspCC* heuristic, which uses the adaptive randomized greedy search procedure (GRASP). The objective of this work is to investigate the execution of scientific workflows in high-performance environments, such as clusters and computer clouds, in order to efficiently estimate the required resources considering the associated time and financial costs. The problem was defined from the proposed structuring of workflow into parallel stages or levels of tasks that are similar to each other and operate independently, called Layered-Bucket (LB). To solve the problem, we proposed an integer mathematical programming formulation and an adaptation of the *GraspCC* heuristic to accommodate the LB approach, named *GraspCC-LB*. The proposed approach was evaluated using real traces of workflows from the fields of bioinformatics and astronomy. The resource estimations produced by *GraspCC-LB* were compared against the actual resource usage in a real-world HPC environment to assess its effectiveness. The results demonstrate the effectiveness of *GraspCC-LB* as a robust approach for resource optimization in the context of large-scale scientific workflows that require HPC capabilities, serving as an important decision-support tool.

Key-words:
Scientific Workflow, High-Performance Computing, Cloud computing, Optimization, Decision support systems.

Rio de Janeiro,

Outubro de 2024

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ANN	*Artificial Neural Networks*	21
BOT	*Bag-of-Tasks*	19, 20, 22, 23, 34
CRCPS	*Computational Resource and Cost Prediction Service*	21
DRPM	*Dynamic Resource Provisioning and Management*	21
GFLOP	*GigaFLOP*	24, 25
GRASP	*Greedy Randomized Adaptive Search Procedure*	13, 19, 20, 21, 28
HPC	*High-Performance Computing*	12, 13, 16, 17, 35, 47
ILP	*Integer Linear Programming*	20
LB	*Layered-Bucket*	13, 24
MILP	*Mixed Integer Linear Programming*	20
MLR	*Multiple Linear Regression*	21
NIST	*National Institute of Standards and Technology*	17
NSGA-II	*Non-dominated Sorting Genetic Algorithm*	21
PSFS	*Pareto-efficient Stepwise Frequency Selection*	20
SGWFCS	Sistemas De Gerenciamento De *Workflows* Científicos	15
VL2	*Virtual Layer 2*	21
WSCAD	Simpósio Em Sistemas Computacionais De Alto Desempenho	47

Capítulo I 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, do inglês *High-Performance Computing* (HPC), 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 II.1, e que podem (*a priori*) ser executadas em paralelo.

Usualmente, mas não necessariamente, os *workflows* são executados em ambientes de HPC. Esses ambientes incluem *clusters* de computadores, supercomputadores e ambientes de computação em nuvem [Alkhanak et al., 2016]. Pesquisas envolvendo *workflows* têm sido amplamente realizadas em ambientes de *cluster* [Geist and Reed, 2017] e em nuvem na última década [Adhikari et al., 2019]. No entanto, com as mudanças constantes na oferta dos recursos de nuvem, ainda existe a necessidade de desenvolvimento de novas abordagens para lidar com desafios e restrições específicas desses ambientes, especialmente relacionadas a gerenciamento e estimativa de uso de recursos.

Neste contexto, a gerência eficaz do uso de recursos por *workflows*, cujo objetivo é a otimização das estimativas e dimensionamento de recursos, representa um desafio significativo, pois pode determinar o sucesso ou o fracasso na execução de um *workflow* complexo e de alto custo computacional [Coutinho et al., 2015]. Estes aspectos são primordiais, uma vez que o desempenho geral do *workflow* e os custos financeiros associados podem ser adversamente afetados se o ambiente estiver subdimensionado (resultando em gargalos de desempenho de tempo) ou superdimensionado (levando ao uso ineficiente de recursos e custos financeiros desnecessários).

Com isso, pesquisas envolvendo estimativas de recursos para execução de experimentos modelados como *workflows* em ambientes de HPC vem sendo realizadas [Coutinho et al., 2015; Moschakis and Karatza, 2015; Durillo 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]. Um exemplo é a heurística *GraspCC* [Coutinho et al., 2015], que usa o procedimento de busca adaptativa randomizada gulosa, do inglês *Greedy Randomized Adaptive Search Procedure* (GRASP), para dimensionar a quantidade de máquinas virtuais a ser instanciada para execução de uma aplicação visando reduzir custos e tempo em nuvens computacionais.

O objetivo deste trabalho é investigar a execução de *workflows* em ambientes HPC (*e.g. clusters* e nuvens de computadores) para estimar de forma eficiente os recursos necessários considerando as dependências das tarefas, o tempo e os custos financeiros associados. O problema foi definido a partir da proposta de estruturação do *workflow* em estágios de tarefas que podem ser executadas de maneira independente e foi chamada de *Layered-Bucket* (LB).

Para resolver o problema, propôs-se um modelo de programação matemática inteira e uma heurística, chamada de *GraspCC-LB*. Na solução, os estágios de processamento do *workflow* são categorizados como *layers*. Esses agrupam tarefas prontas para execução que são independentes entre si, permitindo sua execução de forma paralela, e são chamados de *buckets*. No que se refere ao processamento, os *buckets* de um *layer* são processados de forma paralela, enquanto os *layers* do *workflow* são executados de forma sequencial. Os resultados indicam que *GraspCC-LB* pode se configurar como uma ferramenta de apoio à tomada de decisões importante para execução de *workflows* em ambientes de computação de alto desempenho.

Portanto, as principais contribuições deste trabalho são: *(i)* definição do problema de dimensionamento de recursos para execução em nuvem de *workflows*; *(ii)* modelagem do problema como um modelo de programação matemática inteira que considera os múltiplos níveis do *workflow* para o dimensionamento; *(iii)* Uma heurística, chamada *GraspCC-LB*, capaz de estimar o dimensionamento de recursos para execução de workflows em ambientes de alto desempenho, permitindo ao usuário definir critérios para alocação adequada de recursos, conforme estimativas fornecidas; e *(iv)* avaliação dos métodos propostos para demonstrar a aplicabilidade e utilidade em diferentes contextos práticos.

O restante deste documento está estruturado da seguinte forma. No Capítulo II, é introduzido o referencial teórico, que elucida o conceito de *workflows* científicos e descreve os ambientes de computação de alto desempenho, que incluem *clusters* e computação em nuvem. No Capítulo III, são apresentados os trabalhos relacionados agrupados de acordo com os tipos de processamento empregados. No Capítulo IV, é apresentado o modelo matemático proposto para resolver o problema de dimensionamento de recursos. No Capítulo V é descrita a adaptação da heurística GRASP para uso neste trabalho. No Capítulo VI, são relatados os resultados experimentais obtidos com a aplicação do modelo matemático e da heurística propostos. Finalmente, no Capítulo VII, são apresentados as conclusões obtidas e as possíveis direções para trabalhos futuros.

Capítulo II Referencial Teórico

Neste capítulo, são apresentados os conceitos fundamentais para o entendimento do trabalho. A Seção II.1 conceitua o modelo de *workflows*. A Seção II.2 introduz a computação de alto desempenho, descrevendo seus principais sistemas: *clusters* e nuvens de computadores.

II.1 Modelo de *workflow*

O conceito de *workflow* refere-se a uma especificação de alto nível que descreve um conjunto de tarefas e suas respectivas dependências, as quais devem ser satisfeitas para alcançar um objetivo específico dentro de um processo, conforme discutido por [Deelman et al., 2009]. O *workflow* é uma ferramenta essencial para a gestão de processos complexos em diversas áreas, como ciência, engenharia e negócios, pois permite a coordenação eficiente de tarefas e recursos.

Um *workflow* é representado por $W(T, E)$, onde T é um **conjunto de tarefas** denotado por $T = \{T_1, T_2, T_3, ..., T_k\}$, e E é um conjunto de arestas direcionadas [Baskiyar and Abdel-Kader, 2010]. Esses parâmetros explicitam a relação entre as tarefas e a sequência em que elas devem ser executadas.

A aresta $E_{gh}(T_g, T_h)$ indica a **dependência entre as tarefas** T_g e T_h, onde $(T_g, T_h) \in T$. Esta relação de dependência é crucial para garantir a execução correta das tarefas. A tarefa T_g é considerada predecessora da tarefa T_h, enquanto T_h é a sucessora de T_g. Segundo Russell et al. [2005], uma tarefa de entrada é aquela que não possui predecessores, e as tarefas de saída são aquelas que não têm sucessores.

Esta organização de tarefas destaca a importância de garantir que as tarefas predecessoras sejam sempre concluídas antes do início das tarefas sucessoras, a fim de evitar problemas e inconsistências durante o processo. Um exemplo de representação visual de um *workflow* pode ser observado na Figura II.1, onde podemos observar: *(i)* a tarefa de entrada T_1; *(ii)* a tarefa de saída T_8; *(iii)* tarefas sucessoras, tais como $\{T_2, T_3, T_4\}$ que sucedem T_1; e *(iv)* predecessoras, como a tarefa T_7 que precede T_8.

Para melhorar a eficiência na execução do *workflow*, as tarefas podem ser divididas em **níveis**, permitindo que sejam executadas em paralelo, dado que não apresentam dependências entre si. Por exemplo, na Figura II.1, temos os seguintes níveis de tarefas: **nível 1** composto pela tarefa T_1; **nível**

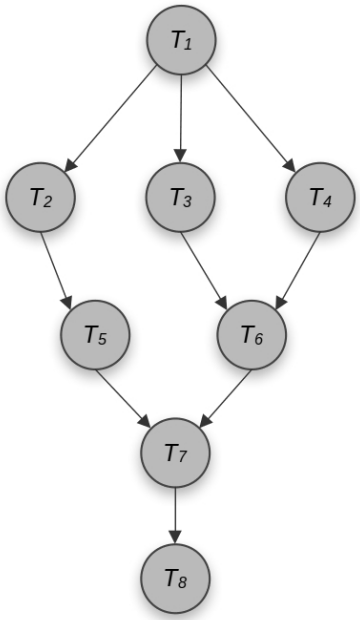


Figura II.1: Representação visual de um *workflow*

2 formado pelas tarefas T_2, T_3, T_4; **nível 3** por T_5, T_6; **nível 4** por T_7 e, por fim, o **nível 5** pela tarefa T_8. Esta abordagem em níveis pode facilitar o dimensionamento de recursos e a distribuição de carga de trabalho, melhorando o desempenho geral do processo.

O termo *workflow* científico é usado para descrever *workflows* que compartilham as características de manipulação de grandes volumes de dados e de demanda por alto poder de processamento computacional [Deelman et al., 2009]. Os *workflow* podem ser classificados quanto ao relacionamento entre suas tarefas, sendo: *(i)* aplicações *Bag-of-Tasks* (BoT), que possuem como característica a independência entre tarefas, permite execução em paralelo. Isso potencializa a escalabilidade e a distribuição das tarefas em ambientes de computação distribuídos de alto desempenho, como os *clusters*, *grids* e nuvens de computadores. Essa estratégia aproveita efetivamente a capacidade de processamento disponibilizada pelos provedores de serviço e pode acelerar a execução das tarefas; e *(ii)* aplicações *pipelines*, que é uma representação sequencial de tarefas interdependentes, onde a saída de uma tarefa é usada como entrada para a próxima, formando uma cadeia de processamento contínuo. Esse tipo de *workflow* é comum em aplicações científicas e de processamento de dados, onde o fluxo de informações segue uma ordem específica e a paralelização das tarefas é limitada pelas dependências entre elas [Yu and Buyya, 2005; Deelman et al., 2009].

Devido à complexidade do dimensionamento de recursos para o processamento de *workflows* científicos, normalmente são utilizadas ferramentas que automatizam e coordenam esse processos, os chamados Sistemas de Gerenciamento de *Workflows* Científicos (SGWfCs). Essas ferramentas permitem o planejamento, programação e execução das tarefas do *workflow* de maneira eficiente e consistente. Atualmente, uma ampla gama de SGWfCs disponibiliza suporte para execução de

workflows em ambientes de *clusters* e nuvens computacionais. A ênfase desses sistemas geralmente recai sobre a otimização de políticas de escalonamento de tarefas e alocação de dados [Ogasawara, 2011]. Comumente, essas execuções são compostas por centenas, ou até milhares, de tarefas que consomem e produzem quantidade igualmente grande de arquivos. Assim, para executar esses *workflows* é necessário o uso de técnicas de paralelismo e capacidade de processamento de alto desempenho.

II.2 Computação de Alto Desempenho

A computação de alto desempenho (HPC) é uma área especializada da ciência da computação que se concentra na otimização do desempenho de sistemas de computação para resolver problemas complexos e de larga escala. Esses problemas geralmente envolvem a análise e processamento de grandes volumes de dados e requerem capacidades computacionais significativas para serem resolvidos em tempo hábil. Os sistemas HPC são projetados para serem altamente eficientes, utilizando múltiplos processadores, memória distribuída, rede de alta velocidade e sistemas de armazenamento de dados de larga escala[Buyya, 1999]. A otimização do desempenho de sistemas HPC também envolve a análise e a melhoria de aspectos relacionados à eficiência energética, confiabilidade e tolerância a falhas. O gerenciamento e a redução do consumo de energia são questões importantes, especialmente considerando a crescente preocupação com a sustentabilidade e os custos operacionais em centros de dados [Jia et al., 2023].

Os sistemas HPC incluem *clusters* de computadores, supercomputadores e redes de computadores distribuídos [Foster and Kesselman, 2003]. A computação em nuvem também desempenha um papel importante na área de HPC, fornecendo recursos computacionais sob demanda e escalabilidade para aplicações de alto desempenho, sem a necessidade de investimento em infraestrutura local, reduzindo os custos associados e permitindo maior flexibilidade na alocação de recursos [Vaquero et al., 2009]. São utilizados em diversas áreas, incluindo pesquisa científica, engenharia, finanças e tecnologia, e permitem a realização de simulações complexas, análise de dados em larga escala, criação de modelos computacionais e outras aplicações que exigem grande poder de processamento. Alguns exemplos incluem simulações climáticas, sequenciamento genômico, previsões meteorológicas, análise de dados financeiros e inteligência artificial [Shalf et al., 2011].

O processamento de *workflows* científicos em sistemas HPC oferece diversas vantagens, como escalabilidade, paralelismo, otimização de recursos e tolerância a falhas [Oinn et al., 2006]. A escalabilidade permite que os *workflows* sejam executados em unidades computacionais com diferentes capacidades de processamento, adaptando-se às demandas computacionais, e o paralelismo possibilita a execução simultânea de múltiplas tarefas do *workflow*, reduzindo o tempo de execução e melhorando a eficiência [Geist and Reed, 2017].

II.2.1 *Clusters*

O uso de *clusters* em HPC tem se tornado cada vez mais comum nos últimos anos, devido ao crescente aumento da demanda por processamento de dados em larga escala. Um *cluster* é uma coleção de computadores interconectados, chamados de nós, que trabalham de forma colaborativa para realizar tarefas computacionais intensivas, proporcionando grande poder de processamento para aplicações científicas e comerciais [Dongarra et al., 2003].

Os *clusters* apresentam alguns benefícios, como escalabilidade, custo-benefício e alta disponibilidade. A escalabilidade permite que os *clusters* sejam facilmente expandidos para aumentar a capacidade de processamento conforme a demanda. A relação custo-benefício é uma vantagem dos *clusters*, pois o uso de *hardware* comercialmente disponível geralmente resulta em menor custo em comparação com soluções proprietárias [Geist and Reed, 2017]. Além disso, a alta disponibilidade é alcançada por meio de redundâncias e tolerância a falhas nos componentes do sistema, garantindo a continuidade das operações em caso de falhas de *hardware* [Jia et al., 2023].

Entretanto, *clusters* também enfrentam desafios, como a complexidade de gerenciamento, balanceamento de carga e comunicação entre os nós. O gerenciamento de *clusters* envolve a coordenação de recursos, monitoramento de desempenho e otimização do uso de energia [Zakarya and Gillam, 2017]. O balanceamento de carga é necessário para distribuir adequadamente as tarefas entre os nós e evitar gargalos de desempenho [Geist and Reed, 2017]. Por fim, a comunicação entre os nós é essencial para garantir a colaboração eficiente e a troca de informações durante a execução das tarefas, o que pode ser alcançado por meio de protocolos e algoritmos de comunicação específicos [Hoefler et al., 2011].

II.2.2 Nuvens Computacionais

Segundo o *National Institute of Standards and Technology* (NIST), a computação em nuvem é um modelo que permite o acesso ubíquo, conveniente e sob demanda a um *pool* de recursos computacionais configuráveis (como redes, servidores, armazenamento, aplicativos e serviços) por meio de uma rede. Esses recursos podem ser provisionados e liberados rapidamente, com um esforço mínimo de gerenciamento ou interação com o provedor de serviços. Este modelo de computação em nuvem é constituído por cinco características essenciais, três modelos de serviço e quatro modelos de implantação [Mell and Grance, 2011].

As cinco características essenciais da computação em nuvem são: *(i)* **Serviço sob demanda**, em que os consumidores podem solicitar unilateralmente recursos de computação conforme necessário, sem a necessidade de interação humana com os provedores de serviço; *(ii)* **Acesso à rede de amplo alcance**, os recursos estão disponíveis através da rede e podem ser acessados por meio de plataformas padronizadas; *(iii)* **Agrupamento de recursos**, os recursos de computação do provedor

são agrupados para atender a vários consumidores, com diferentes recursos físicos e virtuais dinamicamente atribuídos e reatribuídos de acordo com a demanda do consumidor; *(iv)* **Elasticidade rápida**, as capacidades podem ser rapidamente provisionadas e liberadas para escalar rapidamente para dentro e para fora de acordo com a demanda; e (v) **Serviço medido**, a utilização de recursos pode ser monitorada, controlada e relatada, proporcionando transparência tanto para o provedor quanto para o consumidor do serviço utilizado [Mell and Grance, 2011].

Conforme [Vaquero et al., 2009], os modelos de serviço da computação em nuvem podem ser divididos em *(i)* ***Software* como um Serviço (SaaS)**, o consumidor usa os aplicativos do provedor que estão sendo executados em uma infraestrutura em nuvem, podemos citar como exemplo o *Google Drive* e Salesforce; *(ii)* **Plataforma como um Serviço (PaaS)**, o consumidor implanta aplicativos criados ou adquiridos em linguagens de programação e ferramentas suportadas pelo provedor na infraestrutura em nuvem, por exemplo o Microsoft Azure; e *(iii)* **Infraestrutura como um Serviço (IaaS)**, o consumidor provisiona o processamento, armazenamento, redes e outros recursos fundamentais de computação, onde o consumidor é capaz de implantar e executar *software* arbitrário, que pode incluir sistemas operacionais e aplicativos, provedores de serviço, como *Amazon EC2*.

Finalmente, conforme definido por [Mell and Grance, 2011], os modelos de implantação da computação em nuvem são: *(i)* **Nuvem Privada**, a infraestrutura da nuvem é provisionada para uso exclusivo por uma única organização composta por vários consumidores; *(ii)* **Nuvem Comunitária**, a infraestrutura da nuvem é provisionada para uso exclusivo por uma comunidade específica de consumidores de organizações que têm interesses comuns; *(iii)* **Nuvem Pública**, tais como *Google Cloud Platform* e *Amazon Web Services*, a infraestrutura da nuvem é provisionada para uso aberto pelo público em geral; e *(iv)* **Nuvem Híbrida**, a infraestrutura da nuvem é uma composição de duas ou mais infraestruturas de nuvem distintas (privada, comunitária ou pública) que permanecem entidades únicas, mas estão vinculadas por tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicativos.

Capítulo III Trabalhos Relacionados

Diversos trabalhos encontrados na literatura relacionada fornecem soluções para gerenciar recursos para execução de uma aplicação em ambientes computacionais [Coutinho et al., 2013, 2015; Abdi et al., 2018; Oda et al., 2018; Pietri and Sakellariou, 2019; Ramamurthy et al., 2020; Xu et al., 2020; Rosa et al., 2021; Shealy et al., 2021]. A Tabela III.1 apresenta alguns importantes trabalhos que lidam especificamente com o problema de dimensionamento de recursos computacionais para execução de uma aplicação e propõem métodos heurísticos e/ou exatos para resolvê-lo. Para cada trabalho, são explicitados os autores, o ano de publicação, o tipo de aplicação (BoT ou *pipeline*), e, por fim, as qualifica quanto ao método, heurístico ou exato, empregado na solução proposta. Esses trabalhos são descritos com mais detalhes a seguir.

Tabela III.1: Comparativo dos Trabalhos Relacionados

Autor	**Ano**	**Aplicação**	**Método**	
			Heurística	**Exato**
Coutinho et al.	2013	BoT	Sim	Sim
Coutinho et al.	2015	*pipeline*	Sim	Sim
Abdi et al.	2018	BoT	Sim	Sim
Mohammadi et al.	2018	*pipeline*	Não	Sim
Oda et al.	2018	*pipeline*	Sim	Não
Pietri and Sakellariou	2019	*pipeline*	Sim	Não
Ramamurthy et al.	2020	*pipeline*	Sim	Não
Xu et al.	2020	*pipeline*	Sim	Não
Rosa et al.	2021	*pipeline*	Sim	Não
Shealy et al.	2021	*pipeline*	Não	Sim
de la Torre and Halappanavar	2023	*pipeline*	Não	Sim

Coutinho et al. [2013] propõem uma solução para o dimensionamento de recursos em nuvem sob a perspectiva do consumidor, visando reduzir custos financeiro e tempo de execução de aplicações *Bag-of-Tasks* (BoT). A solução envolve uma formulação de programação inteira e uma meta-heurística baseada no GRASP. O problema é modelado para selecionar pacotes de máquinas virtuais que atendam às demandas de processamento, armazenamento e memória dentro de restrições de custo

e tempo. A meta-heurística GRASP mostrou-se eficiente ao encontrar soluções eficazes em menos tempo comparado ao método exato. Entretanto, sendo um problema NP-difícil, o método exato não é viável para grandes instâncias, e a heurística, apesar de mais rápida, não garante soluções ótimas.

Coutinho et al. [2015] propuseram um modelo matemático combinado com uma abordagem heurística para o dimensionamento de recursos para execução de *workflows* em nuvens de computadores, buscando minimizar tanto os custos quanto o tempo de execução dos objetivos, utilizando uma estratégia de agregação. O modelo matemático utilizado consiste em uma formulação de programação inteira linear, do inglês *Integer Linear Programming* (ILP), que leva em consideração várias restrições, como a capacidade dos recursos, custo financeiro e o prazo máximo de conclusão do *workflow*. A abordagem heurística de busca adaptativa, randômica e gulosa, baseada no algoritmo GRASP, consiste em dividir o conjunto de recursos em vários subconjuntos e executar o dimensionamento em cada subconjunto, a fim de encontrar uma solução ótima global.

Abdi et al. [2018] investigaram o escalonamento de aplicações BoT em ambientes de nuvens de computadores, desenvolvendo modelos matemáticos e uma heurística que otimizam o custo, levando em consideração uma restrição de prazo. O modelo matemático utilizado consiste em uma formulação de *Mixed Integer Linear Programming* (MILP), programação linear inteira mista, que considera várias restrições, como a capacidade dos recursos e o prazo máximo de conclusão de aplicações BoT. O estudo implementa, também, uma abordagem de procedimento GRASP, chamada GRASP-FC, para o dimensionamento de recursos para aplicações BoT em nuvens de computadores.

Mohammadi et al. [2018] abordaram o dimensionamento de recursos para *workflow* científicos em ambientes de nuvens de computadores, visando minimizar custos financeiros sob restrições de prazo. Foram propostos modelos de ILP, o que permitiu a análise abrangente do sistema, levando em conta preferências secundárias e análises pós-otimização. A avaliação do custo ótimo foi realizada em função das variações de prazo e tamanho do *workflow*. No entanto, a solução não busca a minimização do tempo de execução do *workflow*.

Pietri and Sakellariou [2019] propõem uma abordagem baseada em Pareto para o dimensionamento de recursos para *workflow* científicos executados em nuvens. O objetivo é selecionar configurações de frequência de CPU que equilibrem o custo e o tempo de execução, considerando as diferentes frequências e preços oferecidos pelos provedores de nuvem. A solução envolve o uso do algoritmo *Pareto-efficient Stepwise Frequency Selection* (PSFS), que inicialmente constrói um conjunto de soluções de Pareto eficientes e, em seguida, melhora essas soluções iterativamente. A abordagem foi avaliada com quatro aplicações reais de *workflow* científicos, demonstrando que é capaz de gerar soluções eficientes em termos de custo e tempo de execução. Uma das limitações mencionadas é o aumento do tempo de execução em aplicações sensíveis à frequência de CPU, es-

pecialmente em cenários com preços lineares e *workflow* intensivos em CPU.

Ramamurthy et al. [2020] propõem um modelo de otimização multiobjetivo para o dimensionamento de recursos para execução de *workflow* científicos sob incertezas. O objetivo é minimizar simultaneamente o custo financeiro de aquisição de máquinas virtuais e o tempo de execução, utilizando a técnica de programação *chance-constrained* para lidar com incertezas nos requisitos dos usuários. O modelo é resolvido utilizando o algoritmo evolutivo *Non-dominated Sorting Genetic Algorithm* (NSGA-II), capaz de gerar soluções aproximadas na fronteira de Pareto. Experimentos demonstraram que a abordagem lida de forma eficiente com incertezas e oferece flexibilidade para ajustar a confiabilidade das soluções. No entanto, o método pode resultar em soluções mais conservadoras à medida que se aumenta a probabilidade de satisfação das restrições, o que reduz a eficiência geral.

Xu et al. [2020] propõem um método de dimensionamento dinâmico de recursos com tolerância a falhas para *workflow* meteorológicos intensivos em dados em nuvens. O método, denominado *Dynamic Resource Provisioning and Management* (DRPM), utiliza a topologia de rede *Virtual Layer 2* (VL2) para construir a infraestrutura de nuvem e o NSGA-II para otimizar o tempo de execução e o balanceamento de carga dos nós de computação. Além disso, técnicas de *checkpoint* e *restart* são empregadas para garantir a recuperação rápida de tarefas com falha. O modelo demonstrou reduzir significativamente o impacto de falhas de nós, otimizando o tempo de execução e melhorando o balanceamento de carga. No entanto, a abordagem requer uma configuração complexa de infraestrutura e apresenta um custo computacional elevado, especialmente para grandes fluxos de trabalho meteorológicos.

Rosa et al. [2021] propuseram o *Computational Resource and Cost Prediction Service* (CRCPs), que é um serviço que prevê e dimensiona recursos computacionais e custos financeiros para a execução de *workflow* científicos em plataformas de nuvens de computadores. O CRCPs combina o processo heurístico do GRASP a o método estatístico de regressão linear múltipla, do inglês *Multiple Linear Regression* (MLR), para prever os recursos do usuário e informar o custo financeiro antes do início da execução. O CRCPs prevê o tempo de execução e o custo financeiro de cada *workflow* por meio do uso de um banco de dados histórico construído pelo monitoramento das execuções. Entretanto, o CRCPs pode enfrentar desafios em cenários com alta variabilidade nos requisitos dos *workflows* e recursos das nuvens, o que pode afetar a eficiência das previsões.

Shealy et al. [2021] utilizam *Artificial Neural Networks* (ANN) para dimensionar o uso de recursos e otimizar a alocação em tempo real para execução de *workflow* científicos. A abordagem coleta dados históricos de execução, como consumo de CPU e memória, para treinar a ANN, que estima as necessidades futuras de recursos. Com base nessas previsões, um algoritmo de otimização multiobjetivo ajusta dinamicamente a alocação de máquinas virtuais para equilibrar o tempo de

execução e os custos. A solução permite reconfigurações em tempo real, garantindo respostas rápidas a variações inesperadas nas demandas das aplicações, resultando em maior eficiência no uso de recursos e redução de custos. Contudo, a eficácia do método depende da disponibilidade de grandes volumes de dados históricos para treinar os modelos.

de la Torre and Halappanavar [2023] propõem uma estratégia para alocação de recursos de computacionais em nuvem para *workflows* científicos com otimização de custo, utilizando a técnica de relaxação de *Lagrange*. O problema é modelado como um programa linear inteiro misto (MILP), e a relaxação de *Lagrange* permite mover algumas restrições complexas para a função objetivo, criando um problema mais simples de resolver. A decomposição da demanda em subproblemas menores é facilitada por essa relaxação, permitindo a otimização da alocação de recursos. O método ajusta os multiplicadores de Lagrange para estimar os custos normalizados e, assim, identificar os recursos mais custo-eficientes para cada intervalo de tempo. Essa abordagem reduz significativamente o tempo computacional, de horas para segundos, com erro relativo inferior a 0,05%. Os autores ressaltam que a definição correta dos multiplicadores de *Lagrange* e das condições de fronteira é essencial para garantir a qualidade das soluções.

Embora soluções puramente matemáticas possam entregar resultados ótimos ou quase ótimos, elas enfrentam limitações relacionadas a escalabilidade e tempo de processamento devido à complexidade computacional envolvida. Essas restrições apoiam a escolha por soluções fundamentadas em heurísticas, que são mais simples de implementar e escaláveis, ainda que possam não oferecer soluções ótimas [Adhikari et al., 2019].

Conforme observado na Tabela III.1, somente dois trabalhos, Coutinho et al. [2015] e Abdi et al. [2018], empregam um modelo matemático para fundamentar a heurística proposta. Além disso, o estudo de Abdi et al. [2018] é projetado para aplicações BoT, de modo que apenas um trabalho aborda o dimensionamento de recursos para *pipelines*, por meio de uma heurística respaldada por um modelo matemático exato.

Diante dessa lacuna da literatura, neste estudo, tratamos o problema de dimensionamento eficiente de recursos em *workflows* científicos com um modelo matemático exato (*CC-LB*) e uma heurística (*GraspCC-LB*). Nesse contexto, a heurística oferece uma solução computacionalmente eficiente em termos de tempo de processamento, sendo respaldada por um modelo matemático que permite validar seus resultados. Propomos, também, a estruturação do *workflow* como um *pipeline*, composto por estágios ou níveis paralelos de tarefas semelhantes operando de forma independente. Acreditamos que esta seja uma abordagem promissora para a estimativa do tempo computacional e para o correto dimensionamento dos recursos necessários aos usuários que precisam executar *workflows* científicos com alta demanda computacional.

Capítulo IV Definição e Modelagem do Problema

Este estudo investiga o problema de dimensionamento de recursos para execução de *workflows* científicos em *clusters* e nuvens computacionais, e tem por premissa a disponibilidade prévia de informações sobre as características dos *workflows* e dos tipos e quantidade de máquinas disponíveis. Neste capítulo o problema é definido e modelado como um problema de programação inteira por meio de uma formulação matemática, na qual um determinado *workflow*, com restrições de custo financeiro e tempo de execução, é submetido a um arranjo computacional. O *workflow* consiste em um conjunto de **atividades**, cada uma podendo englobar múltiplas tarefas executáveis em paralelo, com exigências computacionais heterogêneas, tais como: necessidade de armazenamento em disco, de memória e demanda de processamento.

Esse modelo é aplicável em áreas como ciência de dados, bioinformática e simulações científicas, permitindo a divisão de atividades afins em conjuntos que podem ser executados em paralelo, sem a necessidade de comunicação ou coordenação entre elas [Shalf et al., 2011]. A solução apresentada neste trabalho é particularmente relevante, uma vez que trata de um problema comum em ambientes de nuvem [Habib et al., 2010], no qual é necessário alocar recursos computacionais de maneira eficiente para atender aos custos financeiros e aos prazos de processamento estabelecidos.

A Figura IV.1 apresenta um exemplo de *workflow* científico. Nela apresentamos o conceito de: *(i)* **agregação** - processo conduzido por uma atividade que agrupa e processa as saídas de duas ou mais atividades, gerando uma síntese dos dados como resultado; e *(ii)* **distribuição** - processo realizado por uma atividade que gera dois ou mais conjuntos de dados de saída. Estes conjuntos são posteriormente recebidos como entrada por duas ou mais atividades distintas, representando uma disseminação de dados para várias etapas de processamento subsequentes [Ogasawara, 2011].

Neste estudo, as etapas de processamento do *workflow* são categorizadas como ***layers***, nas quais agrupam atividades que são independentes entre si, permitindo sua execução paralela. Contidos nos *layers*, encontram-se os ***buckets***, que são constituídos por atividades correlatas, independentes entre si, e que podem ser agrupadas e processadas como em um BoT. Para que sejam agrupadas em *buckets*, as atividades, tipicamente, são de um mesmo tipo no *workflow*. No que se refere ao processamento, os *buckets* contidos em um *layer* são executados de forma paralela, enquanto os *layers* do *workflow* são executados de forma sequencial. É importante observar que a solução foi estruturada de modo que os *buckets* do *layer* l só podem ser executados se, e somente se,

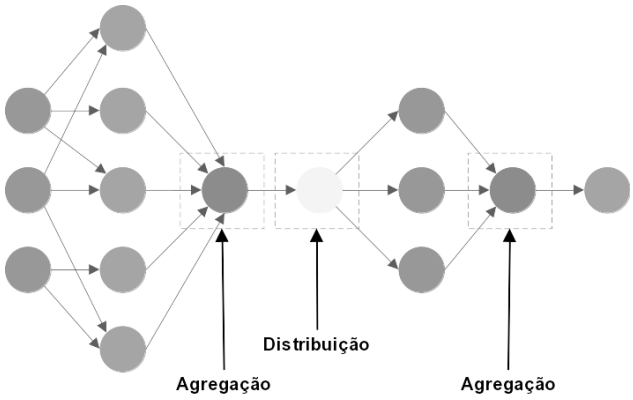


Figura IV.1: Exemplo de *Workflow* científico

todos os *buckets* do *layer* $l-1$ tiverem sido concluídos. Este estudo denomina o esquema descrito anteriormente como ***Layered-Bucket*** (LB), ilustrado na Figura IV.2.

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 (o custo de uso de uma máquina por um período de tempo) e recursos computacionais como armazenamento em disco DS_p, capacidade de memória MC_p e processamento GF_p, medido em *gigaFLOP* (GFLOP) por período de tempo. Além disso, não é incomum que os provedores de serviços de nuvem tenham um limite máximo de máquinas v_{max} alocadas para cada usuário em cada período de tempo. Além disso, são definidos os requisitos dos usuários, tais como: o custo financeiro máximo c_{max}, o tempo máximo de execução t_{max}, o conjunto de *layers* L, a quantidade de *buckets* B_l de cada *layer* $l \in L$. Os requisitos de armazenamento em disco ds_b^l, a capacidade de memória mc_b^l e o número máximo de máquinas disponíveis v_b^l são definidos para cada *bucket* $b \in B_l$ do *layer* $l \in L$, em cada período de tempo. Já o requisito gf_b^l, em GFLOP, representa a demanda total de processamento estabelecida para cada *bucket* $b \in B_l$ do *layer* $l \in L$, independentemente do período de tempo. A Tabela IV.1 sumariza as notações utilizadas.

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, temos a variável z, também inteira, que contém o último período de tempo em que uma máquina foi alocada pelo usuário. A variável z representa, no contexto de escalonamento, o *makespan*, que corresponde ao período de tempo total decorrido desde o início até a conclusão da execução de todas as tarefas do *workflow*. As variáveis são apresentadas na Tabela IV.2.

O modelo pode ser formulado como a seguir:

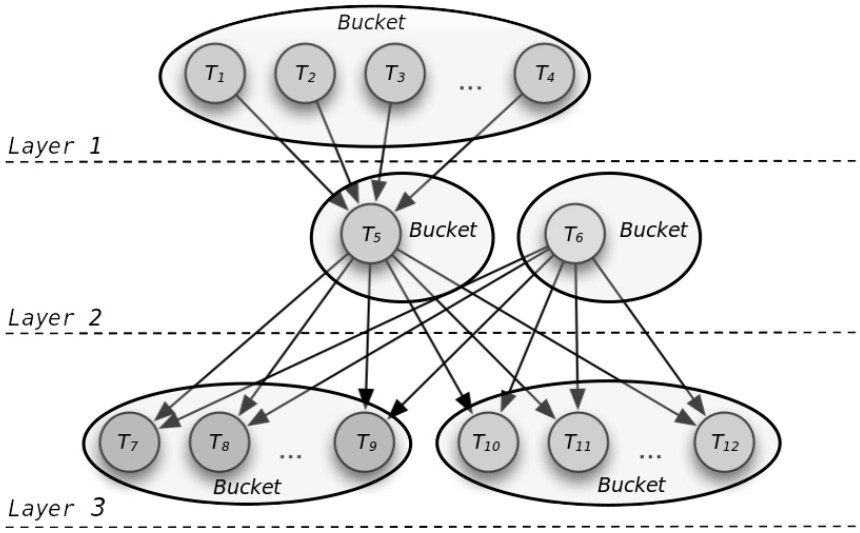


Figura IV.2: Abordagem *Layered-Bucket*

Tabela IV.1: Notações para o ambiente computacional adotado

Notação	***Descrição***
L	o conjunto de *layers*.
B_l	o conjunto de *buckets* de cada *layer* $l \in L$.
P	o conjunto dos tipos de máquinas.
c_{max}	o requisito de custo financeiro máximo.
t_{max}	o requisito de tempo de execução máximo.
v_{max}	o limite máximo de máquinas alocadas para cada usuário, em cada período de tempo.
T	o conjunto de períodos de tempo viáveis ($T = \{1 \dots t_{max}\}$).
v_b^l	o número máximo de máquinas por *bucket* $b \in B_l$ do *layer* $l \in L$, em cada período de tempo.
ds_b^l	o requisito de armazenamento em disco por *bucket* $b \in B_l$ do *layer* $l \in L$, em cada período de tempo.
mc_b^l	o requisito de capacidade de memória por *bucket* $b \in B_l$ do *layer* $l \in L$, em cada período de tempo..
gf_b^l	o requisito de demanda por processamento por *bucket* $b \in B_l$ do *layer* $l \in L$.
DS_p	o armazenamento em disco disponibilizado pela máquina do tipo $p \in P$.
MC_p	a capacidade de memória disponibilizado pela máquina do tipo $p \in P$.
GF_p	o poder de processamento (GFLOP) disponibilizado pela máquina do tipo $p \in P$.
C_p	o custo de alocação de uma máquina do tipo $p \in P$ por um período de tempo.

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

Tabela IV.2: Descrição das variáveis na formulação matemática

Variáveis	***Descrição***
x_{pitbl}	$x_{pitbl} = 1$, se e somente se a máquina $i \in \{1, \cdots, v_{max}\}$ do tipo $p \in P$ estiver alocada no tempo $t \in T$ para o *bucket* $b \in B_l$ do *layer* $l \in L$.
z_{pi}	o último período de tempo em que a máquina $i \in \{1, \cdots, v_{max}\}$ do tipo $p \in P$ foi alocada.
z	o último período de tempo em que uma máquina foi alocada pelo usuário.

sujeito a

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

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

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

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

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

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

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

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

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

$$z \geq z_{pi} \quad \forall p \in P, \forall i \in \{1, \ldots, v_{max}\} \tag{IV.11}$$

$$\sum_{p \in P} \sum_{i=1}^{v_{max}} C_p \cdot z_{pi} \leq c_{max} \tag{IV.12}$$

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

$$z, z_{pi} \in \mathbb{Z} \qquad \forall p \in P, \forall i \in \{1,\ldots,v_{max}\} \tag{IV.14}$$

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

No modelo *CC-LB*, apresentando acima, a função objetivo IV.1 busca minimizar tanto o custo financeiro quanto o tempo de execução total, ambos normalizados. Os pesos α_1 e α_2 atribuem a importância relativa a esses objetivos, conforme definido pelo usuário, permitindo priorizar o custo ou o tempo de execução, respectivamente.

As restrições IV.2 e IV.3 garantem que as quantidades de armazenamento e memória contratadas sejam maiores que as demandas de cada *bucket* dos *layers*, em cada unidade de tempo, respectivamente. De forma semelhante, a restrição IV.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 IV.5, e a restrição IV.6 foi incluída para estabelecer uma ordem entre a alocação das máquinas no *bucket* e eliminar soluções simétricas (ou seja, soluções diferentes, mas com a mesma configuração). Ainda em relação ao processamento, a restrição IV.7 garante a ordem de processamento entre os *layers*.

A desigualdade IV.8 garante que o número de máquinas alocadas simultaneamente não exceda o limite máximo disponibilizado pela infraestrutura computacional adotada. Já a restrição IV.9 garante o limite de máquinas disponibilizadas por *bucket* em cada período de tempo.

A restrição IV.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 IV.11 determina o maior tempo de processamento dentre todas as máquina alocadas, isto é, o tempo total de processamento do *workflow*. Finalmente, a restrição IV.12 garante que o custo de alocação das máquinas não ultrapasse o custo financeiro máximo disponível.

As restrições IV.13 e IV.14 definem os requisitos de integralidade e não-negatividade das variáveis.

Capítulo V Heurística para dimensionamento de recursos para execução de *workflows* em ambientes HPC

Procedimentos exatos muitas vezes se revelam ineficientes na busca por soluções devido à sua elevada demanda temporal [Malawski et al., 2015]. Esta ineficiência se torna ainda mais acentuada quando lidamos com problemas do mundo real de larga escala, como os *workflows* científicos. Tais *workflows* se destacam por sua complexidade intrínseca, a qual surge devido à variedade de tarefas interconectadas e à necessidade de alocação eficiente de recursos, intensificando a dificuldade em localizar soluções ótimas.

No âmbito deste estudo, a alocação eficiente das máquinas surge como uma questão crítica. O tempo despendido neste processo deve ser o mínimo possível, tendo em vista que, em comparação com o tempo total de execução do *workflow*, seu impacto deve ser inexpressivo. Uma demora excessiva na alocação de recursos pode reduzir a eficiência de todo o sistema.

Por essa razão, a utilização de heurísticas torna-se uma alternativa viável. Estas estratégias podem fornecer soluções sub-ótimas em um intervalo de tempo significativamente menor. No contexto dos *workflows* científicos, tais soluções não são apenas aceitáveis, mas frequentemente desejáveis. Isso se deve ao fato de que a flexibilidade oferecida por essas soluções pode facilitar o ajuste dinâmico do sistema a mudanças no ambiente de execução ou nos requisitos do *workflow*.

Neste contexto, a heurística GRASP, descrita por Feo and Resende [1995] como um procedimento de busca local com reinicializações, destaca-se. O algoritmo é estruturado em duas fases principais: a fase de construção e a fase de busca local. Durante a fase de construção, são combinadas técnicas gulosas com seleção aleatória para gerar uma solução inicial diversificada, formando a base para a fase subsequente de busca local. Essa solução inicial é então refinada na fase de busca local, onde pequenas alterações são feitas para tentar melhorar a solução até que um mínimo local seja alcançado. O processo é repetido múltiplas vezes, com reinicializações, permitindo a exploração de diferentes regiões do espaço de solução, a fim de encontrar uma solução satisfatória. Essa característica torna a heurística GRASP adequada para o problema, dado que sua implementação tem o potencial de acelerar significativamente o desempenho da execução do *workflow*. Ao diminuir o tempo necessário para a alocação de recursos, a heurística pode melhorar a eficiência geral do sistema, proporcionando um meio mais eficaz de lidar com a complexidade dos *workflows* científicos. A abordagem proposta considera uma fase de pré-processamento para a construção dos *layers* e

buckets, baseada nas atividades e dependências do *workflow*.

A notação e a solução empregada para a atribuição de máquinas neste trabalho foram adaptadas de [Coutinho et al., 2015]. A notação, conforme descrito no Capítulo IV ao apresentar o modelo *CC-LB*, é replicada nos parágrafos seguintes para fins de simplificação.

Nesse sentido, considere-se que P é o conjunto de todas as máquinas disponíveis de um provedor de serviço. Cada tipo de máquina p, pertencente ao conjunto P, possui um custo financeiro associado C_p. Este custo, que representa o preço de alocar uma máquina por um período de tempo determinado, que pode variar de 1 segundo a 1 hora, dependendo das condições estabelecidas pelo provedor de serviço. Adicionalmente, cada máquina p tem recursos computacionais associados, tais como armazenamento em disco DS_p, capacidade de memória MC_p e poder computacional GF_p.

As exigências do usuário também devem ser consideradas, incluindo o custo financeiro máximo c_{max} e o tempo máximo de execução r_{max}, além das necessidades específicas de cada tarefa do *workflow*, que são estruturadas para determinar as necessidades específicas de cada *layer* e *bucket*. Essas últimas podem incluir a necessidade de armazenamento em disco ds_b^l, de memória mc_b^l, e a demanda de processamento gf_b^l. Considere, ainda, o limite máximo de máquinas v_{max} que podem ser alocadas pelos usuários, que também varia de acordo com as condições estabelecidas pelo provedor de serviço.

Para explicar o método é necessário introduzir algumas notações adicionais, que são apresentadas na Tabela V.1. Uma solução $\{(p_1, i_1, r_1, b_1, l_1), (p_2, i_2, r_2, b_2, l_2), \cdots\}$ é definida como um conjunto de 5-tuplas (p, i, r, l, b). Cada tupla simboliza que a máquina i, do tipo p, foi alocada no período r no *bucket* b do *layer* l. S é o conjunto que engloba todas as soluções possíveis. Uma solução específica $s \in S$ é considerada viável se atender às exigências do usuário. Ademais, denotamos z como $\max_{(p,i,r,b,l)\in s} r$, ou seja, o último período em que uma máquina foi utilizada.

Tabela V.1: Notações adicionais para *GraspCC-LB*

Notação	***Descrição***
(p, i, r, b, l)	$(p, i, r, b, l) \in solution$, se a máquina i do tipo p for alocada no tempo r para o *bucket* b do *layer* l.
$z(s)$	o último período de tempo em que uma máquina foi selecionada na solução s.
q_{max}	a quantidade máxima de *buckets* por *layer*.
T	o conjunto de tarefas do *workflow*.
act_t	a classificação da atividade da tarefa t.
d_t	requisito de armazenamento em disco da tarefa t.
m_t	o requisito de capacidade de memória da tarefa t.
g_t	o requisito de demanda por processamento da tarefa t.
$succ_t$	o conjunto de tarefas sucessoras da tarefa t.
$pred_t$	o conjunto de tarefas predecessoras da tarefa t.
$order_t$	a ordem da tarefa t.

As notações acima são fundamentais para a definição da função de custo, descrita na Equação V.1, que tem o objetivo de medir a qualidade da solução s. A função de custo (Equação V.1), da mesma forma que a função objetivo (Equação IV.1), busca minimizar o custo financeiro e o tempo, a fim de identificar a melhor solução viável do problema.

$$\mathrm{F(s)} = \alpha_1 \sum_{(p,i,r,b,l)\in s} \frac{C_p}{c_{max}} \; + \; \alpha_2 \, \frac{z}{r_{max}} \tag{V.1}$$

A heurística *GraspCC-LB* é composta por duas etapas principais: *(i)* a etapa de pré-processamento do *workflow*, que define as *layers* e os *buckets*; e *(ii)* a etapa de processamento, no qual a estimativa de recursos para a execução de cada *bucket* nos *layers* é realizada. O procedimento *GraspCC-LB*, apresentado no Algoritmo 1, recebe como parâmetros de entrada o *workflow* W para estimar os recursos necessários para executar seu conjunto de tarefas T; o conjunto de máquinas P com informações sobre suas capacidades de disco DS_p, memória MC_p e poder de processamento GF_p; a quantidade máxima de *buckets* por *layer* q_{max}; as restrições de custo c_{max} e de tempo r_{max} do usuário; e os pesos atribuídos ao custo e ao tempo, α_1 e α_2, respectivamente. O procedimento inicia com uma solução global vazia s^* de custo infinito e realiza a etapa de pré-processamento para definir os *layers* e os *buckets* de acordo com a especificação de *workflow* recebida. Para este propósito, o procedimento $LayeredBuckets$, detalhado no Algoritmo 2, é invocado para definir os *layers* e os requisitos de disco, memória e demanda de processamento de cada *bucket* com base nas demandas computacionais das tarefas do *workflow*.

Algoritmo 1 - GraspCC-LB $(W, P, q_{max}, c_{max}, r_{max}, \alpha_1, \alpha_2)$

1: $s^* \leftarrow \emptyset$;
2: $F(s^*) \leftarrow \infty$;
3: $layers \leftarrow LayeredBuckets(W, q_{max})$;
4: $iter \leftarrow 0$;
5: **while** $(iter \leq maxIter)$ **do**
6: $\quad s \leftarrow \emptyset$;
7: $\quad$ **for** $l \in layers$ **do**
8: $\quad\quad$ **for** $b \in l$ **do**
9: $\quad\quad\quad \bar{s} \leftarrow GraspCC(P, c_{max}, r_{max}, ds_b^l, mc_b^l, gf_b^l, \alpha_1, \alpha_2)$;
10: $\quad\quad\quad s \leftarrow s \cup \bar{s}$;
11: $\quad\quad$ **end for**
12: $\quad$ **end for**
13: $\quad iter \leftarrow iter + 1$;
14: $\quad$ **if** $((s \; is \; feasible) \;\; \& \;\; (F(s) < F(s^*))$ **then**
15: $\quad\quad s^* \leftarrow s$;
16: $\quad\quad iter \leftarrow 0$;
17: $\quad$ **end if**
18: **end while**
19: **return** s^*;

O procedimento $LayeredBuckets$ organiza o *workflow* W recebido em *layers* e *buckets*, propor-

cionando o agrupamento de tarefas. Cada tarefa $t \in T$ do *workflow* W possui metadados associados, como a atividade act_t com a qual a tarefa está relacionada, requisitos de disco d_t, memória m_t e demanda de processamento g_t, conjuntos de tarefas sucessoras e predecessoras, $succ_t$ e $pred_t$, respectivamente, e a ordem de execução $order_t$ que será atribuída a ela.

Todas as tarefas do *workflow* W inicialmente possuem ordem zero ($order_t = 0$). O procedimento *LayeredBuckets* inicia estabelecendo a estrutura de dados *layers* como um conjunto vazio. Esta estrutura é uma lista ordenada, onde cada elemento (ou seja, um *layer*) consiste em um conjunto de *buckets*. Cada *bucket*, por sua vez, é organizado considerando a soma dos requisitos computacionais de disco ($\sum d_t$), memória ($\sum m_t$) e poder de processamento ($\sum g_t$) do conjunto de tarefas que possuem a mesma classificação de atividade previamente definida. É importante observar que a classificação de atividade act_t pode ter múltiplos significados (por exemplo, o mesmo programa invocado, atividades executadas em paralelo com MPI, etc.), permitindo que o *GraspCC-LB* explore diferentes abordagens de agrupamento de tarefas.

Algoritmo 2 - LayeredBuckets (W, q_{max})

```
1:  groups ← ∅;
2:  layers ← ∅;
3:  for t ∈ T of W do
4:      for k ∈ succ_t do
5:          if order_k ≤ order_t then
6:              order_k ← order_t + 1;
7:          end if
8:      end for
9:  end for
10:
11: for t ∈ T of W do
12:     G ← t;
13:     for t' ∈ T of W | t ≠ t' do
14:         if order_t = order_t' ∧ act_t = act_t' then
15:             G ← G ∪ t';
16:             T ← T − t';
17:         end if
18:     end for
19:     groups ← groups ∪ G;
20:     T ← T − t;
21: end for
22:
23: for G ∈ groups do
24:     buckets ← BalancedBuckets(G, q_max);
25:     l ← order_t | t ∈ G;
26:     layers[l] ← layers[l] ∪ buckets;
27: end for
28: return layers;
```

Em seguida, o *LayeredBuckets* insere uma tarefa artificial t, que se torna a única tarefa de

entrada do *workflow* W. Esta tarefa artificial é a predecessora de todas as tarefas de entrada reais do W. Cada tarefa t do *workflow* W é processada iterativamente a partir dela. As tarefas sucessoras de t ($succ_t$) são examinadas, e suas ordens são comparadas com a ordem de t. Se a ordem da tarefa sucessora k não for superior à da tarefa t, a tarefa k é atualizada para a ordem subsequente à de t. O processo continua até que todas as tarefas tenham sido processadas. A ordem final de cada tarefa representa a identificação de seu *layer*. Após isso, são criados *groups*, agrupando tarefas que compartilham a mesma ordem ($order_t$) e classificação de atividade (act_t). Posteriormente, para cada agrupamento de tarefas $G \in groups$, o procedimento $BalancedBuckets$, detalhado no Algoritmo 3, é chamado para criar os *buckets*, que são incorporados ao seu respectivo *layer*. Esse procedimento garante que a distribuição das tarefas seja realizada de maneira uniforme, de acordo com a demanda de processamento da tarefa g_t. Finalmente, a estrutura de $layers$ é retornada pelo procedimento $LayeredBuckets$.

Algoritmo 3 - BalancedBuckets ($tasks$, q_{max})

```
1:  buckets ← ∅;
2:  orderedTasks ← sort(tasks);
3:  for t ∈ orderedTasks do
4:      if | buckets | ≤ q_max then
5:          buckets ← buckets ∪ (d_t, m_t, g_t);
6:      else
7:          min ← MinGFLOP(buckets);
8:          ds_min ← ds_min + d_t;
9:          mc_min ← mc_min + m_t;
10:         gf_min ← gf_min + g_t;
11:     end if
12: end for
13: return buckets;
```

O procedimento $BalancedBuckets$ cria um conjunto vazio de *buckets*, tal que $|buckets| = q_{max}$, e então ordena o conjunto de tarefas recebido ($tasks$) em ordem decrescente de acordo com o requisito de processamento g_t, formando o conjunto $orderedTasks$. Para cada tarefa $t \in orderedTasks$, se o conjunto de *buckets* ainda não estiver completo, um *bucket* composto pelos requisitos computacionais de disco (d_t), memória (m_t) e processamento (g_t) da tarefa t é adicionado ao conjunto de *buckets* (em outras palavras, um *bucket* é criado com os requisitos computacionais da tarefa t). Caso contrário, o *bucket* com o menor requisito de processamento até aquele momento (min) é selecionado para incluir uma tarefa t. Este passo visa distribuir as tarefas de modo que nenhum *bucket* no *workflow* fique sobrecarregado. Uma vez identificado o *bucket* min, os requisitos computacionais da tarefa t relacionados a disco (d_t), memória (m_t) e processamento (g_t) são acumulados nele (*i.e.*, ds_{min}, mc_{min} e gf_{min}, respectivamente). Esse procedimento é repetido até que todas as tarefas sejam alocadas em um *bucket*, garantindo que a distribuição das tarefas seja realizada de

maneira uniforme. Ao final do processo, o algoritmo retorna a estrutura de *buckets* criada.

Após definir os $layers$, o procedimento *GraspCC-LB*, apresentado no Algoritmo 1, inicia a etapa de estimativa de recursos e adota a noção de melhoria sucessiva usando o parâmetro $maxIter$. Esse parâmetro determina o número máximo de iterações sem melhoria na melhor solução encontrada. Uma solução vazia s é inicializada para cada iteração, e para cada *layer* l, seus *buckets* são processados individualmente. Para cada *bucket* $b \in l$, o procedimento *GraspCC*, adaptado do trabalho de Coutinho et al. [2015] e apresentado no Algoritmo 4, é invocado para construir uma solução local s, o que é aceitável, visto que *GraspCC* é um método para estimativas de recursos para aplicações executadas como *BoTs*, e as tarefas no mesmo *bucket* são processadas dessa maneira. *GraspCC* recebe como parâmetros o conjunto de recursos P, as restrições de custo financeiro máximo c_{max} e tempo r_{max}, os requisitos computacionais ds_b^l, mc_b^l e gf_b^l, e os pesos atribuídos ao custo financeiro α_1 e ao tempo α_2.

O *GraspCC* também adota a noção de melhorias sucessivas, neste caso, usando o parâmetro $maxIteration$, que determina o número máximo de iterações sem melhoria na melhor solução encontrada, e possui duas fases: a fase de construção *coCC* e a fase de busca local $lsCC$, conforme descrito em Coutinho et al. [2015]. Cada iteração do *GraspCC* tem como ponto de partida a construção de uma solução a partir de um conjunto vazio, realizada de maneira gulosa e aleatória pelo procedimento *coCC*, como mostrado no Algoritmo 5.

Algoritmo 4 - GraspCC $(P, c_{max}, r_{max}, DS, MC, GF, \alpha_1, \alpha_2)$

```
1:  s* ← ∅; F(s*) ← ∞; iterations ← 0;
2:  while (iterations ≤ maxIteration) do
3:      s ← coCC(P, c_max, r_max, DS, MC, GF, α1, α2);
4:      s ← lsCC(s, P, c_max, r_max, DS, MC, GF, α1, α2);
5:      iterations ← iterations + 1;
6:      if (F(s) < F(s*)) and (s is feasible) then
7:          s* ← s; iterations ← 0;
8:      end if
9:  end while
10: return s*;
```

No algoritmo *coCC*, uma estrutura ordenada, denominada LP, é estabelecida. Esta estrutura abriga as máquinas $p \in P$, ordenadas de forma decrescente em relação ao custo financeiro e o poder de processamento. Em cada iteração, uma máquina p é selecionada aleatoriamente, entre as primeiras β máquinas, para ser adicionada ao primeiro período de tempo. Note que o parâmetro β define o grau de aleatoriedade que a fase de construção terá. Esse processo se repete até que a solução satisfaça os requisitos de armazenamento DS e memória MC (para $r = 1$), ainda que as restrições de tempo máximo e custo financeiro sejam flexibilizadas para estimular a diversidade da solução inicial. Após isso, as máquinas selecionadas podem ser replicadas nos períodos futuros até

que a demanda de processamento seja atendida. Vale ressaltar que para o *GraspCC*, uma solução é definida como um conjunto de 3-tuplas (p, i, r), utilizada para dimensionar os recursos de aplicações executadas como BoT. Cada tupla simboliza que a máquina i, do tipo p, foi alocada no período r.

Algoritmo 5 - coCC $(P, c_{max}, r_{max}, DS, MC, GF, \alpha_1, \alpha_2)$

```
1: s ← ∅; LP ← Order(P);
2: while (∑_{p|(p,i,1)∈s} ds_p < DS) or (∑_{p|(p,i,1)∈s} mc_p < MC) do
3:     Escolha a máquina p̄ (índice ī) aleatoriamente entre os primeiros β elementos de LP
4:     s ← s ∪ {(p̄, ī, 1)}
5: end while
6: r̄ ← 2
7: while (∑_{p|(p,i,r)∈s} gf_p < GF) do
8:     s ← ⋃_{(p,i,1)∈s} (p, i, r̄) ∪ s
9:     r̄ ← r̄ + 1
10: end while
11: return s;
```

A abordagem construtiva não assegura que uma solução viável ou localmente ótima sejam alcançadas em relação a um conjunto de vizinhos. Por essa razão, o procedimento de busca local *lsCC*, apresentado no Algoritmo 6, é implementado com o objetivo de aprimorar a solução s. A vizinhança $Nr(s)$ é composta por todas as soluções resultantes da troca de n tuplas em s por outras n tuplas ausentes em s. Tais trocas são realizadas de maneira abrangente, em consonância com a estratégia de melhoramento inicial. O algoritmo *lsCC* inicia com a solução fornecida pela fase de construção e, de maneira iterativa, substitui a solução atual pela solução de custo mínimo F na vizinhança. A busca local é encerrada quando nenhuma melhoria é identificada na vizinhança da solução atual.

Algoritmo 6 - lsCC $(s, P, c_{max}, r_{max}, DS, MC, GF, \alpha_1, \alpha_2)$

```
1: while s improving do
2:     for all s̄ ∈ (N1(s) ∪ N2(s)) do
3:         if F(s̄) < F(s) then
4:             s ← s̄;
5:         end if
6:     end for
7: end while
8: return s;
```

Por fim, a solução $\bar{s}$, retornada pelo *GraspCC*, é incorporada à solução local s do *GraspCC-LB*. Se s for viável, isto é, se atender às restrições de custo financeiro c_{max} e tempo de processamento r_{max} impostas pelo usuário, e tiver um custo menor que a melhor solução global encontrada até o momento (s^*), então s se torna a nova melhor solução global e o iterador é reiniciado. O algoritmo termina quando o número máximo de iterações $(maxIter)$ é alcançado sem melhoria na solução global, retornando a melhor solução global encontrada (s^*).

Capítulo VI Resultados Experimentais

Este capítulo apresenta os experimentos computacionais realizados com o objetivo de avaliar o desempenho do modelo *CC-LB* e da heurística *GraspCC-LB*. O computador onde os experimentos computacionais foram conduzidos, tanto o modelo como as heurísticas, é equipado com um Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz e 130 GB de RAM, sob sistema operacional Linux Ubuntu v20.04.1. O modelo foi implementado no Gurobi versão 9.5, configurado com os parâmetros padrão e limite de tempo de 86.400 segundos, e a heurística *GraspCC-LB* implementada na linguagem ISO/IEC C++17. As execuções no mundo real foram realizadas em um *cluster* de três nós com processadores Intel(R) Core(TM) i7-13700 16-Core CPU @ 5.20GHz, 130 GB de RAM e 4TB de espaço em disco. Para isso, foi utilizado o escalonador de código aberto `Ray`[1].

Para avaliar a heurística *GraspCC-LB*, três diferentes experimentos foram conduzidos: *(i)* comparação dos resultados do modelo exato *CC-LB* com a heurística *GraspCC-LB*; *(ii)* avaliação de diferentes estratégias de agrupamento de tarefas geradas pela heurística *GraspCC-LB*; e *(iii)* avaliação da estimativa de recursos produzida pela heurística *GraspCC-LB* em um ambiente real de HPC.

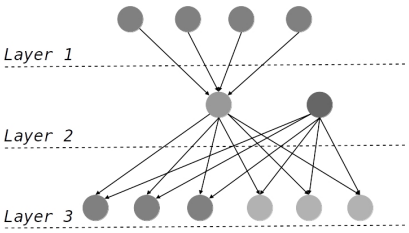
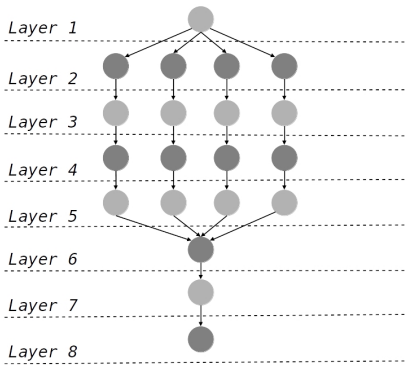
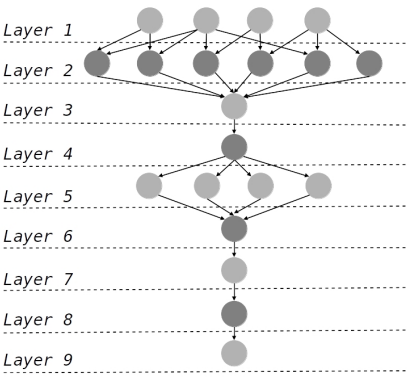
Os experimentos foram conduzidos em cenários reais utilizando três *workflows* científicos, são eles: *(i)* ***1000 Genomes***, que fornece uma referência para a variação humana, reconstruindo os genomas de 2504 indivíduos de 26 populações distintas. O projeto foi conduzido pelo *1000 Genome Project Consortium* [Consortium, 2015]; *(ii)* ***Epigenomics***, que realiza um extenso mapeamento do estado epigenético das células humanas, com o objetivo de compreender as alterações epigenéticas que afetam a expressão genética e estão vinculadas a diversas doenças. O projeto é desenvolvido pelo *USC Epigenome Center* [Bharathi et al., 2008]; e *(iii)* ***Montage***, que é um conjunto de ferramentas de código aberto, criado pela NASA/IPAC *Infrared Science Archive*, usado para gerar mosaicos personalizados do céu [Berriman et al., 2004].

Os detalhes dos *workflows* são apresentados na Tabela VI.1. A primeira coluna contém a visão conceitual do *workflow* com o *pipeline* de atividades e a subdivisão em *layers*. Para cada tipo de *workflow*, as demais colunas indicam o nome das atividades e uma descrição sucinta de cada uma delas, respectivamente.

As configurações das máquinas utilizadas como parâmetros para execução do modelos *CC-*

[1] `https://www.ray.io/`

Tabela VI.1: Visão Geral dos *Workflows* Científicos

Visão conceitual do *workflow*	**Atividade**	**Descrição**
1000Genome [Ferreira da Silva et al., 2019] 	(1) `individuals`	busca e analisa dados da Fase 3 do projeto 1000 Genomes por cromossomo, organizando informações genéticas individuais para posterior análise e comparação.
	(2) `individuals_merge`	combina todos os resultados da tarefa `individuals` em um único arquivo, facilitando o processamento subsequente e a análise de todos os dados recolhidos.
	(3) `sifting`	calcula os *scores1* SIFT de todas as variantes de polimorfismos de nucleotídeo único (SNPs), conforme calculado pelo *Variant Effect Predictor*.
	(4) `mutation_overlap`	mede a sobreposição de mutações (SNPs) entre pares de indivíduos.
	(5) `frequency_overlap_mutations`	calcula a frequência de mutações sobrepostas em subamostras de certos indivíduos.
Epigenomics [Juve et al., 2013] 	(1) `fastqSplit`	divide arquivos FASTQ grandes em subarquivos menores, facilitando o processamento paralelo de sequências genômicas.
	(2) `filterContams`	filtra contaminantes em sequências genômicas, melhorando a qualidade dos dados para análises subsequentes.
	(3) `sol2sanger`	converte os valores de qualidade de sequências genômicas do formato Illumina Solexa para o padrão Sanger, facilitando a compatibilidade com outras ferramentas bioinformáticas.
	(4) `fast2bfq`	converte arquivos de sequência FASTQ para o formato binário BFQ. Usado para otimizar o armazenamento e a velocidade de processamento em análises de sequenciamento de próxima geração.
	(5) `map`	alinhamento de sequências de leitura curtas (*reads*) a uma sequência de referência. Vital para entender a localização genômica de cada *reads* e sua correspondência com o genoma de referência.
	(6) `mapMerge`	combina os resultados de múltiplas tarefas de *map*.
	(7) `chr21`	designação utilizada para o cromossomo 21. Em genômica, tarefas rotuladas como *chr21* envolvem análises específicas desse cromossomo, como mapeamento de leituras de sequenciamento e identificação de variantes.
	(8) `pileup`	formato de arquivo utilizado na bioinformática para representar alinhamentos de sequências de DNA.
Montage [Rynge et al., 2014] 	(1) `mProject`	usado para reprojecionar uma única imagem para a escala definida em um modelo alternativo de cabeçalho *Flexible Image Transport System* (FITS), que é o formato de arquivo padrão mais comum em astronomia para imagens e tabelas.
	(2) `mDiffFit`	calcula a diferença entre duas imagens astronômicas e ajusta um plano ao resíduo para melhor alinhamento.
	(3) `mConcatFit`	combina diversas imagens astronômicas corrigidas em uma única imagem, levando em conta os ajustes de plano feitos em cada uma delas.
	(4) `mBgModel`	modela e corrige diferenças de fundo em imagens sobrepostas, essencial para montagem de mosaicos astronômicos de alta qualidade.
	(5) `mBackground`	remove uma função de plano de fundo modelada de uma única imagem, permitindo a criação de mosaicos com transições de fundo suaves entre imagens.
	(6) `mImgtbl`	extrai metadados de imagens FITS para criar uma tabela ASCII que será usada para gerar um mosaico de imagens a partir destas.
	(7) `mAdd`	junta múltiplas imagens FITS, já alinhadas e processadas, para criar um mosaico final de todas elas.
	(8) `mViewer`	ferramenta que gera uma visualização gráfica de uma ou mais imagens FITS. Ele suporta sobreposições gráficas, ajustes de cor e zoom, entre outras funcionalidades.

LB e da heurística *GraspCC-LB* estão descritas na Tabela VI.2, na qual as colunas representam, respectivamente, o nome da máquina, a capacidade de memória RAM e de armazenamento em disco em GB, a quantidade de GFLOP e o custo financeiro para alocar esta máquina por 1 segundo.

Tabela VI.2: Características das Máquinas

Máquina	**RAM (GB)**	**Armazenamento em Disco (GB)**	**Número de GFLOP**	**Custo Financeiro ($/seg)**
Small	7,50	500	38,40	0,0003
Standards	64,00	500	153,60	0,0007
Intermediate	192,00	500	76,80	0,0016
Large	130,00	3000	150,72	0,0026
Extra Large	2048,00	500	153,60	0,0081

As informações dos *workflows* necessárias para a execução do modelo *CC-LB* e da heurística *GraspCC-LB* foram obtidas de *Pegasus Workflow Execution Traces* [da Silva et al., 2023], que pode ser obtido em `https://github.com/wfcommons/pegasus-instances`. Essas informações estão apresentadas na Tabela VI.3. As colunas desta tabela contêm o nome da instância, a quantidade de *layers* e tarefas, os requisitos de memória RAM, de armazenamento em disco, a quantidade de GFLOP, o custo financeiro e o tempo máximo de processamento impostos pelos clientes, e as máquinas disponibilizadas pelo provedor de serviço para a execução, respectivamente. Nos experimentos realizados o custo máximo (C_{max}) foi fixado em *$1000,00*.

Os *workflows* possuem parâmetros específicos que influenciam o número de tarefas, a quantidade de dados processados e gerados, e o tempo de processamento das tarefas. Assim, os nomes das instâncias identificam o *workflow* e seus parâmetros seguindo um formato pré-estabelecido. As instâncias do *workflow 1000Genomes* são nomeadas seguindo o formato `1000genome-<NUM_CH>ch-<NUM_SEQ>k`, onde `<NUM_CH>` é o número de cromossomos avaliados na execução do *workflow*, `<NUM_SEQ>` é o número de sequências por arquivo de cromossomo. O número de cromossomos impacta o número de tarefas no *workflow*, enquanto o número de sequências afeta a quantidade de dados manipulados (dados de entrada e saída) e o tempo de execução da tarefa.

Os nomes das instâncias do *workflow Epigenomics* seguem o formato `epigenomics-<DATASET>-<NUM_SEQ>seq-<BIN_SIZE>k`, onde `<DATASET>` representa o conjunto de dados de entrada (por exemplo, *hep* e *ilmn*), `<NUM_SEQ>` é o número de arquivos processados pelo *workflow*, e `<BIN_SIZE>` é a quantidade de informações de sequências de DNA e proteínas a serem processadas por cada tarefa computacional.

Finalmente, as instâncias do *workflow Montage* usam a convenção `montage-<DATASET>-<DEGREE>d`, onde `<DATASET>` é o conjunto de dados de entrada a ser usado para o mosaico (por exemplo, *2mass* e *dss*), e `<DEGREE>` é o tamanho em graus a ser usado para a largura e altura do mosaico final. O conjunto de dados (`DATASET`) e o tamanho em graus (`DEGREE`) determinam o número de tarefas no

Tabela VI.3: Descrição das instâncias dos *workflows*

Instância	**Qtd. de *Layers***	**Qtd. de Tarefas**	**RAM (GB)**	**Disco (GB)**	**Qtd. de GFLOP**	r_{max}	**Maquinas**
1000genome-02ch-100k	3	52	8,40	19,42	106.417,72	60	*Intermediate Extra Large*
1000genome-06ch-250k	3	246	61,33	359,34	1.593.343,60	900	*Small Standard Intermediate Extra Large*
1000genome-14ch-250k	3	574	143,10	841,10	3.488.393,97	900	*Small Standard Intermediate Extra Large*
1000genome-22ch-250k	3	902	224,87	1.319,57	5.548.133,64	900	*Small Standard Intermediate Extra Large*
epigenomics-hep-1seq-50k	9	41	2,17	1,21	20.709,38	60	*Small Intermediate*
epigenomics-ilmn-1seq-50k	9	241	12,29	5,59	162.445,13	900	*Small Intermediate*
epigenomics-ilmn-4seq-50k	9	1.095	55,21	25,38	953.679,21	900	*Small (4) Intermediate*
epigenomics-ilmn-6seq-50k	9	1.695	85,38	39,22	1.188.018,74	900	*Small (4) Intermediate*
montage-dss-05d	8	58	0,58	9,25	214.495,14	60	*Extra Large Standard*
montage-dss-05d-b1			0,20	3,16	84.868,80		
montage-dss-05d-b2	8	20	0,19	3,03	53.940,41	60	*Extra Large Standard*
montage-dss-05d-b3			0,19	3,06	75.685,63		
montage-2mass-02d	8	1.444	103,64	14,50	53.771,60	900	*Large (2)*
montage-2mass-05d	8	4.572	300,16	40,84	251.735,20	900	*Large (2)*
montage-2mass-06d	8	6.824	451,22	59,22	369.492,00	900	*Large (2)*

workflow.

VI.1 Comparação dos resultados do modelo exato *CC-LB* com a heurística *GraspCC-LB*

A Tabela VI.4 apresenta os resultados obtidos pelo modelo matemático *CC-LB*, bem como a solução fornecida pela heurística *GraspCC-LB*, descritos nos Capítulos IV e V, respectivamente. A primeira coluna contém os nomes das instâncias do problema, conforme descritas na Tabela VI.3. Na coluna *CC-LB*, para cada instância, é reportado o valor do custo financeiro, o tempo em minutos e o tempo computacional obtidos pelo modelo matemático. Na coluna *GraspCC-LB*, além das mesmas informações reportadas para o *CC-LB*, é apresentado o *Gap* entre as soluções da heurística e do modelo matemático. Por fim, os pesos α_1 e α_2 atribuem a importância dos objetivos

de custo financeiro e de tempo de contratação definidos pelo usuário, respectivamente.

A variação das execuções estão relacionadas em como os pesos afetam os objetivos: *(i)* $\alpha_1 = 0,5$ e $\alpha_2 = 0,5$, objetivo considera de maneira igualitária o custo financeiro e tempo de execução; *(ii)* $\alpha_1 = 1$ e $\alpha_2 = 0$, objetivo privilegia o custo financeiro; e *(iii)* $\alpha_1 = 0$ e $\alpha_2 = 1$, objetivo privilegia o tempo de execução.

O cálculo do *Gap* é apresentado na Equação VI.1, onde o FO_d e FO_r são, respectivamente, o valor calculado da função objetivo da heurística e do modelo matemático exato. É importante destacar que todos os resultados apresentados nas tabelas foram arredondados para duas casas decimais.

$$Gap\ (\%) = \frac{FO_d - FO_r}{FO_r} \times 100 \tag{VI.1}$$

A partir dos resultados apresentados na Tabela VI.4, pode-se notar que o modelo `CC-LB` gerou resultados ótimos em 78% das execuções. Resultados marcados com (*) indicam que o modelo não atingiu a otimalidade dentro do limite de tempo estabelecido (86.400 segundos) para essas instâncias, ou seja, não encontrou soluções ótimas dentro do prazo. Em duas execuções das instâncias *montage-dss-05d*, o modelo não conseguiu encontrar uma solução viável (indicado por "-"). Assim, considerando os resultados nos quais o modelo retornou uma solução ótima, foi feita uma comparação com os resultados gerados pela heurística proposta. A `GraspCC-LB` atingiu o resultado ótimo como o modelo em 50% dos cenários avaliados e encontrou soluções para instâncias que o modelo não conseguiu encontrar.

Além disso, em todas as instâncias avaliadas, o tempo computacional da heurística *GraspCC-LB* foi inferior ao do modelo *CC-LB*. Observou-se também que a *GraspCC-LB* não alcançou a otimalidade em 7 das 12 variações do *workflow Montage*. Uma conclusão que se pode extrair é que a heurística *GraspCC-LB* apresenta melhor desempenho em instâncias maiores do que em menores. Além do mais, o *Montage* inclui várias atividades que não podem ser paralelizadas, característica não observada nos outros *workflows* avaliados.

VI.2 Avaliação de diferentes estratégias de agrupamento de tarefas geradas pela heurística *GraspCC-LB*

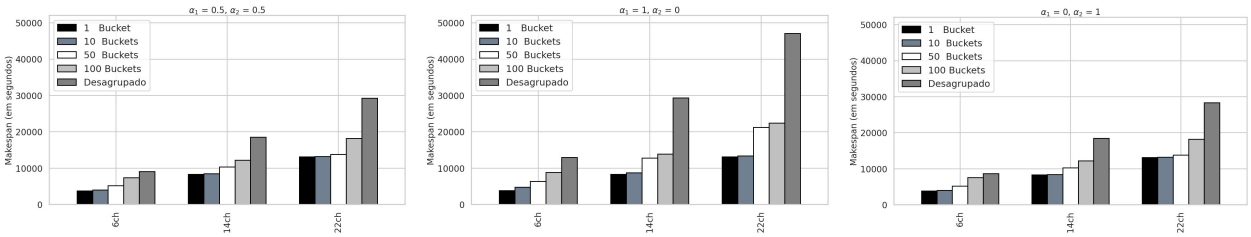
Nesta seção, avaliamos o impacto de diferentes estratégias de agrupamento de tarefas em diferentes *buckets* sobre a estimativa de recursos gerada pela heurística *GraspCC-LB* para as instâncias de *1000Genome*, *Epigenomics*, e *Montage*. Cinco estratégias de agrupamento de tarefas foram analisadas: a abordagem desagrupada, na qual cada *bucket* contém apenas uma tarefa, e abordagens com diferentes números de *buckets* (1, 10, 50, e 100) por *layer*, com pesos variáveis α_1 e α_2 para

Tabela VI.4: Resultados da Formulação Matemática *CC-LB* e da Heurística *GraspCC-LB*

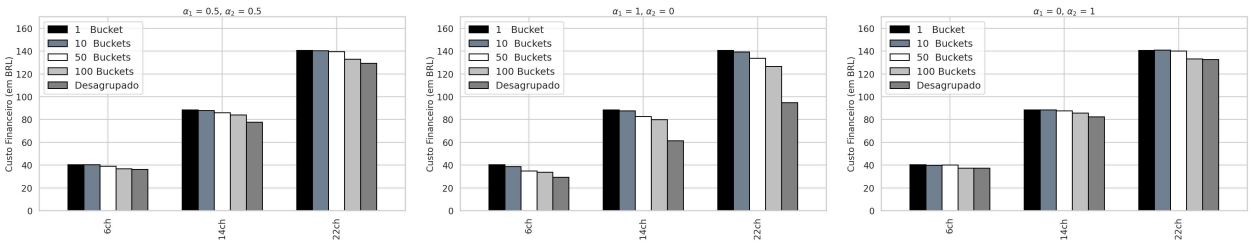
Instâncias	*CC-LB*			*GraspCC-LB*				α_1	α_2
	Valor da Solução		Tempo Total (s)	Valor da Solução		Tempo Total (s)	*Gap* (%)		
	Custo ($)	Tempo (min)		Custo ($)	Tempo (min)				
1000genome-2ch-100k	15,13	26	2728,13	15,13	26	4,91	0	0,5	0,5
	4,99	60	173,05	4,99	52	6,45	0	1	0
	15,13	26	3686,17	15,13	26	5,27	0	0	1
epigenomics-hep-1seq-50k	1,99	25	357,32	1,99	25	3,94	0	0,5	0,5
	1,59*	30	86400,00	1,59	30	4,82	0	1	0
	2,57	25	162,65	2,30	25	4.02	0	0	1
montage-dss-05d	-	-	86400,00	17,44	45	6,16	-	0,5	0,5
	15,22*	50	86400,00	8,56	65	5,76	0	1	0
	-	-	86400,00	20,11	45	6,03	-	0	1
montage-dss-05d-b1	6,06	17	34,53	5,17	19	1,94	10,15 ↑	0,5	0,5
	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
montage-dss-05d-b2	5,00	15	67,76	5,00	15	1,93	0	0,5	0,5
	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
montage-dss-05d-b3	5,57	17	41,67	5,13	18	1,94	5,35 ↑	0,5	0,5
	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

otimizar custo e tempo total de execução, respectivamente. Como mencionado anteriormente, os *buckets* foram criados de forma a equilibrar os requisitos de GFLOPs de acordo com as tarefas que os compõem.

As Figuras VI.1, VI.2 e VI.3 apresentam os resultados obtidos pela heurística *GraspCC-LB* para os *workflows 1000Genome*, *Epigenomics* e *Montage*, respectivamente. As figuras mostram especificamente o tempo total de execução (*makespan*) *(a)* e o custo financeiro *(b)* da estimativa de recursos fornecida pela heurística para cada estratégia de agrupamento e variação de α_1 e α_2.



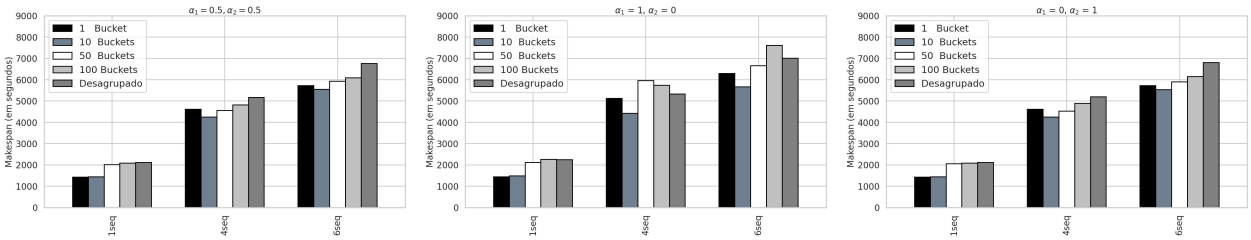
(a) *Makespan*



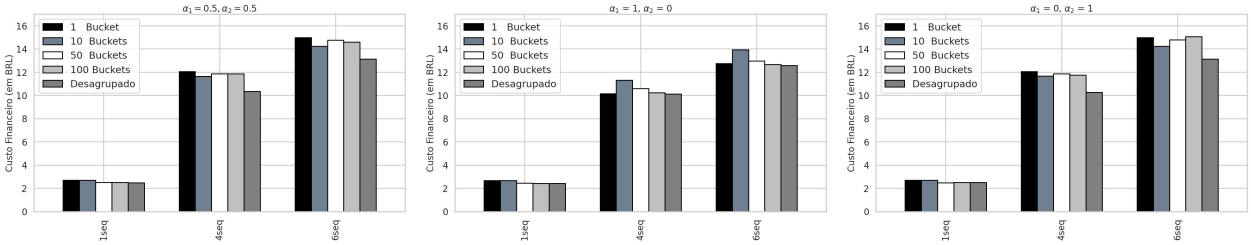
(b) *Custo Financeiro*

Figura VI.1: Solução *GraspCC-LB* para instâncias do *1000Genome* com diferentes estratégias de agrupamento e variações de α_1 e α_2: (a) *Makespan* e (b) *Custo Financeiro*

Os resultados indicaram que utilizar diferentes estratégias de agrupamento teve um impacto significativo no *makespan*. Em geral, observa-se que o tempo de execução se eleva com o aumento do



(a) *Makespan*



(b) *Custo Financeiro*

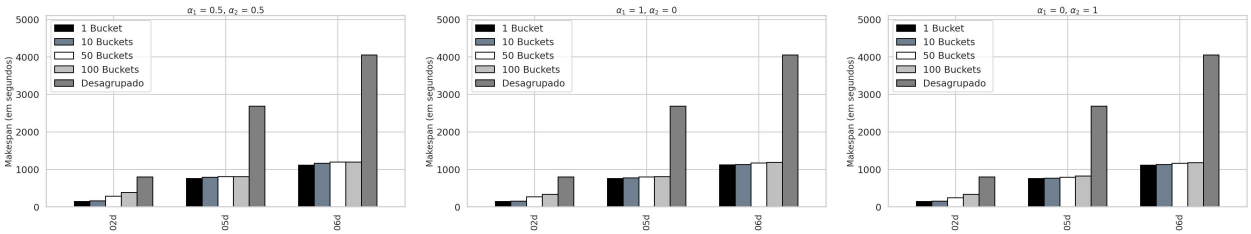
Figura VI.2: Solução *GraspCC-LB* para instâncias do *Epigenomics* com diferentes estratégias de agrupamento e variações de α_1 e α_2: (a) *Makespan* e (b) *Custo Financeiro*

número de *buckets*, ou seja, quando menos tarefas são agrupadas em cada *bucket*. Por consequência, a implementação de agrupamentos conseguiu reduzir o *makespan* para todos os *workflows*. Contudo, o comportamento do custo financeiro mostrou variações conforme o cenário analisado.

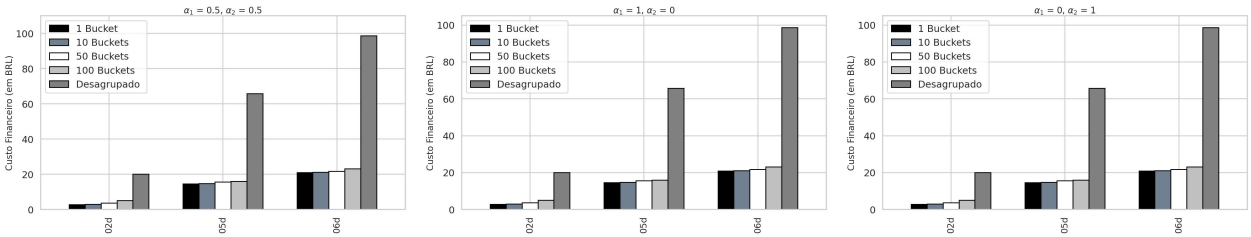
Nas instâncias dos *workflows 1000Genome* e *Epigenomics*, verifica-se uma redução no custo financeiro com o aumento do número de *buckets*. Esse comportamento ocorre porque a heurística seleciona as máquinas com os menores custos por unidade de tempo, as quais geralmente possuem capacidades de processamento inferiores. Essa configuração com mais *buckets* resulta em uma demanda reduzida por capacidade de processamento em GFLOPs, devido ao menor número de tarefas agrupadas.

Por outro lado, nas instâncias de *Montage*, tanto o *makespan* quanto o custo aumentam à medida que o número de *buckets* aumenta. Isso acontece porque as máquinas disponíveis para essas instâncias têm configurações e preços idênticos. Portanto, aumento do custo está diretamente relacionado ao acréscimo no tempo total de execução, influenciado pelo maior número de *buckets*.

Além disso, observa-se que agrupamentos com um único *bucket* se mostraram mais eficazes na maioria das instâncias dos *workflows* avaliados. Para as instâncias do *1000Genome*, esse tipo de agrupamento apresentou melhores resultados em todas as variações dos pesos α_1 e α_2. Da mesma forma, no *workflow Montage*, o agrupamento com um *bucket* também foi o que obteve o melhor resultado em todas as avaliações. Por outro lado, nas instâncias do *Epigenomics*, um agrupamento com 10 *buckets* provou ser mais eficiente, particularmente quando os pesos α_1 e α_2 estão configurados de maneira equilibrada ($\alpha_1 = 0,5$ e $\alpha_2 = 0,5$) ou com foco na redução do *makespan* ($\alpha_1 = 0$ e $\alpha_2 = 1$).



(a) *Makespan*



(b) *Custo Financeiro*

Figura VI.3: Solução *GraspCC-LB* para instâncias do *Montage* com diferentes estratégias de agrupamento e variações de α_1 e α_2: (a) *Makespan* e (b) *Custo Financeiro*

Por fim, o tempo de processamento da heurística GrapsCC-LB para encontrar soluções para as instâncias mencionadas também foi avaliado. Como esperado, observou-se que o tempo de processamento da heurística foi maior para configurações com mais *buckets* (*e.g.*, abordagem desagrupada e com agrupamentos maiores que 50 *buckets*). Por exemplo, considerando o cenário configurado de maneira equilibrada ($\alpha_1 = 0,5$ e $\alpha_2 = 0,5$), o tempo de processamento da heurística variou de $1,99$ a $144,28$[2], de $2,06$ a $353,02$ e de $0,52$ a $322,00$ segundos para as maiores instâncias de cada *workflow 1000Genome*, *Epigenomics* e *Montage*, respectivamente. Observa-se que, para os agrupamentos com um único *bucket*, o tempo de processamento da heurística é significativamente menor, cerca de 98%, em comparação com a abordagem desagrupada em todas as instâncias dos *workflows* avaliados. Esse padrão indica que a segmentação em muitos *buckets* aumenta a complexidade do problema e penaliza o tempo necessário para que o *GraspCC-LB* encontre uma solução.

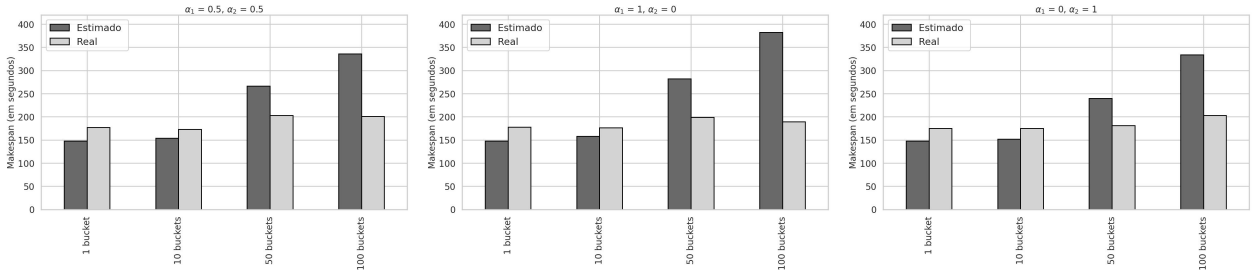
VI.3 Avaliação da estimativa de recursos produzida pela heurística *GraspCC-LB* em um ambiente real

Nesta seção, foi avaliada a estimativa de recursos gerada pela heurística *GraspCC-LB* ao executar o *workflow Montage* em um ambiente real, descrito no início deste capítulo. As estratégias de agrupamento de tarefas (*i.e.*, o número de *buckets*) e os pesos α_1 e α_2 variaram durante os testes. Considerando que o *makespan* e o custo financeiro para o *workflow Montage* na estratégia desagrupada (Seção VI.2) foram consideravelmente superiores aos das outras abordagens, a avaliação em ambiente de execução real limitou-se a quatro estratégias de agrupamento: 1, 10, 50 e 100 *buckets*

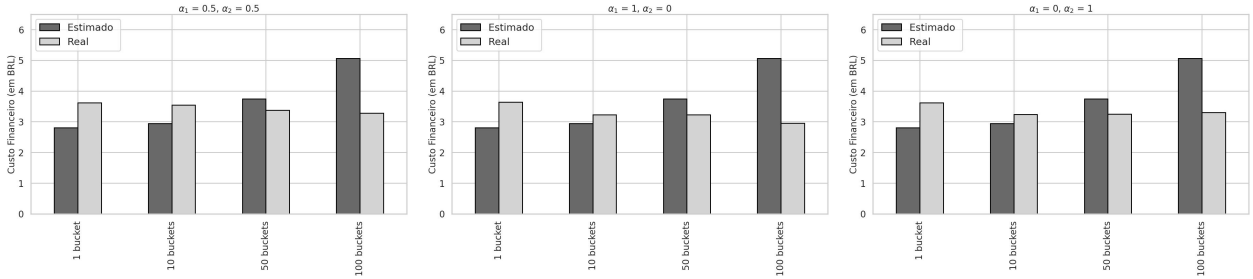
[2]Em todos os casos, o tempo mais curto refere-se à abordagem com um *bucket*, e o tempo mais longo refere-se à abordagem desagrupada.

por *layer*.

As Figuras VI.4, VI.5 e VI.6 apresentam os resultados para as instâncias `montage-2mass-02d`, `montage-2mass-05d` e `montage-2mass-06d`, respectivamente. Observa-se na Figura VI.4 uma tendência de aumento tanto no *makespan* quanto no custo financeiro em configurações a partir de 50 *buckets*. Isso pode ocorrer devido às características da instância, como o número de tarefas a serem processadas e a baixa demanda de processamento de algumas tarefas, mesmo quando agrupadas. Consequentemente, tarefas estimadas para uma janela de tempo específica são executadas mais rapidamente no ambiente real. A janela de tempo não é totalmente utilizada para essa tarefa, permitindo que outra tarefa inicie a execução dentro da mesma janela, o que pode resultar em um menor *makespan* para execuções reais. Além disso, foi possível notar que os resultados reais para a estratégia de agrupamento com 10 *buckets* foram os mais próximos dos estimados, apresentando, aproximadamente, um *makespan* 11% a 13% maior e um custo de 3% a 9% maior.



(a) *Makespan*

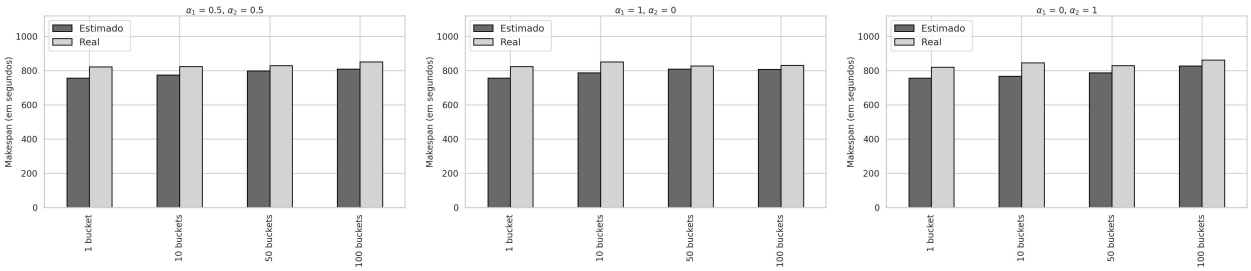


(b) *Custo Financeiro*

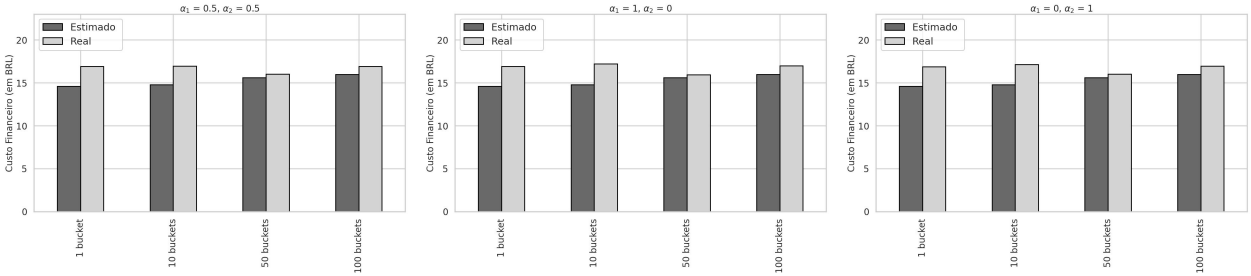
Figura VI.4: Estimado *versus* Real para a instância `montage-2mass-2d` com diferentes estratégias de agrupamento de tarefas e variações de α_1 e α_2: (a) *Makespan* e (b) *Custo Financeiro*

As instâncias `montage-2mass-05d` e `montage-2mass-06d` exibem comportamentos semelhantes, conforme apresentados nas Figuras VI.5 e VI.6, respectivamente. Os valores para as execuções reais se aproximam dos estimados à medida que o número de *buckets* aumenta. Especificamente, o *makespan* e o custo financeiro das execuções reais usando a estratégia de agrupamento com 50 *buckets* na `montage-2mass-05d` e com 100 *buckets* na `montage-2mass-06d` são os mais próximos dos valores estimados, com diferenças de $3,37\%$ e $2,58\%$, respectivamente.

É importante notar que no ambiente real foi impossível configurar a execução de forma que uma máquina fosse dedicada a executar um número específico de tarefas. Em outras palavras, uma vez



(a) *Makespan*

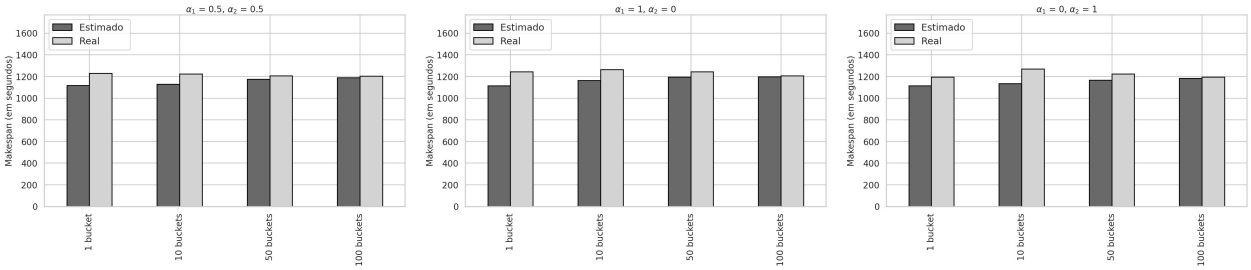


(b) *Custo Financeiro*

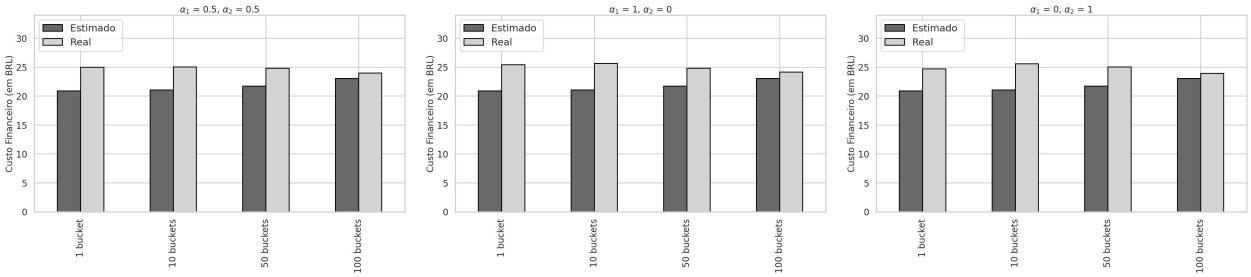
Figura VI.5: Estimado *versus* Real para a instância `montage-2mass-5d` com diferentes estratégias de agrupamento de tarefas e variações de α_1 e α_2: (a) *Makespan* e (b) *Custo Financeiro*

que uma tarefa é agendada pelo gerenciador *Ray* para ser executada em uma máquina, ela pode começar sua execução assim que o recurso estiver disponível e suas dependências satisfeitas. Portanto, as tarefas no ambiente real não serão necessariamente executadas dentro das mesmas janelas de tempo definidas pela solução heurística. Essa discrepância pode ajudar a explicar o *makespan* menor e os custos mais baixos observados nas execuções reais da instância `montage-2mass-02d` à medida que mais *buckets* são considerados. Os resultados indicam que, dependendo do cenário avaliado, as estratégias de agrupamento — excluindo as estratégias de um único *bucket* e desagrupadas — tendem a ser mais eficientes no uso dos recursos das máquinas.

Um desvio absoluto médio de $29,92\%$, $7,21\%$ e $8,81\%$ foi observado entre o valor da função objetivo das execuções reais e os estimados pela heurística para as instâncias `montage-2mass-02d`, `montage-2mass-05d` e `montage-2mass-06d`, respectivamente; considerando um cenário equilibrado entre *makespan* e custo financeiro, com pesos $\alpha_1 = 0,5$ e $\alpha_2 = 0,5$. O desvio absoluto médio entre as três instâncias do *Montage* é de $15,31\%$, refletindo a superestimação observada na instância `montage-2mass-02d` para agrupamentos de 50 e 100 *buckets*.



(a) *Makespan*



(b) *Custo Financeiro*

Figura VI.6: Estimado *versus* Real para a instância `montage-2mass-6d` com diferentes estratégias de agrupamento de tarefas e variações de α_1 e α_2: (a) *Makespan* e (b) *Custo Financeiro*

Capítulo VII Considerações Finais

Diversos experimentos científicos utilizam aplicações que demandam elevada capacidade de processamento computacional, as quais manipulam volumes substanciais de dados. Estes experimentos são, com frequência, estruturados como *workflows* científicos e processados de maneira paralela em ambientes de HPC, tais como infraestruturas de nuvem e *clusters* computacionais.

Atualmente, uma ampla gama de Sistemas de Gerenciamento de *Workflows* Científicos (SWfMS) disponibiliza suporte para execução destes *workflows* em ambientes de nuvem e *clusters* computacionais. A ênfase desses sistemas geralmente recai sobre a otimização de políticas de escalonamento de tarefas e alocação de dados.

Neste estudo, argumentou-se que o dimensionamento apropriado do arranjo computacional é uma etapa fundamental para execução de *workflows* científicos em ambientes de alto desempenho. Este aspecto é primordial, uma vez que o desempenho geral do sistema e os custos financeiros associados podem ser adversamente afetados se o ambiente estiver subdimensionado (resultando em gargalos de desempenho de tempo) ou superdimensionado (levando ao uso ineficiente de recursos e custos financeiros desnecessários). Assim, introduzimos uma heurística para o dimensionamento de *workflows* científicos em ambientes HPC denominada *GraspCC-LB*. Essa abordagem, adaptada do algoritmo *GraspCC*, proposto por Coutinho et al. [2015], visa a minimização dos custos financeiros e do tempo de processamento, associados à execução do *workflow*, respeitando as restrições impostas pelos clientes e as limitações do provedor de serviços. A abordagem utiliza uma estratégia chamada *Layered-Bucket*, que explora várias estratégias de agrupamento de tarefas em *buckets* baseados em seus níveis no *workflow*. Esta estratégia visa encontrar uma estimativa de recursos eficiente.

Três diferentes experimentos foram conduzidos usando dados históricos de execuções reais de *workflows* científicos. O primeiro experimento comparou a heurística *GraspCC-LB* com o modelo matemático *CC-LB*, que fornece soluções ótimas. O segundo experimento avaliou *GraspCC-LB* sob diferentes estratégias de agrupamento de tarefas em *buckets*. Finalmente, o terceiro experimento avaliou a estimativa de recursos gerada por *GraspCC-LB* em um ambiente de execução real para testar sua aplicabilidade prática. Os resultados experimentais mostram que o modelo exato *CC-LB* alcançou a otimalidade em 78% dos cenários, mas em alguns casos, não conseguiu alcançar a otimalidade dentro do limite de tempo estabelecido (*i.e.*, 24 horas). Em contraste, a heurística *GraspCC-LB* encontrou soluções em instâncias onde o modelo exato falhou e obteve os mesmos

resultados ótimos que o modelo exato em 50% dos casos avaliados. Além disso, *GraspCC-LB* exigiu significativamente menos tempo computacional em todas as execuções.

Quanto ao impacto de diferentes estratégias de agrupamento de tarefas pela *GraspCC-LB*, os experimentos mostraram que a configuração do agrupamento de tarefas em *buckets* influencia significativamente o *makespan* e os custos financeiros associados. Cenários com menos *buckets* tendem a reduzir o *makespan*, mas podem aumentar os custos financeiros ao usar recursos computacionais mais custosos. Por outro lado, configurações com mais *buckets* aumentam o *makespan*, mas reduzem os custos usando recursos menos dispendiosos. Nos *workflows 1000Genome* e *Epigenomics*, o aumento de *buckets* geralmente resultou em custos financeiros mais baixos devido à seleção de máquinas menos dispendiosas. No entanto, no *workflow Montage*, mais *buckets* resultaram em custos financeiros e *makespan* mais altos devido às características específicas dos recursos computacionais disponíveis para execução.

Em execuções no mundo real, observou-se que a menor instância do *workflow Montage*, quando configurada com 50 *buckets* ou mais, experimentou aumentos tanto no *makespan* quanto nos custos financeiros. Ao contrário, quando configurada com 50 e 100 *buckets*, as instâncias maiores mostraram pequenas diferenças — menos de 5% em média — entre a solução estimada e a execução real. Esses resultados sugerem que o número ótimo de *buckets* varia significativamente entre diferentes tipos e instâncias de *workflows*. Portanto, os resultados indicam que a heurística *GraspCC-LB* pode ser uma abordagem eficaz para o dimensionamento de recursos em ambientes HPC, oferecendo uma alternativa viável em situações onde modelos matemáticos exatos são impraticáveis devido a limitações de tempo ou complexidade computacional.

Em 2023, os resultados preliminares deste trabalho foram apresentados no Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)[1], no artigo intitulado "*GraspCC-LB: Dimensionamento de Recursos para Execução de Workflows em Ambientes de Computação de Alto Desempenho*" [Alvarenga et al., 2023].

Para trabalhos futuros, propõe-se avaliar a heurística *GraspCC-LB* em novos cenários, utilizando outros *workflows* e diferentes configurações de ambientes computacionais (HPC), além de analisar o impacto de diferentes estratégias de agrupamento de tarefas nesses cenários. Além disso, planeja-se adaptar a heurística *GraspCC-LB* para generalizar a *GraspCC*, incorporando a estruturação do *workflow* em *layers* e *buckets*, eliminando, assim, a necessidade da fase de pré-processamento. Por fim, sugere-se explorar possíveis integrações com técnicas de aprendizado de máquina para predição dinâmica de recursos, comparando-as com outras abordagens existentes.

[1] `https://cradrs.github.io/wscad2023/`

Referências

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

Adhikari, M., Amgoth, T., and Srirama, S. A survey on scheduling strategies for workflows in cloud environment and emerging trends. *ACM Computing Surveys*, 52, 2019.

Alkhanak, E. N., Lee, S. P., Rezaei, R., and Parizi, R. M. Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. *Journal of Systems and Software*, 113:1–26, 2016.

Alvarenga, L. C. R., Frota, Y., de Oliveira, D., and Coutinho, R. Graspcc-lb: Dimensionamento de recursos para execução de workflows em ambientes de computação de alto desempenho. In *Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD)*, pages 145–156. SBC, 2023.

Baskiyar, S. and Abdel-Kader, R. Energy aware dag scheduling on heterogeneous systems. *Cluster Computing*, 13:373–383, 2010.

Berriman, G. B., Deelman, E., Good, J. C., Jacob, J. C., Katz, D. S., Kesselman, C., Laity, A. C., Prince, T. A., Singh, G., and Su, M.-H. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In Quinn, P. J. and Bridger, A., editors, *Optimizing Scientific Return for Astronomy through Information Technologies*, volume 5493, pages 221 – 232. International Society for Optics and Photonics, SPIE, 2004.

Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., and Vahi, K. Characterization of scientific workflows. In *2008 third workshop on workflows in support of large-scale science*, pages 1–10. IEEE, 2008.

Buyya, R. *High Performance Cluster Computing: achitectures and systems*, volume 2. Pearson Education India, 1999.

Consortium, . G. P. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.

Coutinho, R., Drummond, L., and Frota, Y. Optimization of a cloud resource management problem from a consumer perspective. In *Euro-Par 2013: Parallel Processing Workshops*, pages 218–227, Berlin, Heidelberg. Springer Berlin Heidelberg, 2013.

Coutinho, R., Drummond, L., Frota, Y., and de Oliveira, D. Optimizing virtual machine allocation for parallel scientific workflows in federated clouds. *Future Generation Computer Systems*, 46:51–68, 2015.

da Silva, R. F., Shah, H., and Casanova, H. wfcommons/pegasus-instances: v1.4, 2023.

de la Torre, L. and Halappanavar, M. Scaling optimal allocation of cloud resources using lagrange relaxation. In Klusáček, D., Corbalán, J., and Rodrigo, G. P., editors, *Job Scheduling Strategies for Parallel Processing*, pages 173–192, Cham. Springer Nature Switzerland, 2023.

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

Deelman, E., Gannon, D., Shields, M., and Taylor, I. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

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

Dongarra, J., Foster, I., Fox, G., Gropp, W., Kennedy, K., Torczon, L., and White, A. *Sourcebook of parallel computing*, volume 3003. Morgan Kaufmann Publishers San Francisco, 2003.

Durillo, J., Prodan, R., and Barbosa, J. Pareto tradeoff scheduling of workflows on federated commercial clouds. *Simulation Modelling Practice and Theory*, 58:95–111, 2015.

Feo, T. A. and Resende, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6:109–133, 1995.

Ferreira da Silva, R., Filgueira, R., Deelman, E., Pairo-Castineira, E., Overton, I. M., and Atkinson, M. P. Using simple pid-inspired controllers for online resilient resource management of distributed scientific workflows. *Future Generation Computer Systems*, 95:615–628, 2019.

Foster, I. and Kesselman, C. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

Geist, A. and Reed, D. A. A survey of high-performance computing scaling challenges. *The International Journal of High Performance Computing Applications*, 31(1):104–113, 2017.

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

Habib, S. M., Ries, S., and Mühlhäuser, M. Cloud computing landscape and research challenges regarding trust and reputation. page 410 – 415, 2010.

Hoefler, T., Gropp, W., Kramer, W., and Snir, M. Performance modeling for systematic performance tuning. In *State of the Practice Reports*, SC '11, New York, NY, USA. Association for Computing Machinery, 2011.

Jia, J., Liu, Y., Zhang, G., Gao, Y., and Qian, D. Software approaches for resilience of high performance computing systems: a survey. *Frontiers of Computer Science*, 17(4):174105, 2023.

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

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

Malawski, M., Figiela, K., Bubak, M., Deelman, E., and Nabrzyski, J. Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization. *Scientific Programming*, 2015:5, 2015.

Mell, P. and Grance, T. The nist definition of cloud computing, 2011.

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

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

Oda, R., Cordeiro, D., and Braghetto, K. R. Dynamic resource provisioning for scientific workflow executions in clouds. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 291–294, 2018.

Ogasawara, E. Uma abordagem algébrica para workflows científicos com dados em larga escala. *Universidade Federal do Rio de Janeiro*, 2011.

Oinn, T., Greenwood, M., Addis, M., Alpdemir, M. N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., et al. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and computation: Practice and experience*, 18(10):1067–1100, 2006.

Pietri, I. and Sakellariou, R. A pareto-based approach for cpu provisioning of scientific workflows on clouds. *Future Generation Computer Systems*, 94:479–487, 2019.

Ramamurthy, R., Balaji, P., and Prabhu, N. Multi-objective optimization for virtual machine allocation in computational scientific workflows under uncertainty. *Journal of Parallel and Distributed Computing*, 141:35–48, 2020.

Rosa, M. J., Ralha, C. G., Holanda, M., and Araujo, A. P. Computational resource and cost prediction service for scientific workflows in federated clouds. *Future Generation Computer Systems*, 125:844–858, 2021.

Russell, N., ter Hofstede, A. H. M., Edmond, D., and van der Aalst, W. M. P. Workflow data patterns: Identification, representation and tool support. In Delcambre, L., Kop, C., Mayr, H. C., Mylopoulos, J., and Pastor, O., editors, *Conceptual Modeling – ER 2005*, pages 353–368, Berlin, Heidelberg. Springer Berlin Heidelberg, 2005.

Rynge, M., Juve, G., Kinney, J., Good, J., Berriman, B., Merrihew, A., and Deelman, E. Producing an infrared multiwavelength galactic plane atlas using montage, pegasus, and amazon web services. *Astronomical Data Analysis Software and Systems XXIII*, 485:211, 2014.

Shalf, J., Dosanjh, S., and Morrison, J. Exascale computing technology challenges. In *High Performance Computing for Computational Science–VECPAR 2010: 9th International conference, Berkeley, CA, USA, June 22-25, 2010, Revised Selected Papers 9*, pages 1–25. Springer Berlin Heidelberg, 2011.

Shealy, B. T., Feltus, F. A., and Smith, M. C. Intelligent resource provisioning for scientific workflows and hpc. In *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*, pages 9–16, 2021.

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

Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

Xu, J., Wu, Y., Li, F., and Zhao, Y. Dynamic resource provisioning with fault tolerance for data-intensive meteorological workflows in cloud. *IEEE Transactions on Industrial Informatics*, 16(9):6174–6183, 2020.

Yu, J. and Buyya, R. A taxonomy of scientific workflow systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200, 2005.

Zakarya, M. and Gillam, L. Energy efficient computing, clusters, grids and clouds: A taxonomy and survey. *Sustainable Computing: Informatics and Systems*, 14:13–33, 2017.

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