UNIVERSIDADE FEDERAL FLUMINENSE

MURILO BRUGGER STOCKINGER

Escalonamento de Tarefas em CPU/GPU e Alocação de Arquivos de Dados de Workflows Científicos em Nuvens Computacionais

Niterói

2020

UNIVERSIDADE FEDERAL FLUMINENSE

MURILO BRUGGER STOCKINGER

Escalonamento de Tarefas em CPU/GPU e Alocação de Arquivos de Dados de Workflows Científicos em Nuvens Computacionais

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Algoritmos e Otimização.

Orientador:
Isabel Cristina Mello Rosseti

Co-orientador:
Alexandre Plastino de Carvalho

Niterói

2020

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

S864e Stockinger, Murilo Brugger
Escalonamento de Tarefas em CPU/GPU e Alocação de Arquivos de Dados de Workflows Científicos em Nuvens Computacionais / Murilo Brugger Stockinger ; Isabel Cristina Mello Rosseti, orientadora ; Alexandre Plastino de Carvalho, coorientador. Niterói, 2020.
68 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2020.

DOI: http://dx.doi.org/10.22409/PGC.2020.m.11361368780

1. Metaheurística Híbrida. 2. Produção intelectual.I. Rosseti, Isabel Cristina Mello, orientadora. II. Carvalho, Alexandre Plastino de, coorientador. III. Universidade Federal Fluminense. Instituto de Computação. IV. Título.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

MURILO BRUGGER STOCKINGER

Escalonamento de Tarefas em CPU/GPU e Alocação de Arquivos de Dados de Workflows Científicos em Nuvens Computacionais

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Algoritmos e Otimização.

Aprovada em 13/02/2020

BANCA EXAMINADORA

Prof^{a}. Isabel Cristina Mello Rosseti - Orientador, UFF
(Presidente)

Prof. Alexandre Plastino de Carvalho - Coorientador, UFF

Prof. Celso da Cruz Carneiro Ribeiro - Avaliador, UFF

Prof. Luidi Gelabert Simonetti - Avaliador, UFRJ

Niterói

2020

Resumo

Neste trabalho, propõe-se uma heurística híbrida baseada em GRASP e ILS para solucionar o Problema de Escalonamento de Tarefas em CPU/GPU e Alocação de Arquivos de Dados de *Workflows* Científicos em Nuvens Computacionais. Experimentos computacionais realizados indicaram uma melhora nos resultados obtidos, quando comparados aos resultados atingidos por um algoritmo evolutivo já existente na literatura, para a versão do problema que usa somente CPU, alcançando algumas novas soluções de melhor qualidade e melhores médias do que essa heurística estado-da-arte. Uma formulação matemática inédita é apresentada para o problema de escalonamento em ambientes híbridos que suportam GPU. Para esta versão, a mesma heurística híbrida proposta para a versão com uso exclusivo de CPU é utilizada, viabilizando a execução de instâncias da ordem de 25 tarefas, ou mais, quando o modelo exato não obteve solução em tempo viável.

Palavras-chave: Metaheurística Híbrida, Escalonamento, GRASP, ILS

Abstract

In this work, a new hybrid heuristic based on GRASP and ILS is proposed to solve the Task Scheduling and Data Files Allocation on Scientific Clouds Problem on CPU/GPU. Computational experiments showed improvement in solution quality when compared to the results of a the state-of-the-art evolutive algorithm in the literature for the CPU-only version of the problem, achieving, for a relevant number of instances, new best solutions and best average solutions. A unprecedented mathematical model is presented to solve a variation of the problem on hybrid systems that allow GPU processing. For this problem variant, a second version of the proposed heuristic is also used to solve the problem, where instances with 25 or more tasks been could be solved when the exact solver could not finish the executions within reasonable computational time.

Keywords: Hybrid Metaheuristic, Scheduling , GRASP, ILS

Sumário

1 Introdução **8**

2 Descrição do Problema e Revisão Bibliográfica **12**

2.1 Descrição do Problema 12

2.2 Revisão Bibliográfica 14

2.2.1 Heurísticas para WfCs 14

2.2.2 Heurísticas baseadas em Metaheurísticas para WfCs 15

2.2.3 Estratégias com Escalonamento de Tarefas e Alocação de Dados para WfCs 16

2.3 Problema de Escalonamento de Tarefas e Alocação de Arquivos em *Workflows* Científicos 17

2.3.1 Formulação Matemática 17

2.3.2 Heurística *Hybrid Evolutive Algorithm* 21

3 Heurística Proposta para o PEAW-CPU **25**

3.1 Metaheurísticas 25

3.1.1 GRASP 25

3.1.2 Iterated Local Search 26

3.1.3 GRASP-ILS 27

3.2 Heurística Proposta para o PEAW-CPU 29

3.2.1 Construtivo com Componente Aleatória 30

3.2.2 Busca Local 31

3.2.2.1 *Fix File Allocation* 33

3.2.2.2 *Relocate Job* 33

3.2.2.3 *Swap Machine* 34

3.2.2.4 *Swap Machine Pair* 35

3.2.3 Perturbação 36

4 Problema de Escalonamento de Tarefas e Armazenamento de Dados em Worflows Científicos - CPU/GPU 39

4.1 Trabalhos Relacionados 39

4.2 Formulação Matemática 40

4.3 Abordagem Heurística 47

4.4 GRASP-ILS para o PEAW-CPU/GPU 48

4.4.1 Construtivo com Componente Aleatória 48

4.4.2 Busca Local e Perturbação 50

5 Resultados Computacionais 51

5.1 Ajuste de Parâmetros 52

5.2 Testes Computacionais para o PEAW-CPU 56

5.3 Testes Computacionais para o PEAW-CPU/GPU 58

6 Conclusões e Trabalhos Futuros 64

Referências 66

Capítulo 1

Introdução

O escalonamento ou programação da produção é um dos problemas operacionais que visam planejar e controlar a execução das tarefas de produção e de serviços. Esse problema tem como finalidade determinar uma sequência factível de processamento de um conjunto de operações por um conjunto de recursos, ao longo de um intervalo de tempo, visando otimizar uma ou mais medidas de desempenho. Tais operações fazem parte das tarefas ou pedidos de clientes por produtos ou serviços. Além disso, nesse problema podem existir ainda restrições de precedência entre as operações e outras restrições específicas para um cenário.

O interesse pela investigação de problemas de escalonamento teve seu início juntamente com a era industrial e com o surgimento das indústrias de grande porte, visando o aumento na produção de bens de consumo, por meio da otimização das linhas de produção. Esses problemas encontram-se presentes em uma ampla gama de diferentes aplicações, incluindo-se o despacho de voos em aeroportos; a determinação da ordem de ancoração de navios em sistemas portuários; o tratamento de pacientes em hospitais; a distribuição de atividades a professores em escolas; as linhas de produção das fábricas, entre outras atividades.

Uma versão desse problema em Computação é o escalonamento de tarefas de uma aplicação computacional exigindo, muitas vezes, que tarefas sejam executadas em uma certa ordem, já que recursos criados ao longo da execução são utilizados como entrada de tarefas futuras. Em um ambiente onde o paralelismo de execução de tarefas é possível, o problema de otimização de escalonamento adquire outras proporções, já que taferas podem ser escalonadas em diversas máquinas e executadas concorrentemente. Vários fatores, como a capacidade espacial para armazenamento de arquivos criados por tarefas, o tempo de processamento de cada tarefa e, até mesmo, a ordem de execução das tarefas

computacionais, podem ter uma influência relevante no tempo final de um escalonamento criado. Ademais, as questões tornam-se imprescindíveis quando se trata do paradigma de programação distribuída, onde, além dos quesitos existentes para uma única máquina, os tempos de transferência entre cada nó de uma rede de computadores podem fazer parte do custo final de uma solução. Essas aplicações estão produzindo e consumindo um volume considerável de dados, fazendo com que problemas relacionados às limitações computacionais e ao gerenciamento de recursos sejam constantemente enfrentados pelos usuários [8].

Escalonamentos de processos são comumente representados como uma cadeia de aplicações, na qual a saída de um programa é a entrada para outro. Nesse contexto, os *Workflows* Científicos (WfCs) destacam-se como uma solução promissora para elaborar e gerenciar essas cadeias de aplicações. Um WfC é uma abstração que estrutura as etapas do experimento como um grafo, no qual os nós correspondem às atividades de processamento de dados e as arestas representam os fluxos de dados entre as atividades [28].

Problemas de escalonamento fazem parte de um conjunto generalizado de problemas de otimização combinatória que têm como encontrar uma solução ótima, seja com o objetivo de maximizar ou minimizar, para um problema sobre um conjunto finito de soluções. [11]. Existem vários modos de formular, por meio de modelos matemáticos, e resolver, de maneira exata, problemas de otimização combinatória, destacando-se programação linear e programação linear inteira. Na programação linear, todas as relações entre variáveis, a função objetivo e as restrições, são lineares, e as variáveis são contínuas, isto é, podem assumir valores reais. Problemas de programação linear podem ser resolvidos, por exemplo, por implementações computacionais do algoritmo Simplex. Já na programação inteira, diferentemente da programação linear, onde valores fracionários podem ser atribuídos às variáveis, as variáveis do problema assumem apenas valores inteiros. Destacam-se como algoritmos de resolução de problemas de programação inteira os métodos de planos de cortes, de *branch and bound*, de *branch and cut* e de *branch and price* [10, 29].

O escalonamento de dados e de tarefas em sistemas distribuídos é um tópico amplamente discutido nos ambientes de *grids* e *clusters* [17, 23, 36, 46]. Em Ranganathan et al. [36], por exemplo, várias heurísticas de escalonamento foram avaliadas em conjunto com heurística de replicação e movimentação de dados. A avaliação foi feita em um ambiente simulado de *grid* computacional e, segundo os autores, os resultados são importantes para o tratamento do escalonamento de tarefas e dados de forma conjunta. Propostas e abordagens heurísticas para o escalonamento de *workflows* científicos têm sido desenvolvidas

nos últimos anos [7, 14, 27, 32, 33, 44, 50]. No entanto, soluções que consideram tanto a distribuição dos dados, quanto a alocação de tarefas foram pouco exploradas, sendo investigadas apenas em Teylo et al. [45].

Para permitir a execução de WfCs em ambientes de nuvens computacionais é necessário escalonar cada tarefa que compõe o *workflow* para uma das máquinas virtuais disponíveis. Assim, de maneira geral, um algoritmo de escalonamento busca mapear tarefas aos recursos, de forma que os requisitos definidos pelo usuário, pelas aplicações ou pelo provedor, sejam atendidos [5]. O escalonamento de tarefas em recursos distribuídos é um problema $\mathcal{NP}$-Difícil [47] e há algumas características das nuvens computacionais que fazem com que esse processo seja ainda mais complexo.

Para problemas $\mathcal{NP}$-Difíceis, a obtenção de soluções exatas pode requerer algoritmos de enorme complexidade espacial e temporal. Dessa forma, a utilização de resolvedores exatos se torna impraticável quando os problemas passam a tomar proporções mais próximas da realidade. Assim sendo, torna-se evidente, nesse contexto, a necessidade de aplicação de algoritmos aproximados que, ou podem ser combinados com os métodos exatos para acelerar seu tempo de processamento, ou eventualmente utilizados de maneira isolada para se obter soluções aproximadas, ou até ótimas, para esses problemas.

Um dos objetivos deste trabalho é a extensão das investigações realizadas em Teylo et al. [45] sobre escalonamentos de WfCs em CPU com alocação de dados. O Problema de Escalonamento e Alocação de Arquivos de Dados de *Workflows* Científicos em Nuvens Computacionais (PEAW-CPU) foi apresentado pela primeira vez em [45]. Uma nova heurística, baseada nas metaheurísticas GRASP e ILS, é proposta neste trabalho para o PEAW-CPU.

O segundo objetivo deste trabalho é explorar um novo problema de escalonamento de WfCs que comporta o uso de GPU e CPU para a execução das tarefas do *workflow*. Para esse problema é definida uma formulação matemática, além de uma nova versão da heurística proposta para o problema anterior.

Essa dissertação é composta dos seguintes capítulos. No Capítulo 2, é feita a revisão bibliográfica do Problema de Escalonamento de Tarefas e Alocação de Arquivos em *Workflows* Científicos com o uso de CPU, em conjunto com o modelo matemático e a heurística proposta em Teylo et al. [45]. No Capítulo 3, são apresentadas as metaheurísticas genéricas GRASP e ILS utilizadas como base para a construção da heurística proposta nesse trabalho e, em seguida, a heurística, baseada em GRASP-ILS, construída e suas componentes. No Capítulo 4, é definido um novo Problema de Escalonamento de

Tarefas e Alocação de Arquivos em *Workflows* Científicos com o uso de CPU e GPU, sua formulação matemática e a heurística GRASP-ILS, baseada na versão para o PEAW-CPU, para este problema. No Capítulo 5, estão os resultados computacionais das execução de ambas as heurísticas propostas neste trabalho e, por fim, no Capítulo 6, estão as conclusões e trabalhos futuros.

Capítulo 2

Descrição do Problema e Revisão Bibliográfica

2.1 Descrição do Problema

Segundo Yu et al. [49], o escalonamento de tarefas é o processo de escolha da ordem de execução dessas tarefas em um processador computacional. Dessa forma, é função do escalonador de processos escolher o momento e a forma de execução de uma tarefa, de acordo com o contexto do sistema no momento da execução. Esse processo se torna ainda mais importante quando se trata de um sistema distribuído de máquinas.

Em geral, o problema de escalonamento é, formalmente, composto por um conjunto de tarefas N = $\{tf_1, tf_2, \ldots, tf_n\}$ e um conjunto de máquinas M = $\{mvs_1, mvs_2, \ldots, mvs_m\}$. O escalonador deve, então, minimizar uma função objetivo f que qualifica uma solução, conforme um conjunto de restrições. Caso alguma restrição não seja satisfeita, a solução deve ser descartada. Em um problema de escalonamento de WfCs, a função objetivo é baseada na redução do tempo total de execução, chamado *makespan*, desse escalonamento, na atenuação do custo do uso de máquinas virtuais e no balanceamento do armazenamento de arquivos gerados pelo *workflow*.

É amplamente conhecido que, para problemas da classe $\mathcal{NP}$-Difícil, a busca por uma solução ótima de forma exata é impraticável devido ao tempo necessário para os resolvedores encontrarem uma solução exata e ao uso intenso de memória ocasionado pelo número de variáveis e restrições. Devido a esse fato, a geração de heurísticas baseadas em metaheurísticas para resolver esses problemas é amplamente utilizada na literatura. Metaheurísticas são métodos heurísticos genéricos para resolução de problemas computacionalmente difíceis na área de otimização combinatória. O uso desses métodos se

torna interessante pela forma como a busca de soluções é realizada. Os algoritmos metaheurísticos são, como o nome sugere, de natureza heurística. Esse fato os distingue de métodos exatos, que desenvolvem uma prova de que a solução ótima será encontrada em uma quantidade de tempo finita (embora muitas vezes proibitivamente grande). As metaheurísticas são, portanto, desenvolvidas especificamente para encontrar uma solução que seja "boa o suficiente"em um tempo de computação que seja "pequeno o suficiente"[21]. Como resultado, as metaheurísticas não estão sujeitas à explosão combinatória, isto é, o fenômeno de crescimento do tempo e do espaço computacional necessários para alcançar uma das soluções ótimas de um problema, que aumenta como uma função exponencial do tamanho do problema.

Como mencionado em Fakhfakh et al. [19], algoritmos de escalonamento podem ser estáticos ou dinâmicos. Nos algoritmos de escalonamento estático, todos os dados sobre o *workflow*, como o tempo de execução de cada tarefa, o grafo de dependências, a lista e o tamanho de arquivos produzidos, devem ser conhecidos *a priori*. Dessa forma, o escalonador analisa e constrói uma solução sobre as restrições pré-estabelecidas, buscando a minimização do *makespan* total. Não é possível, contudo, uma variação desses valores previamente inseridos, não havendo a possibilidade de o algoritmo recalcular alguma escolha de escalonamento devido a alguma mudança no sistema de máquinas. Um fator importante para algoritmos de escalonamento estático é a confiabilidade dos dados de entrada, uma vez que, caso as previsões de dados sejam equivocadas, o escalonamento proposto pode não ter a repercussão esperada em um cenário real.

Já nos algoritmos de escalonamento dinâmico, o escalonamento é feito durante a execução do *workflow*. O escalonador monitora a disponibilidade de tarefas e do sistema em tempo real e, caso existam tarefas disponíveis e máquinas ociosas aptas a receber a tarefa, escalona as execuções. Diferentemente do escalonamento estático, durante uma execução do algoritmo dinâmico, os valores de transferência, tempos de execução e disponibilidade de máquinas são calculados em tempo real. Embora essa abordagem reproduza mais fielmente um cenário real de escalonamento de tarefas dinâmico durante a execução de um *workflow*, a mesma gera um grande custo computacional e exige algoritmos de enorme complexidade quando comparados com algoritmos de escalonamento de *workflows* estáticos.

2.2 Revisão Bibliográfica

2.2.1 Heurísticas para WfCs

De acordo com um recente levantamento bibliográfico, feito em Teylo et al. [45], a maioria das técnicas de escalonamento de WfCs propostas na literatura são baseadas em heurísticas. Um dos algoritmos mais utilizados é o *Heterogeneous Earliest-Finish-Time* (HEFT) proposto em Topcuoglu et al. [46]. O HEFT é um algoritmo de aplicação de escalonamentos em um número finito de máquinas e é dividido em duas fases. Na primeira fase, chamada *task prioritising phase*, todas as tarefas são ordenadas de acordo com o custo médio de comunicação e com o custo médio computacional. Empates são resolvidos de forma aleatória. Dessa forma, uma ordem topológica é construída para as tarefas. Na segunda fase, chamada *processor selection phase*, as máquinas de execução são selecionadas para cada tarefa. O algoritmo HEFT procura o primeiro tempo disponível onde cada tarefa ordenada pode ser executada. Ao término da execução das duas fases do algoritmo, um escalonamento viável é criado.

Seguindo as investigações de Topcuoglu et al. [46], em Durillo et al. [18], uma nova heurística chamada *Multi-Objective* HEFT (MOHEFT) foi concebida. O novo algoritmo utiliza grande parte da versão HEFT como base, mas gera um conjunto de soluções. Sobre o conjunto de soluções, uma curva de pareto [42] é construída e a escolha dos movimentos é realizada com base nessa curva. O MOHEFT apresentou soluções viáveis com bons resultados quando comparado aos desempenhos de heurísitcas mais simples, incluindo o algoritmo base HEFT. Embora esse algoritmo leve em consideração alguns aspectos do ambiente de execução para o cálculo da função objetivo, ambos não se preocupam com a localização de armazenamento dos dados para o cálculo do custo da solução.

Uma abordagem gulosa foi apresentada no algoritmo MinMin de Blythe et al. [5]. O algoritmo lista um conjunto de tarefas que podem ser executadas e, dentre essas tarefas e todo o conjunto de máquinas, gera-se um conjunto de pares tarefa-máquina. O par que possui o menor custo entre todos é escolhido e inserido na solução. Essa etapa é repetida até que todas as tarefas tenham sido alocadas a alguma máquina do sistema. Essa heurística não leva em consideração relações entre tarefas para as escolhas de inclusão. Além disso, a capacidade de armazenamento das máquinas não é considerada como uma restrição do problema. Assim, soluções inviáveis com péssimo escalonamento podem ser geradas.

2.2.2 Heurísticas baseadas em Metaheurísticas para WfCs

Algumas heurísticas baseadas em metaheurísticas foram empregadas nas soluções de problemas de escalonamento de WfCs, tais como *Particle Swarm Optimization* (PSO), *Ant Colony Optimization* (ACO) e Algoritmos Genéticos (AG).

PSO é uma técnica de otimização adaptativa baseada em buscas globais introduzida por Kennedy et al. [25]. O algoritmo é similar a outros baseados em população, mas não há recombinação direta entre indivíduos. PSO é baseado no comportamento social das partículas (nome atribuído a soluções construídas). Em cada geração, cada partícula ajusta sua trajetória baseada na melhor solução local e na posição da melhor solução global de toda a população. Esse processo aumenta a natureza estocástica de uma partícula e converge rapidamente para um mínimo local com uma solução razoavelmente boa. Em Pandey et al. [33], uma heurística baseada em PSO foi apresentada. A heurística de escalonamento é dinâmica e tem como objetivo minimizar o custo financeiro de *workflows* executados em ambientes de nuvens públicas. A abordagem proposta foi capaz de balancear dinamicamente a carga de tarefas entre os recursos disponíveis e apresentou resultados três vezes melhor em relação às heurísticas baseadas em variações do algoritmo HEFT. Embora a versão proposta por Pandey et al. [33] considere o tempo de transferência de arquivos dentro do *workflow*, a mesma não considera a distribuição de dados no ambiente.

ACO é uma técnica que funciona simulando o depósito de feromônios em caminhos feito por colônias de formigas, sendo eficaz em vários problemas de otimização combinatória [9]. Em Zhang et al. [9], foi proposta uma heurística baseada em ACO para uma versão de escalonamento de WfCs que visa atender três objetivos: (i) confiabilidade de serviços; (ii) custo monetário da solução; e (iii) *makespan*. Dessa maneira, trata-se de uma heurística multi-objetivo que tenta encontrar a melhor solução balanceada entre esses objetivos simultaneamente. Os autores propuseram diversas maneiras de calcular o decaimento dos feromônios que são selecionadas de forma heurística no decorrer da execução do algoritmo. Os resultados reportados mostram uma melhora de 20% a 30% em relação às heurísticas baseadas em MinMin e HEFT.

Em Hu et al. [22], o algoritmo *knowledge-based ant colony optmization* (KBACO) foi apresentado. O KBACO utiliza ACO em conjunto com heurísticas de aprendizado que, durante a execução do algoritmo, auxiliam nas decisões de escalonamento. Nessa versão, o escalonamento deve respeitar um tempo máximo de *makespan* imposto pelo usuário. O algoritmo não considera taxas de transferência de dados na rede na decisão

de escalonamento.

Em Yu et al. [49], um AG para o problema de escalonamento estático de *workflows* em nuvens foi apresentado. Assim como no KBACO, o usuário pode selecionar o valor máximo para o *makespan* de um escalonamento, além de poder, também, escolher o custo monetário máximo de aluguel de máquinas. A representação da solução é realizada por meio de uma tupla (máquinas x taferas), que define a execução da tarefa na máquina presentes na tupla. A população inicial é gerada por meio de dois métodos: *round-robin*, onde as permutações de pares são inseridas na solução; e *Best-Fit*, onde a melhor tupla viável para inserção na solução é adicionada a cada iteração. Os movimentos de busca local, *crossover* e mutação lidam com um par de tuplas, fazendo trocas entre cada par.

2.2.3 Estratégias com Escalonamento de Tarefas e Alocação de Dados para WfCs

Nesta subseção, serão apresentados algoritmos e soluções para escalonamento de tarefas em WfCs que levam em conta tanto o escalonamento de tarefas quanto a alocação dos arquivos e o custo de transferência dos mesmos durante a execução de um *workflow* científico.

Em Szabo et al. [44], o impacto da transferência de arquivos durante a execução de escalonamento de WfCs foi discutido pela primeira vez. Os autores argumentaram que, por causa do grande aumento do número de arquivos e do volume de transferências necessárias para conclusão de um *workflow*, é fundamental considerar a relação entre dados e tarefas para otimizar um plano de escalonamento. Um algoritmo evolutivo que otimiza o escalonamento de tarefas em máquinas, mas, também, a redução das transferências foi apresentado. Nesse trabalho, o modelo de armazenamento funciona da seguinte forma: os arquivos oriundos da execução de uma tarefa são armazenados tanto no servidor geral quanto na máquina na qual foi executada a tarefa. Dessa forma, tarefas executadas na mesma máquina, que necessitam de arquivos em comum, têm certa vantagem, já que não precisam contar com o tempo de *download* do servidor geral.

Wang et al. [48] apresentaram uma solução para o problema da minimização das transferências de dados em *workflows* alocados em *data-centers* diversos. A localização inicial dos arquivos é definida por meio do algoritmo de clusterização k-Means [6] e, em seguida, uma técnica de replicação de tarefas é aplicada para reduzir a transferência dos dados produzidos entre *data-centers* distintos. Além da complexidade inerente à replicação, Wang et al. [48] não levaram em consideração a transferência de arquivos

entre máquinas de um mesmo *data-center*.

Bryk et al. [7] propuseram um modelo para execução de múltiplos *workflows* em ambientes de nuvem. O algoritmo proposto *File Locality-Aware Scheduling* (FLA-S) executa uma alocação dinâmica de tarefas onde, a cada etapa, tarefas prontas para serem executadas são alocadas a uma máquina. De acordo com a localização dos dados, algumas tarefas são priorizadas, minimizando as taxas de transferência. O modelo, assim como apresentado por Szabo et al. [44], considera que os dados são armezenados em um nó central, assim como nas máquinas onde foram produzidos. Isso faz com que tarefas que apresentam relação de dependência devido à criação e ao consumo de arquivos sejam, preferencialmente, alocadas na mesma máquina, a fim de evitar transferências.

2.3 Problema de Escalonamento de Tarefas e Alocação de Arquivos em *Workflows* Científicos

Teylo et al. [45] propuseram a primeira abordagem para o Problema de Escalonamente de Tarefas e Alocação de Arquivos em *Workflows* Científicos em máquinas com CPU (PEAW-CPU) que levava em conta, também, a distribuição dos dados gerados durante a execução do *workflow*. Em [45], foram apresentados o modelo matemático em conjunto com uma heurística híbrida baseada em AG. Os autores constataram que, devido à complexidade gerada pelas restrições e o grande número de variáveis do modelo matemático, o uso de heurísticas se mostrou mais eficiente.

2.3.1 Formulação Matemática

Em Teylo et al. [45], o primeiro modelo de programação inteira mista para o PEAW-CPU foi apresentado e, também, uma solução heurística baseada em AG.

Seja $F = S \cup D$ o conjunto de todos os arquivos onde cada arquivo $d \in F$ tem tamanho $W(d)$. Cada um dos arquivos pode ser tanto estático (S), com máquina de origem $O(d) \in M$, onde M é o conjunto de máquinas, quanto dinâmico (D), gerado durante a execução do *workflow*. Para cada $i \in N$, onde N é o conjunto de tarefas, um conjunto de arquivos de entrada $\Delta_{in}(i) \subseteq F$ necessários para a execução da tarefa i e um conjunto de arquivos de saída $\Delta_{out}(i) \subseteq D$ gerados pela execução da tarefa i são considerados. Além disso, define-se T_M como o tempo máximo de execução do *workflow* e $T = \{1, ..., T_M\}$ como o conjunto discreto de tempos onde uma ação de execução de

Tabela 2.1: Lista de parâmetros e suas descrições.

Parâmetros	***Descrição***
S	Conjunto de arquivos estáticos.
D	Conjunto de arquivos dinâmicos.
$F = S \cup D$	Conjunto de arquivos.
$O(d)$	Máquina de origem do arquivo estático $d \in S$.
$W(d)$	Tamanho do arquivo $d \in F$.
N	Conjunto de tarefas.
a_i	Quantidade de trabalho da tarefa $i \in N$.
M	Conjunto de máquinas.
t_{ij}	Tempo de processamento da tarefa $i \in N$, $j \in M$.
$\overrightarrow{t}_{djp}$	Tempo despendido pela máquina $j \in M$ para ler o arquivo $d \in F$ armazenado na máquina $p \in M$
$\overleftarrow{t}_{djp}$	Tempo despendido pela máquina $j \in M$ para escrever o arquivo $d \in D$ armazenado na máquina $p \in M$.
$\Delta_{in}(i) \subseteq F$	Conjunto de arquivos necessários para executar a tarefa $i \in N$.
$\Delta_{out}(i) \subseteq D$	Conjunto de arquivos gerados pela tarefa $i \in N$.
cm_j	Capacidade de armazenamento da máquina j.

tarefas, escrita ou leitura de arquivos pode ser executada.

O PEAW-CPU é definido como um escalonamento de tarefas e alocação de arquivos em máquinas, levando em consideração as capacidades e buscando a minimização do *makespan*, que é o maior tempo de finalização de trabalho entre todas as máquinas. Apresenta-se, na Tabela 2.1, uma descrição dos parâmetros e, na Tabela 2.2, uma descrição das variáveis usadas no modelo matemático proposto em Teylo et al. [45].

A função objetivo (2.1) minimiza o *makespan* da aplicação e está sujeita às restrições definidas a seguir:

$$\min z_T \tag{2.1}$$

As restrições (2.2) garantem que toda tarefa deve ser executada. As restrições (2.3) e (2.4) asseguram que toda operação de leitura e escrita deve ser finalizada, respectivamente.

Tabela 2.2: Lista de variáveis e suas descrições.

Variáveis	***Descrição***
x_{ijt}	Variável binária que indica se a tarefa $i \in N$ começa a execução na máquina $j \in M$ no tempo $t \in T$, ou não.
a_{idjpt}	Variável binária que indica se a tarefa $i \in N$, executada na máquina $j \in M$, começa a ler o arquivo $d \in \Delta_{in}(i)$ armazenado na máquna $p \in M$ no tempo $t \in T$, ou não.
b_{djpt}	Variável binária que indica se o arquivo $d \in D$ começa a ser transferido da máquina $j \in M$ para a máquina $p \in M$ no tempo $t \in T$, ou não.
y_{djt}	Variável binária que indica se o arquivo $d \in D$ está armazenado na máquina $j \in M$ no tempo $t \in T$, ou não.
z_T	Variável contínua que indica o tempo total de execução do *workflow* (*makespan*).

$$\sum_{j \in M} \sum_{t \in T} x_{ijt} = 1, \qquad \forall i \in N \tag{2.2}$$

$$\sum_{j,p \in M} \sum_{t \in T} a_{idjpt} = 1, \qquad \forall i \in N, \forall d \in \Delta_{in}(i) \tag{2.3}$$

$$\sum_{j,p \in M} \sum_{t \in T} b_{djpt} = 1, \qquad \forall d \in D \tag{2.4}$$

Inequações (2.5) garantem que os arquivos $d \in \Delta_{out}(i)$ só podem ser escritos se a tarefa i foi executada no tempo correto. Além disso, as restrições (2.6) asseguram que o arquivo d não pode ser escrito antes do tempo de processamento da tarefa i (responsável pela sua escrita). Note que ambos os conjuntos de restrições (2.5) e (2.6) funcionam em conjunto para garantir um tempo viável para o processo de escrita.

$$b_{djpt} \leq x_{ij(t-t_{ij})}, \qquad \forall d \in D, \forall j, p \in M, \forall t = (t_{ij} + 1) \cdots T_M \text{ onde } d \in \Delta_{out}(i) \tag{2.5}$$

$$b_{djpt} = 0, \qquad \forall d \in D, \forall j, p \in M, \forall t \in [1, t_{ij}] \text{ onde } d \in \Delta_{out}(i) \tag{2.6}$$

A restrição (2.7) afirma que uma tarefa só pode ser executada se todas as leituras necessárias foram concluídas em tempo viável.

$$x_{ijt} \leq \sum_{p \in M} a_{idjp(t-\overrightarrow{t}_{djp})}, \quad \forall i \in N, \forall d \in \Delta_{in}(i), \forall j \in M, \forall t \in T, \text{ onde } (t - \overrightarrow{t}_{djp}) \geq 1 \quad (2.7)$$

Inequações (2.8) ratificam que somente uma ação (execução, leitura ou escrita) pode ser realizada em cada tempo em cada máquina.

$$\sum_{i \in N} \sum_{q=\max(1,t-t_{ij}+1)}^{t} x_{ijq} + \sum_{d \in D} \sum_{p \in M} \sum_{r=\max(1,t-\overleftarrow{t}_{djp}+1)}^{t} b_{djpr} +$$
$$\sum_{i \in N} \sum_{d \in \Delta_{in}(i)} \sum_{p \in M} \sum_{r=\max(1,t-\overrightarrow{t}_{djp}+1)}^{t} a_{idjpr} \leq 1, \qquad \forall j \in M, \forall t \in T \qquad (2.8)$$

Restrições (2.9) afirmam que não existem arquivos dinâmicos pré-armazenados. Restrições (2.10) sustentam que todo arquivo do tipo dinâmico está alocado nas suas máquinas de origem.

$$y_{dj1} = 0, \qquad \forall d \in D, \forall j \in M \qquad (2.9)$$
$$y_{djt} = 1, \qquad \forall d \in S \mid j \in O(d), \forall t \in T \qquad (2.10)$$

Restrições (2.11) e (2.12) conectam a variável de armazenamento y com a variável de escrita $\overleftarrow{x}$ e as variáveis de leitura $\overrightarrow{x}$, garantindo um processo viável de leitura e escrita, respectivamente.

$$y_{dp(t+1)} \leq y_{dpt} + \sum_{j \in M} b_{djp(t-\overleftarrow{t}_{djp})}, \quad \forall d \in F, \forall p \in M, \forall t \in T, \text{ onde } (t - \overrightarrow{t}_{djp}) \geq 1 \quad (2.11)$$
$$\sum_{j \in M} a_{idjpt} \leq y_{dpt}, \qquad \forall i \in N, \forall d \in \Delta_{in}(i), \forall p \in M, \forall t \in T \quad (2.12)$$

As restrições (2.11) garantem que os arquivos só serão armazenados em alguma máquina após terem sido produzidos, e as restrições (2.12) asseguram que os arquivos só serão lidos se foram previamente armazenados em alguma máquina.

A capacidade de armazenamento das máquinas está restrito em (2.13). Restrições (2.14) correlacionam a última operação de escrita com o tempo total de execução, isto é,

o *makespan*, da aplicação. Note que uma tarefa sempre cria arquivos.

$$\sum_{d \in D} y_{djt} W(d) \leq cm_j, \qquad \forall j \in M, \forall t \in T \tag{2.13}$$

$$b_{djpt} \cdot (t + \overleftarrow{t}_{djp}) \leq z_T, \qquad \forall d \in D, \forall j, p \in M, \forall t \in T \tag{2.14}$$

Além disso, a restrição operacional (2.15) precisa ser satisfeita: uma tarefa i só pode começar qualquer atividade de leitura se todos os arquivos $d \in \Delta_{in}(i)$ ja estão disponíveis. Finalmente, as restrições restantes são de intregralidade e não-negatividade.

$$a_{idjpt} \cdot |\Delta_{in}(i) \cap D| \leq \sum_{g \in \{\Delta_{in}(i) \cap D\}} \sum_{l,o \in M} \sum_{u=1}^{t - \overleftarrow{t}_{glo}} b_{glou}, \quad \forall i \in N, \forall d \in \Delta_{in}(i), \forall j, p \in M, \forall t \in T \tag{2.15}$$

$$x_{ijt}, a_{idjpt}, b_{djpt}, y_{djt} \in \{0, 1\}, \qquad \forall j, p \in M, \forall t \in T, \forall d \in D \tag{2.16}$$

$$z_T \in \mathbb{R}^+ \tag{2.17}$$

2.3.2 Heurística *Hybrid Evolutive Algorithm*

A heurística para o escalonamento de WfCs com armazenamento de dados apresentada em Teylo et al. [45] usa um AG com *path-relinking* [21], buscando minimizar o custo de transferência de arquivos e o *makespan* do escalonamento, levando-se em conta a capacidade de armazenamento de cada máquina. No PEAW-CPU, existem arquivos do tipo de estático que, como dado de entrada para esse problema, estão pré-armazenados em algumas máquinas. Esses arquivos estáticos são necessários para iniciar tarefas que não possuem dependência em relação a outras tarefas. Os dados criados durante a execução do *workflow* são chamados de dinâmicos e, diferente do proposto em Szabo et al. [44], podem ser transferidos e armazenados em qualquer máquina do ambiente.

A heurística *Hybrid Evolutive Algorithm*, descrita no Algoritmo 1, constrói uma população inicial (linha 1) por meio de duas abordagens distintas: 80% das soluções são geradas por heurísticas e 20% são geradas aleatoriamente. As heurísticas MinMin e HEFT foram utilizadas para gerar a maior parcela da população inicial, sendo 40% delas oriundas do MinMin e 40% do HEFT. Como essas abordagens são determinísticas, um processo de

mutação altera de 5% até 90% das alocações das tarefas às máquinas e essa mutação não gera soluções inviáveis. O conjunto elite de soluções é inicializado como vazio (linha 3) e será preenchido, ao longo da execução do algoritmo, com as β melhores soluções.

Após a criação da população inicial, a melhor solução do conjunto é armazenada na variável *melhor_global* (linha 2) e o algoritmo passa por um processo iterativo (linhas 6 a 26), com *iter* iterações definidos como parâmetro de entrada do algoritmo. Em cada uma dessas iterações, existe a probabilidade de 50% (linha 7) de o procedimento de busca local ser ativado sobre a população. Caso o procedimento de busca local seja executado, um conjunto de três vizinhanças de busca é executado uma única vez e sequencialmente sobre as 15% melhores soluções da população corrente (linha 9). As vizinhanças de busca local são as seguintes: (i) troca-mv, que executa a troca da máquina de execução entre duas tarefas; (ii) troca-posição, que troca a posição de duas tarefas na ordem de execução (caso a nova ordem respeite a topologia de dependência); e (iii) move-elemento, que altera a execução de uma tarefa ou a alocação de um arquivo para outra máquina. Todas as buscas são de *first improvement*, isto é, executam até que um movimento melhore a solução ou até que todas as combinações sejam testadas.

Seguindo o fluxo da iteração, se, após a aplicação da busca local, o *melhor_atual* tenha um custo de solução menor do que o custo de *melhor_global*, a solução *melhor_global* é atualizada para a melhor solução corrente (linhas 12 e 13). Caso o conjunto elite não esteja vazio (linha 14), uma heurística *Backward Path Relinking* [21] é executada sobre o conjunto elite e a melhor solução global *melhor_global* (linha 15). A heurística *Path Relinking* modifica cada solução do conjunto elite, alterando a máquina onde cada tarefa foi executada, de acordo com a melhor solução global. A melhor dentre todas essas trocas é retornada pela versão de *Path Relinking* implementada.

A distância entre duas soluções é calculada pelo número de componentes que estão com valores diferentes entre elas (o número de tarefas iguais executadas em máquinas diferentes, a ordem de execução das tarefas distintas ou alocação de arquivos em máquinas diferentes). Caso a solução *melhor_global* esteja a uma distância maior que α de todo o conjunto elite (linha 17), então essa solução é considerada diferente o suficiente para fazer parte do conjunto elite e é adicionada a esse conjunto. Após a inserção da solução *melhor_global* nesse conjunto, seu tamanho é incrementado e, caso o tamanho seja maior que β previamente estabelecido, a pior solução é removida do conjunto elite (linha 20).

Para finalizar a iteração, o processo de *crossover* e de mutação é aplicado sobre a população no método *geraPopulacao*() (linha 24). Nesse procedimento, uma nova solução

é construída, a partir da mistura de um par de soluções presentes na população, escolhidas por meio de um processo *round-robin*. Após a nova solução ser criada pelo *crossover* das duas soluções iniciais selecionadas, a mesma passa por um processo de mutação, onde a alocação dos arquivos pode ser alterada de forma aleatória. Com o fim das etapas de *crossover* e mutação, uma nova população $P*$ é criada com a união da população inicial com as soluções resultantes dos movimentos. As 5% melhores soluções presentes na população $P*$ são adicionadas à população da próxima iteração. O restante da população é preenchida aleatoriamente com componentes da população $P*$ até o tamanho máximo ser atingido.

A próxima iteração do algoritmo usa a população gerada pelo procedimento *geraPopulação* (linha 24).

Algoritmo 1 HEA (max_iter, α, β)

1: $P \leftarrow populacaoInicial()$
2: $melhor_global \leftarrow encontraMelhor(P)$
3: $ConjElite \leftarrow \{\}$
4: $iter \leftarrow 0$
5: $busca_local \leftarrow 0$
6: **Enquanto** $iter < max_iter$ **faça**
7: $\quad busca_local \leftarrow aleatorioZeroUm()$
8: $\quad$ **Se** $busca_local = 1$ **então**
9: $\quad\quad P \leftarrow buscasLocais()$
10: $\quad$ **Fim-se**
11: $\quad melhor_atual \leftarrow encontraMelhor(P)$
12: $\quad$ **Se** $melhor_atual < melhor_global$ **então**
13: $\quad\quad melhor_global \leftarrow melhor_atual$
14: $\quad\quad$ **Se** $ConjElite \neq \{\}$ **então**
15: $\quad\quad\quad melhor_global \leftarrow pathRelinking(melhor_global, ConjElite)$
16: $\quad\quad$ **Fim-se**
17: $\quad\quad$ **Se** $\forall solucao \in ConjElite, distancia(melhor_global, solucao) \geq \alpha$ **então**
18: $\quad\quad\quad ConjElite \leftarrow ConjElite \cup melhor_global$
19: $\quad\quad\quad$ **Se** $|conjElite| \geq \beta$ **então**
20: $\quad\quad\quad\quad removeCromossomo(ConjElite)$
21: $\quad\quad\quad$ **Fim-se**
22: $\quad\quad$ **Fim-se**
23: $\quad$ **Fim-se**
24: $\quad P \leftarrow geraPopulacao(P)$
25: $\quad iter \leftarrow iter + 1$
26: **Fim-enquanto**
27: **Retorne** $melhor_global$

No próximo capítulo, será descrita a heurística proposta neste trabalho para resolver o PEAW-CPU, bem como as metaheurísticas que serviram de base.

Capítulo 3

Heurística Proposta para o PEAW-CPU

3.1 Metaheurísticas

Nesta seção, serão apresentadas as metaheurísticas genéricas utilizadas como base para a criação da heurística proposta neste trabalho.

3.1.1 GRASP

Greedy Randomized Adaptative Search Procedures [20] (GRASP), descrito no Algoritmo 2, é uma metaheurística simples e iterativa que foi aplicada com sucesso em diversas classes de problemas de otimização [12, 38, 39, 40, 41]. Cada iteração GRASP é dividida em duas fases. Primeiramente, uma solução viável é construída na fase de construção. Posteriormente, na segunda fase, a vizinhança da solução é explorada por um procedimento de busca local que procura por uma solução melhor. A melhor solução de todas as iterações é retornada como resultado.

A fase de construção do GRASP é uma heurística que busca balancear o caráter aleatório com o determinístico das escolhas durante a construção de uma solução, na tentativa de alcançar uma solução melhor [21]. Em cada iteração do procedimento de construção (linhas 6 a 12), é considerado um conjunto de candidatos que podem ser adicionados à solução parcial em construção. Após a avaliação de todos os candidatos, segundo uma função gulosa (que geralmente calcula o custo da solução após a inserção do candidato), a lista de candidatos (LC) é criada e ordenada (linha 7). Então, a lista restrita de candidatos (LRC) é definida, composta dos melhores candidatos, usando um parâmetro $\alpha \in [0, 1]$ para truncar a LC (linhas 8 e 9). Finalmente, um elemento da LRC

é selecionado aleatoriamente e inserido na solução (linhas 10 e 11).

Com o fim da fase de construção, é iniciada a fase de busca local (linha 13). O processo de busca local é realizado por meio de vizinhanças de busca isoladas ou com emprego de heurísticas baseadas em metaheurísticas, tais como *Variable Neighbourhood Descend* (VND), *Variable Neighbourhood Search* (VNS), *Iterated Local Search* (ILS), *Simulated Annealing* (SA), entre outras [21]. Após o fim das iterações, a melhor solução obtida pelo GRASP é retornada na linha 20.

A metaheurística GRASP, bem como as duas fases dessa metaheurística, para problemas de minimização, é apresentada no Algoritmo 2.

Algoritmo 2 GRASP (max_iter, α)

1: $S^* \leftarrow \{\}$
2: $f^* \leftarrow \infty$
3: $iter \leftarrow 0$
4: **Enquanto** $iter < max_iter$ **faça**
5: $S \leftarrow \{\}$
6: **Enquanto** S não está completo **faça**
7: $LC \leftarrow$ candidatosViáveis()
8: $corte \leftarrow |LC| * \alpha$
9: $LRC \leftarrow constroiLRC(LC, corte)$
10: $candidato \leftarrow selecionaAleatoriamente(LRC)$
11: $S \leftarrow$ insereCandidato($S, candidato$)
12: **Fim-enquanto**
13: $S \leftarrow$ BuscaLocal(S)
14: **Se** $f(S) < f^*$ **então**
15: $S^* \leftarrow S$
16: $f^* \leftarrow f(S)$
17: **Fim-se**
18: $iter \leftarrow iter + 1$
19: **Fim-enquanto**
20: **Retorne** S^*

3.1.2 Iterated Local Search

Iterated Local Search (ILS), apresentado no Algoritmo 3, é um método de buscas e perturbação para obtenção de soluções de boa qualidade para problemas de otimização com-

binatória proposto por Lourenço et al. [21]. Essa metaheurística consiste na construção de uma solução inicial (linha 3) e um posterior processo iterativo (linhas 5 a 13) no qual se aplica, em cada iteração, uma perturbação na solução (linha 6) e um aprimoramento na solução perturbada por meio de uma busca local (linha 7). A escolha dos movimentos de pertubação e busca local é uma questão importante na fase de planejamento de uma heurística baseada em ILS. A perturbação não deve ser grande o suficiente para descaracterizar uma solução e fugir da região de busca e nem perturbar partes da solução não afetadas pelas vizinhanças de busca [21]. A solução inicial pode ser adquirida por meio de um método construtivo de soluções ou criada de forma aleatória. A estratégia de busca local pode ser realizada por meio de uma heurística ou de uma ordem de vizinhanças de busca previamente estipulada. Ao fim das iterações, a melhor solução encontrada é retornada pelo algoritmo (linha 14).

Algoritmo 3 ILS (max_iter)

1: $S^* \leftarrow \{\}$
2: $f^* \leftarrow \infty$
3: $S \leftarrow$ SoluçãoInicial()
4: $iter \leftarrow 0$
5: **Enquanto** $iter < max_iter$ **faça**
6: $\quad S \leftarrow$ Perturbação(S)
7: $\quad S \leftarrow$ BuscaLocal(S)
8: $\quad$ **Se** $f(S) < f^*$ **então**
9: $\quad\quad S^* \leftarrow S$
10: $\quad\quad f^* \leftarrow f(S)$
11: $\quad$ **Fim-se**
12: $\quad iter \leftarrow iter + 1$
13: **Fim-enquanto**
14: **Retorne** S^*

3.1.3 GRASP-ILS

A heurística denominada GRASP-ILS é a combinação de duas heurísticas baseadas nas metaheurísticas supracitadas. Foi usada com êxito em Ribeiro et al. [37] e faz uso da metaheurística ILS como procedimento de busca local do GRASP, que funciona como processo de construção de soluções. O pseudo-código do GRASP-ILS está apresentado no Algoritmo 4. As iterações do GRASP são realizadas entre as linhas 4 e 28. Em cada

iteração, a construção da solução é realizada (linhas 6 a 12) e, em seguida, é iniciado o ILS (linhas 18 a 26). Dentro da abordagem ILS, os movimentos de perturbação são aplicados (linha 19), seguidos pela aplicação das vizinhanças de busca local (linha 20).

Algoritmo 4 GRASP-ILS ($max_iterGRASP$, $max_iterILS$, α)

1: $S^* \leftarrow \{\}$
2: $f^* \leftarrow \infty$
3: $iterGRASP \leftarrow 0$
4: **Enquanto** $iterGRASP < max_iterGRASP$ **faça**
5: $S \leftarrow \{\}$
6: **Enquanto** S não está completo **faça**
7: $LC \leftarrow$ candidatosViáveis()
8: $corte \leftarrow |LC| * \alpha$
9: $LRC \leftarrow constroiLRC(LC, corte)$
10: $candidato \leftarrow selecionaAleatoriamente(LRC)$
11: $S \leftarrow$ insereCandidato($S, candidato$)
12: **Fim-enquanto**
13: **Se** $f(S) < f^*$ **então**
14: $S^* \leftarrow S$
15: $f^* \leftarrow f(S)$
16: **Fim-se**
17: $iterILS \leftarrow 0$
18: **Enquanto** $iterILS < max_iterILS$ **faça**
19: $S \leftarrow$ Perturbação(S)
20: $S \leftarrow$ BuscaLocal(S)
21: **Se** $f(S) < f^*$ **então**
22: $S^* \leftarrow S$
23: $f^* \leftarrow f(S)$
24: **Fim-se**
25: $iterILS \leftarrow iterILS + 1$
26: **Fim-enquanto**
27: $iterGRASP \leftarrow iterGRASP + 1$
28: **Fim-enquanto**
29: **Retorne** S^*

3.2 Heurística Proposta para o PEAW-CPU

A estratégia proposta neste trabalho utiliza uma heurística híbrida baseada em GRASP-ILS e usa quatro vizinhanças de busca local, além de duas vizinhanças de perturbação. As componentes e suas estruturas de dados serão apresentadas nas subseções a seguir.

Três componentes podem caracterizar uma solução para o PEAW-CPU. Essas componentes são *Job*, *File* e *Machine*. Em *Job*, são representadas as tarefas do *workflow*, com os dados de tempo de execução, requisitos de arquivo e dados da topologia de dependências da instância. Em *Machine*, são representadas as máquinas existentes na instância, contando com *slowdown* (valor fracionário utilizado para modificar o desempenho entre as máquinas: quanto menor, mais rápida é a máquina), lista de arquivos alocados e tempos de transferência entre as máquinas. *File* representa um arquivo gerado ou construído pelo *workflow* e que pode ser estático (quando iniciam a execução do *workflow* pré-alocados em alguma *Machine*) ou dinâmico (quando são criados no decorrer do *workflow*).

Para definir uma solução para o PEAW-CPU, é necessário saber a ordem das tarefas no escalonamento, as máquinas onde as tarefas serão executadas e a alocação dos arquivos gerados pelas tarefas. A representação visual de uma solução está esquematizada na Figura 3.1, onde *Arquivos* mostra a identificação da máquina na qual cada arquivo está armazenado; *Tarefas* mostra a identificação de qual máquina é responsável pela execução das tarefas; e *Ordem de Execução*, que mostra a ordem de execução das tarefas no escalonamento.

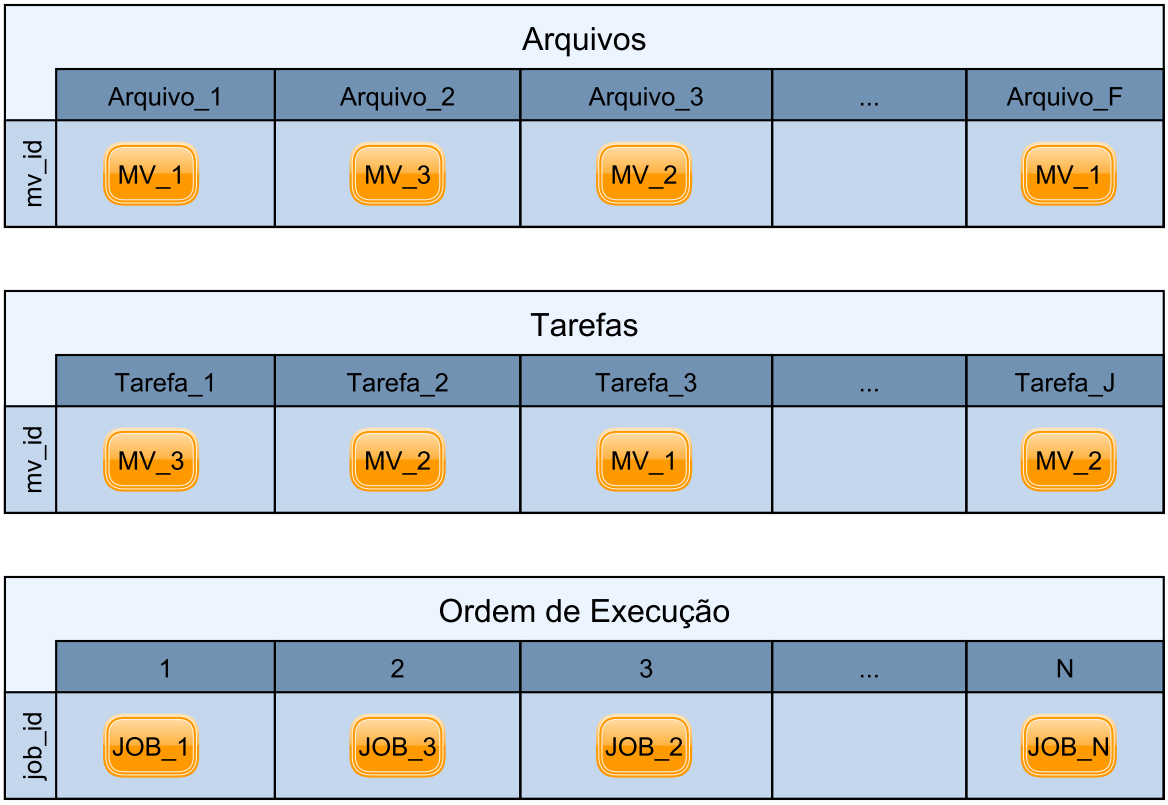

Figura 3.1: Representação de uma solução para o PEAW-CPU.

3.2.1 Construtivo com Componente Aleatória

O pseudo-código do construtivo está apresentado no Algoritmo 5. A componente construtiva da heurística proposta funciona da seguinte maneira: A LC é preenchida com todas as possíveis triplas (*Job*, *Machine* 1, *Machine* 2) (linhas 4 a 10), onde a primeira posição representa uma possível tarefa a ser inserida na solução, a segunda posição indica a máquina onde a tarefa será executada e a terceira posição a máquina onde os arquivos serão armazenados. O custo de cada inserção é calculado e os candidatos são ordenados de acordo com os respectivos custos (linha 11). Após o preenchimento de todos os candidatos possíveis, é montada a LRC com os $\alpha * |LC|$ candidatos com menor custo de inserção (linha 13). É escolhido, aleatoriamente, um candidato que será inserido na solução de acordo com os dados da tripla (*Job*, *Machine* 1, *Machine* 2) (linhas 14 e 15).

Algoritmo 5 Construtivo (α)

1: $S \leftarrow \{\}$
2: **Enquanto** S não está completo **faça**
3: $LC \leftarrow \{\}$
4: **Para** cada *Job* j não inserido e viável **faça**
5: **Para** cada *Machine* m de execução **faça**
6: **Para** cada *Machine* w de escrita **faça**
7: $LC \leftarrow$criaTripla(j, m, w)
8: **Fim-para**
9: **Fim-para**
10: **Fim-para**
11: $LC \leftarrow$ OrdenarPorCusto(LC)
12: $corte \leftarrow |LC| * \alpha$
13: $LRC \leftarrow$constroiLRC$(LC, corte)$
14: $candidato \leftarrow$selecionaAleatoriamente(LRC)
15: $S \leftarrow$ insereCandidato$(candidato)$
16: **Fim-enquanto**
17: **Retorne** S

3.2.2 Busca Local

A heurística GRASP-ILS proposta conta com quatro vizinhanças exaustivas de busca local, cada uma com o foco em uma área específica da solução. O uso das vizinhanças dentro de uma heurística de busca VND [21] foi possível, uma vez que o movimento realizado por uma busca não interfere ou destrói movimentos realizados anteriormente, sendo possível uma melhoria constante da solução. Os pseudocódigos da heurística de busca estão apresentados no Algoritmo 6 a seguir.

No Algoritmo 6, que representa o método de busca, linhas 1 e 2 definem valores iniciais para *melhoria* e f^*. Entre as linhas 3 e 33 está o comando de repetição principal da heurística VND, que continua enquanto há melhorias. Nas linhas 5, 12, 19 e 26, são aplicadas as vizinhanças de busca local apresentadas nas subseções seguintes. A melhoria dos movimentos é avaliada nos blocos de linhas de 6 a 11, de 13 a 18, de 20 a 25 e de 27 a 32. Se nenhuma solução de custo menor puder ser alcançada pelas buscas, o algoritmo VND retorna, na linha 34, a melhor encontrada.

Algoritmo 6 VND (S^*)

1: $melhoria \leftarrow$ True
2: $f^* \leftarrow f(S^*)$
3: **Enquanto** $melhoria$ **faça**
4: $melhoria \leftarrow$ False
5: $S \leftarrow FixFileAllocation(S^*)$
6: **Se** $f(S) < f^*$ **então**
7: $melhoria \leftarrow$ True
8: $S^* \leftarrow S$
9: $f^* \leftarrow f(S)$
10: Volte para passo 3:
11: **Fim-se**
12: $S \leftarrow RelocateJob(S^*)$
13: **Se** $f(S) < f^*$ **então**
14: $melhoria \leftarrow$ True
15: $S^* \leftarrow S$
16: $f^* \leftarrow f(S)$
17: Volte para passo 3:
18: **Fim-se**
19: $S \leftarrow SwapMachine(S^*)$
20: **Se** $f(S) < f^*$ **então**
21: $melhoria \leftarrow$ True
22: $S^* \leftarrow S$
23: $f^* \leftarrow f(S)$
24: Volte para passo 3:
25: **Fim-se**
26: $S \leftarrow SwapMachinePair(S^*)$
27: **Se** $f(S) < f^*$ **então**
28: $melhoria \leftarrow$ True
29: $S^* \leftarrow S$
30: $f^* \leftarrow f(S)$
31: Volte para passo 3:
32: **Fim-se**
33: **Fim-enquanto**
34: **Retorne** S^*

3.2.2.1 *Fix File Allocation*

A vizinhança *Fix File Allocation*, proposta neste trabalho, lida com a alocação de arquivos gerados pelas tarefas. Para cada arquivo, é simulado o custo de sua alocação ser alterada para qualquer uma das demais máquinas. O movimento funciona como *first improvement* e serve para arrumar a solução após os movimentos mais intensos, realizados posteriormente. O custo da solução é simulado para cada movimento e apenas o primeiro melhor, se existente, é realizado. A Figura 3.2 ilustra a aplicação do movimento *Fix File Allocation*, onde a troca da máquina onde o Arquivo_2 está alocado trouxe benefício para o custo da solução.

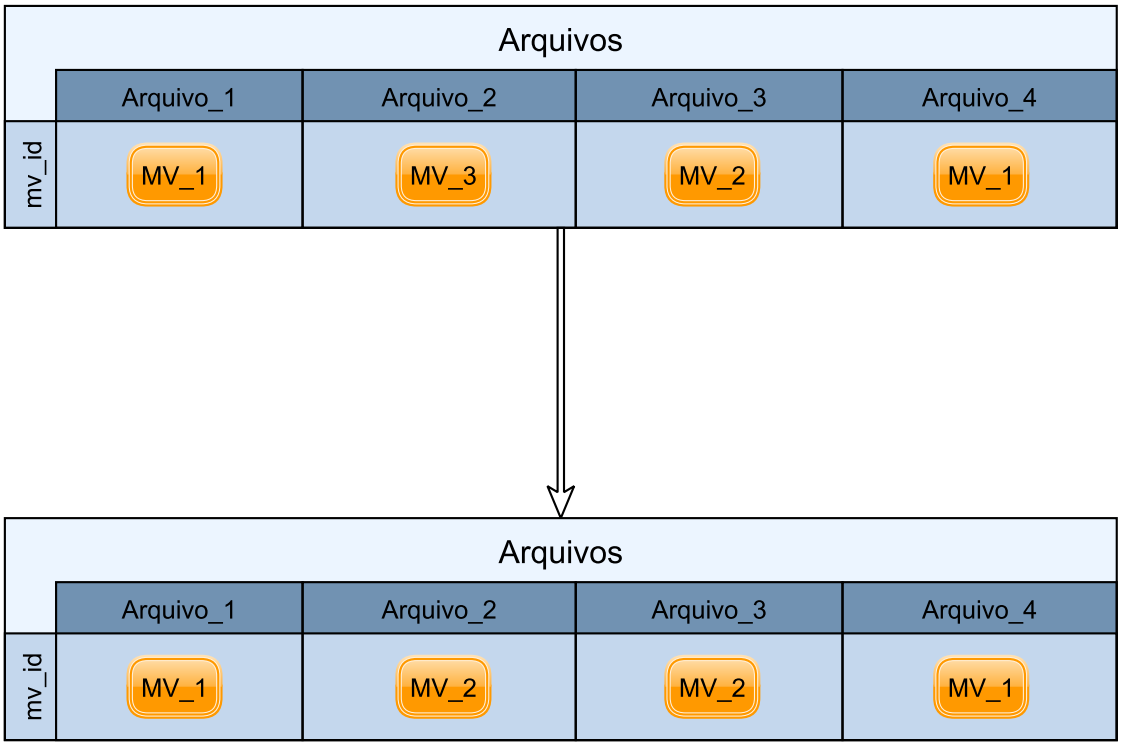

Figura 3.2: Representação da aplicação do movimento *Fix File Allocation*.

3.2.2.2 *Relocate Job*

A vizinhança *Relocate Job*, proposta neste trabalho, altera diretamente a ordem das tuplas (*Job*, *Machine* 1, *Machine* 2) e a topologia de execução do *workflow*. É testado, para cada item do vetor, se existe alguma outra posição viável na qual o custo da solução é melhorado. O *RelocateJob* segue a estratégia *first improvement*, onde o primeiro movimento de melhoria já é aplicado, assim que for encontrado. O movimento tem sua aplicação

exemplificada na Figura 3.3, onde a troca da ordem de execução do Job_1 e Job_2 trouxe melhora para o custo da solução.

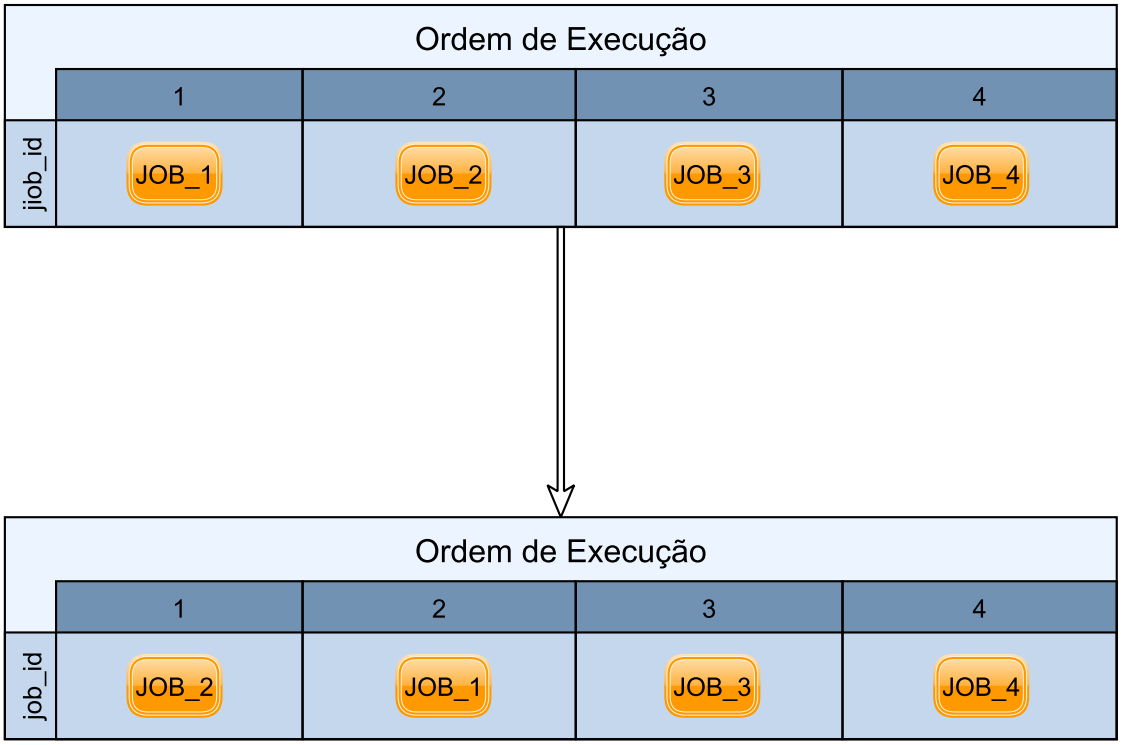

Figura 3.3: Representação da aplicação do movimento *Relocate Job*.

3.2.2.3 *Swap Machine*

A vizinhança *Swap Machine* é baseada no trabalho de Teylo et al. [45] e altera a máquina onde uma tarefa é executada. A troca da execução de cada tarefa para cada máquina é simulada antes da realização do movimento. Este movimento não causa inviabilidades na ordem de execução das tarefas, apenas altera o *makespan* da solução. Na Figura 3.4, está ilustrado o movimento *Swap Machine* aplicado sobre as máquinas nas quais as tarefas são executadas. O movimento em questão realiza a troca da máquina onde a Tarefa_2 é executada.

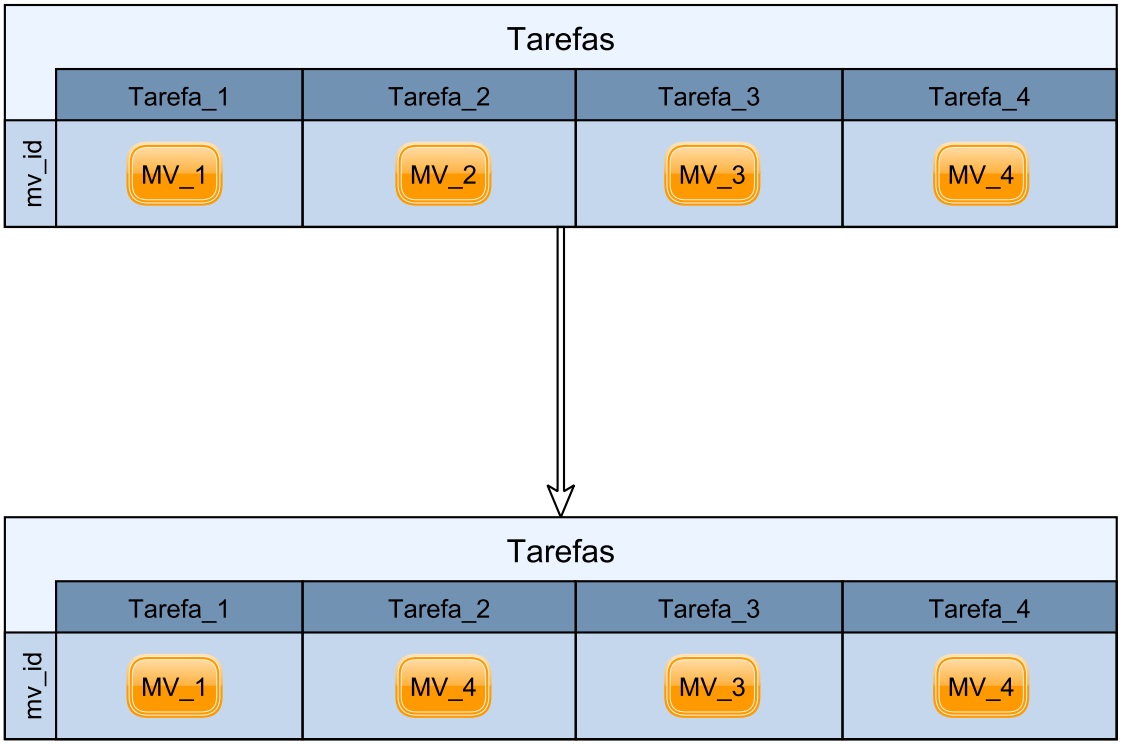

Figura 3.4: Representação da aplicação do movimento *Swap Machine*.

3.2.2.4 *Swap Machine Pair*

A vizinhança *Swap Machine Pair* também é baseada no trabalho de Teylo et al. [45] e trabalha com pares de tarefas, trocando a máquina de execução entre as tarefas selecionadas. Para cada par de tarefas, a troca de máquinas é simulada e o custo pŕe-calculado. Assim como as buscas anteriores, o primeiro movimento que causa melhoria no custo de solução é aplicado. O movimento tem uma complexidade de simulação maior do que o *Swap Machine* e, por isso, foi colocado numa camada mais interna do VND, onde as execuções são mais esporádicas. O movimento está representado na Figura 3.5, onde é feita a troca das máquinas de execução da Tarefa_2 e da Tarefa_4.

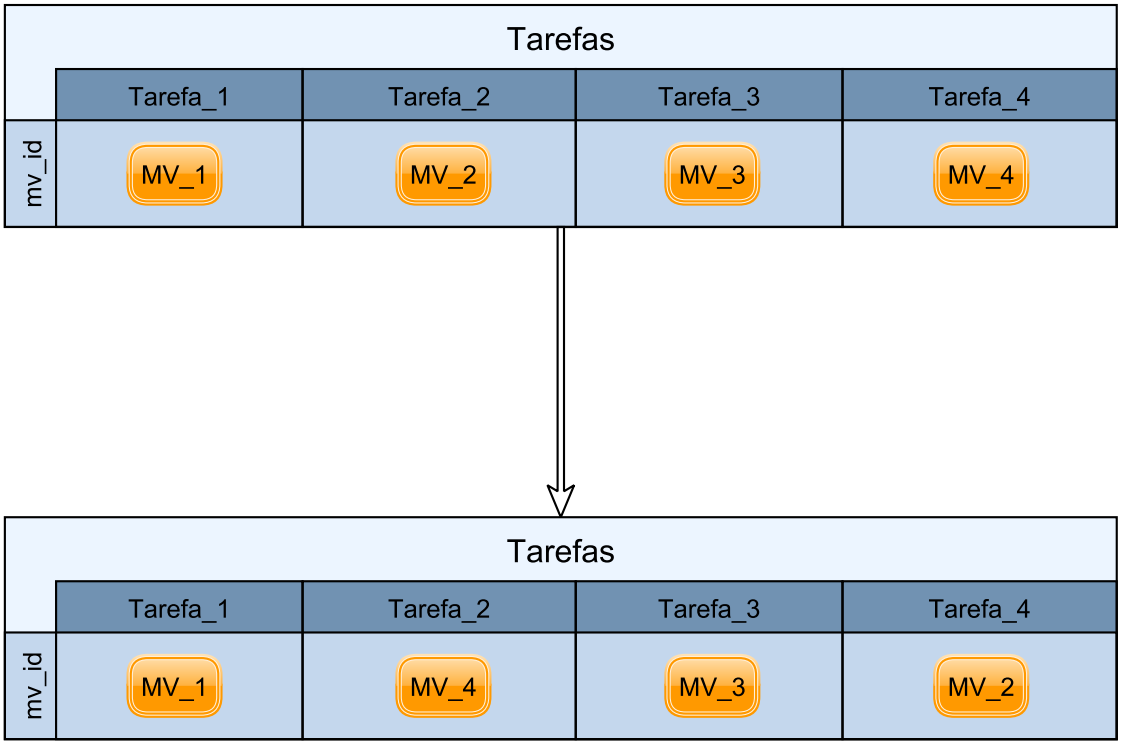

Figura 3.5: Representação da aplicação do movimento *Swap Machine Pair*.

3.2.3 Perturbação

Os movimentos de perturbação funcionam sobre partes da solução: a alocação dos arquivos e a máquina de execução das tarefas. Cada iteração da perturbação acontece em 10% da solução e cada movimento tem 50% de chance de ser selecionado. O Algoritmo 7 demonstra a perturbação sobre os arquivos. Nas linhas 1 e 2, um arquivo e uma máquina são selecionados aleatoriamente. Na linha 3, é realizado o movimento e, na linha 4, a solução é retornada.

O código apresentado no Algoritmo 8 define a perturbação sobre as máquinas. Nas linhas 1 e 2, são escolhidas a tarefa e máquina, respectivamente. Na linha 3, o movimento é feito e, na linha 4, a solução modificada é retornada. A estrutura de perturbação é apresentada no Algoritmo 9. Na linha 1, o número de movimentos de perturbação é definido de acordo com a quantidade de tarefas da instância. Na linha 3, é escolhido, com 50% de chance para cada, qual dos movimentos será realizado. Nas linhas 4 a 8, a aplicação dos movimentos é realizada e, na linha 11, ocorre o retorno da solução modificada. Os movimentos de perturbação são efetuados independentemente da eventual piora na qualidade da solução.

Algoritmo 7 PerturbateFileAllocation (S)

1: $File \leftarrow$ ArquivoAleatório($S.Files$)
2: $Machine \leftarrow$ MáquinaAleatória($S.Machines$)
3: RealizarMovimento(S, $File$, $Machine$)
4: **Retorne** S

Algoritmo 8 PerturbateMachine (S)

1: $Job \leftarrow$ ArquivoAleatório($S.Jobs$)
2: $Machine \leftarrow$ MáquinaAleatória($S.Machines$)
3: RealizarMovimento(S, Job, $Machine$)
4: **Retorne** S

Algoritmo 9 Perturbação (S)

1: $totalPerturbations \leftarrow |S.Jobs| * 0.1$
2: **Enquanto** $totalPerturbation > 0$ **faça**
3: $id \leftarrow$ selecionaAleatoriamenteEntreZeroEUm()
4: **Se** $id = 0$ **então**
5: PerturbateFileAllocation(S)
6: **senão**
7: PerturbateMachine(S)
8: **Fim-se**
9: $totalPerturbation \leftarrow totalPerturbation - 1$
10: **Fim-enquanto**
11: **Retorne** S

Por fim, o Algoritmo 10 mostra o pseudocódigo da heurística GRASP-ILS para resolver o PEAW-CPU.

Algoritmo 10 GRASP-ILS ($max_iterGRASP$, $max_iterILS$, α)

1: $S^* \leftarrow \{\}$
2: $f^* \leftarrow \infty$
3: $iterGRASP \leftarrow 0$
4: **Enquanto** $iterGRASP < max_iterGRASP$ **faça**
5: $S \leftarrow$ Construtivo(α)
6: $S \leftarrow$ VND(S)
7: **Se** $f(S) < f^*$ **então**
8: $S^* \leftarrow S$
9: $f^* \leftarrow f(S)$
10: **Fim-se**
11: $iterILS \leftarrow 0$
12: **Enquanto** $iterILS < max_iterILS$ **faça**
13: $S \leftarrow$ Perturbação(S)
14: $S \leftarrow$ VND(S)
15: **Se** $f(S) < f^*$ **então**
16: $S^* \leftarrow S$
17: $f^* \leftarrow f(S)$
18: **Fim-se**
19: $iterILS \leftarrow iterILS + 1$
20: **Fim-enquanto**
21: $iterGRASP \leftarrow iterGRASP + 1$
22: **Fim-enquanto**
23: **Retorne** S^*

No próximo capítulo, o novo problema PEAW-CPU/GPU será apresentado em conjunto com a adaptação da heurística baseada em GRASP-ILS proposta para o PEAW-CPU, além de uma implementação exata para solução do problema.

Capítulo 4

Problema de Escalonamento de Tarefas e Armazenamento de Dados em Worflows Científicos - CPU/GPU

4.1 Trabalhos Relacionados

Na última década, um número cada vez maior de estratégias e algoritmos foram propostos para problemas de escalonamento. Muitos desses algoritmos são focados em *workflows* na nuvem, mas são restritos a escalonamento em máquinas que possuem apenas CPU, como pesquisado em Liu et al. [26], o que faz deles não diretamente aplicáveis a nuvens híbridas. Embora existam algumas abordagens para escalonamentos em nuvens híbridas [4, 24, 34, 43], nenhuma das estratégias existentes considera todas as características que são tratadas no modelo a ser apresentado neste capítulo. Desta forma, o problema de escalonamento eficiente de *workflows* em nuvens híbridas ainda é um problema pouco explorado, mas bastante importante.

Shweta et al. [34] propuseram uma ferramenta de *workflow* automatizado para realizar simulações AMBER GPGPUs, uma aplicação de dinâmicas moleculares desenvolvida para a simulação de sistemas biomoleculares. A abordagem facilita o acesso a *clusters* de GPUs mas não propõe uma estratégia de escalonamento na execução do *workflow*. Os autores também não consideraram nuvens híbridas ou variações no *workflow*. Blattner et al. [4] propuseram o *Hybrid Task Graph Scheduler* (HTGS) para escalonar *workflows* em sistemas *multi-core* e *multi*-GPU. A abordagem busca ocupar todas as GPUs e usar todos os recursos computacionais disponíveis. Embora os autores levassem GPUs em conta nessa abordagem, variações no *workflow* ou custos financeiros não foram contabilizados.

Jiang et al. [24] propuseram uma abordagem para escalonar aplicações baseadas em *MapReduce* em ambientes *multi*-GPU. Embora tenha mecanismos sofisticados que se beneficiam de componentes da GPU, essa abordagem se desacoplou do conceito de *workflow* científico. Além disso, ela não tratava o cenário onde as GPUs são virtualizadas. Shirahata et al. [43] propuseram um algoritmo de escalonamento híbrido para aplicações GPU-*based* que busca minimizar o *makespan* e o tempo de espera na fila. Embora os autores tenham alcançado bons resultados em seus experimentos, eles não consideraram o cenário onde GPUs são virtualizadas nem variações nas especificações do *workflow*.

4.2 Formulação Matemática

Neste trabalho, o Problema de Escalonamento de Tarefas e Armazenamento de Dados em *Workflows* Científicos em nuvens com CPU e GPU (PEAW-CPU/GPU) é formulado como um problema de programação inteira. Considere $M = M^{cpu} \cup M^{dual}$ o conjunto de todas as máquinas disponíveis para execução ou armazenamento, onde M^{cpu} é o conjunto de máquinas com um processador CPU e M^{dual} o conjunto de máquinas com um processador CPU e GPU. Cada máquina $j \in M$ tem a capacidade de armazenamento cm_j e um custo financeiro c_j (por unidade de tempo). O conjunto de tarefas N é composto por tarefas que só são executadas em CPU (N^{cpu}), por atividades que só podem ser executadas em GPU (N^{gpu}) e tarefas que podem ser executadas em ambos os processadores (N^{dual}). Para simplificar a notação, denota-se $M_i \subseteq M$ como o conjunto de máquinas que podem executar a tarefa $i \in N$. Note que uma tarefa do conjunto N^{gpu} só pode ser executada no conjunto de máquinas M^{dual}. Por outro lado, tarefas do conjunto $N^{cpu} \cup N^{dual}$ podem ser executadas em todas as máquinas

Define-se $F = S \cup D$ como o conjunto de todos os arquivos, onde cada arquivo $d \in F$ tem tamanho $W(d)$ e pode ser estático (S) ou dinâmico (D). Também determinam-se os requisitos do usuário C_M como o custo máximo financeiro e T_M como o custo máximo de tempo para execução do *workflow*, onde $T = \{1, \ldots, T_M\}$ é o conjunto discreto de tempos onde uma ação de execução de tarefas, escrita ou leitura de arquivos pode ser executada. Ademais, estabelece-se que t_{ij}^{cpu} (t_{ij}^{gpu}) é o tempo de processamento de uma tarefa $i \in N$ executada em uma CPU (GPU) na máquina $j \in M$. Similarmente, define-se $\overrightarrow{t}_{djp}$ como o tempo gasto pela máquina $j \in M$ para ler o arquivo $d \in F$ armazenado na máquina $p \in M$ e $\overleftarrow{t}_{djp}$ como o tempo de a máquina $j \in M$ escrever o arquivo $d \in D$ na máquina $p \in M$. A Tabela 4.1 mostra todos os parâmetros e suas descrições, enquanto na Tabela 4.2 apresentam-se as variáveis e suas definições.

Tabela 4.1: Lista de parâmetros e suas descrições.

Parâmetros	***Descrição***
S	Conjunto de arquivos estáticos.
D	Conjunto de arquivos dinâmicos.
$F = S \cup D$	Conjunto de arquivos.
$O(d)$	Máquina de origem do arquivo estático $d \in S$.
$W(d)$	Tamanho do arquivo $d \in F$.
N^{cpu}	Conjunto de tarefas que só podem ser executadas em CPU.
N^{gpu}	Conjunto de tarefas que só podem ser executadas em GPU.
N^{dual}	Conjunto de tarefas que podem ser executadas em ambos processadores (CPU e GPU).
$N = N^{cpu} \cup N^{gpu} \cup N^{dual}$	Conjunto de tarefas.
M^{cpu}	Conjunto de máquinas com CPU.
M^{dual}	Conjunto de máquinas com GPU e CPU.
$M = M^{cpu} \cup M^{dual}$	Conjunto de máquinas.
$T = \{1, ..., T_M\}$	Conjunto de períodos de tempo viáveis, onde T_M é o tempo máximo de execução do *workflow*.
c_j	Custo financeiro de comprar a máquina $j \in M$ por um período de tempo.
C_M	Custo máximo financeiro estimado para execução do *workflow*.
t_{ij}^{cpu}	Tempo de processamento da tarefa $i \in (N^{cpu} \cup N^{dual})$ na máquina $j \in M$ em CPU.
t_{ij}^{gpu}	Tempo de processamento da tarefa $i \in (N^{gpu} \cup N^{dual})$ na máquina $j \in M^{dual}$ em GPU.
$\overrightarrow{t}_{djp}$	Tempo gasto pela máquina $j \in M$ para ler o arquivo $d \in F$ armazenado na máquina $p \in M$.
$\overleftarrow{t}_{djp}$	Tempo despendido pela máquina $j \in M$ para escrever o arquivo $d \in D$ armazenado na máquina $p \in M$.
$\Delta_{in}(i) \subseteq F$	Conjunto de arquivos necessários para execução da tarefa $i \in N$.
$\Delta_{out}(i) \subseteq D$	Conjunto de arquivos gerados pela tarefa $i \in N$.
cm_j	Capacidade de armazenamento da máquina j.

Tabela 4.2: Lista de variáveis e suas descrições.

Parâmetros	***Descrição***
x_{ijt}^{cpu}	Variável binária que indica se a tarefa $i \in N^{cpu} \cup N^{dual}$ começa sua execução na máquina $j \in M$ em CPU no período $t \in T$, ou não.
x_{ijt}^{gpu}	Variável binária que indica se a tarefa $i \in N^{gpu} \cup N^{dual}$ começa sua execução na máquina $j \in M^{dual}$ em GPU no período $t \in T$, ou não.
a_{idjpt}	Variável binária que indica se a tarefa $i \in N$, executada na máquina $j \in M_i$, começa a ler o arquivo $d \in \Delta_{in}(i)$, armazenado em $p \in M$, no período $t \in T$, ou não.
b_{djpt}	Variável binária que indica se o arquivo $d \in D$ começa a ser transferido da máquina $j \in M_i$ (onde $d \in \Delta_{out}(i)$) para máquina $p \in M$ no período $t \in T$, ou não.
y_{djt}	Variável binária que indica se o arquivo $d \in F$ está armazenado na máquina $j \in M$ no período $t \in T$, ou não.
z_{jt}	Variável binária que indica se a máquina $j \in M$ estava alugada no período $t \in T$, ou não.
z_T	Variável contínua que indica o tempo total para execução do *workflow* (*makespan*).

A função objetivo (4.1) busca tanto a minimização do tempo de execução (*makespan*), quanto do custo financeiro. O parâmetro φ define o peso de cada objetivo e é estabelecido pelo usuário. Note que ambos os objetivos são normalizados usando T_M e C_M respectivamente.

$$\min \varphi(\frac{z_T}{T_M}) + (1-\varphi)(\frac{\sum_{j \in M} \sum_{t \in T} z_{jt} . c_j}{C_M}) \tag{4.1}$$

Restrições (4.2 a 4.4) garantem que toda tarefa deve ser executada. Restrições (4.5) e (4.6) garantem que toda operação de leitura e escrita deve ser terminada, respectivamente.

$$\sum_{j \in M} \sum_{t \in T} x_{ijt}^{cpu} = 1, \qquad \forall i \in N^{cpu} \tag{4.2}$$

$$\sum_{j \in M^{dual}} \sum_{t \in T} x_{ijt}^{gpu} = 1, \qquad \forall i \in N^{gpu} \tag{4.3}$$

$$\sum_{j \in M} \sum_{t \in T} x_{ijt}^{cpu} + \sum_{j \in M^{dual}} \sum_{t \in T} x_{ijt}^{gpu} = 1, \qquad \forall i \in N^{dual} \tag{4.4}$$

$$\sum_{j \in M_i} \sum_{p \in M} \sum_{t \in T} a_{idjpt} = 1, \qquad \forall i \in N, \forall d \in \Delta_{in}(i) \tag{4.5}$$

$$\sum_{j \in M_i} \sum_{p \in M} \sum_{t \in T} b_{djpt} = 1, \qquad \forall d \in D, \text{ tal que } d \in \Delta_{out}(i) \text{ e } i \in N \tag{4.6}$$

Inequações (4.7 a 4.10) garantem que o arquivo $d \in \Delta_{out}(i)$ só pode ser escrito se a tarefa i foi executada no tempo correto.

$$b_{djpt} \leq \sum_{q=1}^{t-t_{ij}^{cpu}} x_{ijq}^{cpu}, \quad \forall d \in D, \forall j \in M^{cpu}, \forall p \in M, \forall t = (t_{ij}^{cpu} + 1) \cdots T_M, \text{ tal que } d \in \Delta_{out}(i) \text{ e } i \in (N^{cpu} \cup N^{dual}) \tag{4.7}$$

$$b_{djpt} \leq \sum_{q=1}^{t-t_{ij}^{gpu}} x_{ijq}^{gpu}, \quad \forall d \in D, \forall j \in M^{dual}, \forall p \in M, \forall t = (t_{ij}^{gpu} + 1) \cdots T_M, \text{ tal que } d \in \Delta_{out}(i) \text{ e } i \in N^{gpu} \tag{4.8}$$

$$b_{djpt} \leq \sum_{q=1}^{t-t_{ij}^{cpu}} x_{ijq}^{cpu}, \quad \forall d \in D, \forall j \in M^{dual}, \forall p \in M, \forall t = (t_{ij}^{cpu} + 1) \cdots T_M, \text{ tal que } d \in \Delta_{out}(i) \text{ e } i \in N^{cpu} \tag{4.9}$$

$$b_{djpt} \leq \sum_{q=1}^{t-t_{ij}^{cpu}} x_{ijq}^{cpu} + \sum_{q=1}^{t-t_{ij}^{gpu}} x_{ijq}^{gpu}, \qquad \forall d \in D, \forall j \in M^{dual}, \forall p \in M, \forall t = (t^{min} + 1) \cdots T_M, \text{ tal que } t^{min} = min\{t_{ij}^{cpu}, t_{ij}^{gpu}\}, d \in \Delta_{out}(i) \text{ e } i \in N^{dual}, \text{ onde } (t - t_{ij}^{cpu}) \geq 1 \text{ e } (t - t_{ij}^{gpu}) \geq 1 \tag{4.10}$$

As restrições (4.11) a (4.14) definem que o arquivo d não pode ser escrito antes do tempo de processamento da tarefa i (responsável pela escrita). Note que as restrições (4.7) a (4.14) funcionam em conjunto para garantir um tempo viável para o processo de escrita.

$$b_{djpt} = 0, \quad \forall d \in D, \forall j \in M^{cpu}, \forall p \in M, \forall t \in [1, t_{ij}^{cpu}] \text{ tal que } d \in \Delta_{out}(i) \text{ e } i \in (N^{cpu} \cup N^{dual}) \tag{4.11}$$

$$b_{djpt} = 0, \quad \forall d \in D, \forall j \in M^{dual}, \forall p \in M, \forall t \in [1, t_{ij}^{gpu}] \text{ tal que } d \in \Delta_{out}(i) \text{ e } i \in N^{gpu} \tag{4.12}$$

$$b_{djpt} = 0, \quad \forall d \in D, \forall j \in M^{dual}, \forall p \in M, \forall t \in [1, t_{ij}^{cpu}] \text{ tal que } d \in \Delta_{out}(i) \text{ e } i \in N^{cpu} \tag{4.13}$$

$$b_{djpt} = 0, \quad \forall d \in D, \forall j \in M^{dual}, \forall p \in M, \forall t \in [1, min\{t_{ij}^{cpu}, t_{ij}^{gpu}\}] \text{ tal que } d \in \Delta_{out}(i) \text{ e } i \in N^{dual} \tag{4.14}$$

Restrições (4.15) a (4.18) garantem que uma tarefa só pode ser executada se todas as leituras terminaram em um tempo viável.

$$x_{ijt}^{cpu} \leq \sum_{p \in M} \sum_{q=1}^{t-\overrightarrow{t}_{djp}} a_{idjpq}, \quad \forall i \in (N^{cpu} \cup N^{dual}), \forall d \in \Delta_{in}(i), \forall j \in M^{cpu}, \forall t \in T \text{ tal que } (t - \overrightarrow{t}_{djp}) \geq 1 \tag{4.15}$$

$$x_{ijt}^{gpu} \leq \sum_{p \in M} \sum_{q=1}^{t-\overrightarrow{t}_{djp}} a_{idjpq}, \quad \forall i \in N^{gpu}, \forall d \in \Delta_{in}(i), \forall j \in M^{dual}, \forall t \in T \text{ tal que } (t - \overrightarrow{t}_{djp}) \geq 1 \tag{4.16}$$

$$x_{ijt}^{cpu} \leq \sum_{p \in M} \sum_{q=1}^{t-\overrightarrow{t}_{djp}} a_{idjpq}, \quad \forall i \in N^{cpu}, \forall d \in \Delta_{in}(i), \\ \forall j \in M^{dual}, \forall t \in T \\ \text{tal que } (t - \overrightarrow{t}_{djp}) \geq 1 \tag{4.17}$$

$$x_{ijt}^{cpu} + x_{ijt}^{gpu} \leq \sum_{p \in M} \sum_{q=1}^{t-\overrightarrow{t}_{djp}} a_{idjpq}, \qquad \forall i \in N^{dual}, \\ \forall d \in \Delta_{in}(i), \forall j \in M^{dual}, \\ \forall t \in T, \text{ tal que } (t - \overrightarrow{t}_{djp}) \geq 1 \tag{4.18}$$

Inequações (4.19) e (4.20) garantem que somente uma ação (execução, leitura ou escrita) pode estar acontecendo em determinado tempo em cada máquina (por exemplo, uma máquina não pode executar e escrever ao mesmo tempo).

$$\sum_{i \in (N^{cpu} \cup N^{dual})} \sum_{q=\max(1, t-t_{ij}^{cpu}+1)}^{t} x_{ijq}^{cpu} + \\ \sum_{i \in (N^{cpu} \cup N^{dual})} \sum_{d \in \Delta_{out}(i)} \sum_{p \in M} \sum_{r=\max(1, t-\overleftarrow{t}_{djp}+1)}^{t} b_{djpr} + \\ \sum_{i \in (N^{cpu} \cup N^{dual})} \sum_{d \in \Delta_{in}(i)} \sum_{p \in M} \sum_{r=\max(1, t-\overrightarrow{t}_{djp}+1)}^{t} a_{idjpr} \leq z_{jt}, \\ \forall j \in M^{cpu}, \forall t \in T \tag{4.19}$$

$$\sum_{i \in (N^{cpu} \cup N^{dual})} \sum_{q=\max(1, t-t_{ij}^{cpu}+1)}^{t} x_{ijq}^{cpu} + \\ \sum_{i \in (N^{gpu} \cup N^{dual})} \sum_{r=\max(1, t-t_{ij}^{gpu}+1)}^{t} x_{ijr}^{gpu} + \\ \sum_{i \in N} \sum_{d \in \Delta_{out}(i)} \sum_{p \in M} \sum_{r=\max(1, t-\overleftarrow{t}_{djp}+1)}^{t} b_{djpr} + \\ \sum_{i \in N} \sum_{d \in \Delta_{in}(i)} \sum_{p \in M} \sum_{r=\max(1, t-\overrightarrow{t}_{djp}+1)}^{t} a_{idjpr} \leq z_{jt}, \\ \forall j \in M^{dual}, \forall t \in T \tag{4.20}$$

As restrições (4.21) e (4.22) junto com as restrições (4.19) e (4.20) garantem a inter-

pretação correta das variáveis z_{jt}. Note que as restrições (4.19) e (4.20) garantem que a máquina está alocada durante uma ação de leitura ou escrita. Já as restrições (4.21) garantem uma alocação viável da máquina durante uma atividade passiva (ser escrita ou lida). Finalmente, as restrições (4.22) garantem períodos de aluguel contínuos.

$$\begin{aligned} &\sum_{i\in N}\sum_{d\in\Delta_{in}(i)}\sum_{p\in M_i}\sum_{r=\max(1,t-\overrightarrow{t}_{dpj}+1)}^{t} a_{idpjr}+ \\ &\sum_{i\in N}\sum_{d\in\Delta_{out}(i)}\sum_{p\in M_i}\sum_{r=\max(1,t-\overleftarrow{t}_{dpj}+1)}^{t} b_{dpjr} \leq z_{jt}.(|N|.|F|), \\ &\qquad\qquad \forall j\in M, \forall t\in T \end{aligned} \tag{4.21}$$

$$z_{j(t+1)} \leq z_{jt}, \qquad \forall j\in M, \forall t\in\{1...T_M-1\} \tag{4.22}$$

Restrições (4.23) estabelecem que não há arquivos dinâmicos alocados no início da execução. Já as restrições (4.24) garante que todos os arquivos estáticos já estão armazenados em suas máquinas de origem.

$$y_{dj1} = 0, \qquad \forall d\in D, \forall j\in M \tag{4.23}$$

$$y_{djt} = 1, \qquad \forall d\in S \mid j\in O(d), \forall t\in T \tag{4.24}$$

Restrições (4.25 a 4.27) condicionam a variável de armazenamento y com a variável de escrita $\overleftarrow{x}$ e a de leitura $\overrightarrow{x}$, garantindo um processo de leitura e escrita viável.

$$\begin{aligned} &y_{dp(t+1)} \leq y_{dpt}, \qquad \forall d\in S, \forall p\in M, \\ &\qquad\qquad \forall t\in\{1...T_M-1\} \end{aligned} \tag{4.25}$$

$$\begin{aligned} &y_{dp(t+1)} \leq y_{dpt} + \sum_{j\in M_i} b_{djp(t-\overleftarrow{t}_{djp}+1)}, \quad \forall d\in D, \\ &\qquad\qquad \forall p\in M, \forall t\in\{1...T_M-1\}, \\ &\qquad \text{tal que } d\in\Delta_{out}(i) \text{ e } (t-\overleftarrow{t}_{djp}+1)\geq 1 \end{aligned} \tag{4.26}$$

$$\begin{aligned} &\sum_{j\in M_i} a_{idjpt} \leq |M_i|.y_{dpt}, \qquad \forall i\in N, \forall d\in\Delta_{in}(i), \\ &\qquad\qquad \forall p\in M, \forall t\in T \end{aligned} \tag{4.27}$$

Em detalhes, as restrições (4.25) e (4.26) garantem que arquivos só serão armzenados

em máquinas depois de terem sido produzidos (escritos) e as restrições (4.27) garantem que arquivos só serão lidos se forem previamente armazenados em uma máquina.

A capacidade de armazenamento das máquinas é restringido por (4.28). As restrições (4.29) relacionam a última operação de escrita com o tempo total de execução *makepan*. Note que no modelo toda tarefa cria arquivos.

$$\sum_{d \in F} y_{djt} W(d) \leq cm_j, \qquad \forall j \in M, \forall t \in T \tag{4.28}$$

$$b_{djpt} \cdot (t + \overleftarrow{t}_{djp}) \leq z_T, \qquad \begin{aligned} &\forall d \in D, \forall j \in M_i, \\ &\forall p \in M, \forall t \in T, \\ &\text{tal que } d \in \Delta_{out}(i) \end{aligned} \tag{4.29}$$

As seguintes restrições operacionais (4.30) devem ser satisfeitas: uma tarefa i só pode começar um processo de leitura se todo arquivo $d \in \Delta_{in}(i)$ já está disponível. Finalmente, as restrições (4.31) são para integralidade e não-negatividade.

$$a_{idjpt} \cdot |\Delta_{in}(i) \cap D| \leq \sum_{\substack{g \in \{\Delta_{in}(i) \cap D\} \\ talque\ g \in \Delta_{out}(h)}} \sum_{l \in M_h} \sum_{o \in M} \sum_{u=1}^{t - \overleftarrow{t}_{glo}} \overleftarrow{x}_{glou} \tag{4.30}$$

$$\begin{aligned} &\forall i \in N, \forall d \in \Delta_{in}(i), \\ &\forall j \in M_i, \forall p \in M, \forall t \in T \end{aligned}$$

$$\begin{aligned} x_{ijt}, a_{idjpt}, b_{djpt}, y_{djt} &\in \{0, 1\} \\ z_T &\in \mathbb{R}^+ \\ \forall i \in N, \forall d &\in \Delta_{in}(i), \\ \forall j \in M_i, \forall p &\in M, \forall t \in T \end{aligned} \tag{4.31}$$

4.3 Abordagem Heurística

Devido às complexas características e ao tamanho das instâncias para o problema apresentado, o uso efetivo de métodos exatos se torna impraticável. Embora instâncias de pequeno porte possam ser resolvidas pelo modelo com o uso de resolvedores apropriados, instâncias de grande porte têm grande consumo de tempo e memória, demandando uma abordagem mais efetiva e rápida. Além disso, mesmo quando um resolvedor atinge uma

solução viável, ele não pode demorar muito tempo para alcançá-la porque pode ocorrer uma adição de tempo não aceitável ao tempo de execução do *workflow*, isto é, o tempo de escalonamento de tarefas não pode comprometer o tempo de execução do *workflow* escalonado.

4.4 GRASP-ILS para o PEAW-CPU/GPU

Em Teylo et al. [45], o desempenho de resolvedores matemáticos é questionado quando usados para resolução do PEAW-CPU. Pela maior complexidade apresentada no modelo matemático da generalização PEAW-CPU/GPU, se tornou necessária uma alternativa para obtenção de soluções para o problema. Como alternativa para o alto custo computacional das soluções oriundas dos resolvedores exatos, este trabalho propõe uma adaptação da estratégia apresentada no Capítulo 3 para o PEAW-CPU. Nas subseções seguintes, serão apresentadas apenas as alterações feitas no estratégia original. As demais partes funcionam analogamente, como definido no Capítulo 3.

4.4.1 Construtivo com Componente Aleatória

A componente construtiva do GRASP funciona de maneira similar à versão para o PEAW-CPU. O pseudo-código do construtivo está apresentado no Algoritmo 11. A diferença está no cálculo do tempo de execução das tarefas e no custo de aluguel da máquina, valores que não eram necessários na versão CPU. A Lista de Canditatos (LC) é preenchida com todas as triplas (*Job*, *Machine*, *Machine*) que possuam um *Job* viável para inserção na solução nas linhas 4 a 6. O custo de cada inserção é calculado (linha 7) e os candidatos são ordenados de acordo com os respectivos custos (linha 12). Após o preenchimento de todos os candidatos possíveis, é montada a LRC com os $\alpha * |LC|$ candidatos com menor custo de inserção (linhas 13 e 14). Dessa LRC é escolhido, aleatoriamente, um candidato que é inserido na solução (linhas 15 e 16).

Diferentemente do custo da versão CPU, onde somente o *makespan* é considerado como função objetivo, o método *CalcularCusto* leva em conta o custo de aluguel das máquinas. O pseudocódigo do método *CalcularCusto* está apresentado no Algoritmo 12. Na linha 1, o mesmo peso é definido para o custo financeiro e o *makespan*. Na linha 2, o valor da função objetivo é inicializado com zero e, nas linhas 3 e 4, os valores de *makespan* e custo são inicializados com o valor da solução de entrada S. Caso o *Job* j possa ser executado em GPU e a *Machine* m tenha uma GPU disponível (linha 5), a previsão do

custo e *makespan* causados pela inserção da tripla são pré-calculados (linhas 6 e 7). Se a *Machine* m ou o *Job* j não suportarem execuções em GPU, as linhas 9 e 10 são executadas. O valor da função objetivo é calculado na linha 12 e o seu valor é retornado na linha 13.

Algoritmo 11 Construtivo (α)

1: $S \leftarrow \{\}$
2: **Enquanto** S não está completo **faça**
3: $LC \leftarrow \{\}$
4: **Para** cada *Job* j não inserido e viável **faça**
5: **Para** cada *Machine* m de execução **faça**
6: **Para** cada *Machine* w de escrita **faça**
7: $custo \leftarrow$ CalcularCusto(S, j, m, w)
8: $LC \leftarrow$ Allocation(j, m, w, $custo$)
9: **Fim-para**
10: **Fim-para**
11: **Fim-para**
12: $LC \leftarrow$ OrdenarPorCusto(LC)
13: $corte \leftarrow |LC| * \alpha$
14: $LRC \leftarrow$constroiLRC($LC, corte$)
15: $candidato \leftarrow$selecionaAleatoriamente(LRC)
16: $S \leftarrow$ insereCandidato($candidato$)
17: **Fim-enquanto**
18: **Retorne** S

Algoritmo 12 CalcularCusto(S, j, m, w)

1: $\varphi \leftarrow 0.5$
2: $OFValue \leftarrow 0$
3: $mkspn_inicial \leftarrow$ calculaMakespan(S)
4: $cost_inicial \leftarrow f(S)$
5: **Se** j aceita GPU e m tem GPU **então**
6: $\quad mkspn \leftarrow$ preverMakeSpanComGPU(j, m, w) + $mkspn_inicial$
7: $\quad cost \leftarrow$ preverCustoGPU(j, m) + $cost_inicial$
8: **senão Se** j não aceita GPU ou m não tem GPU **então**
9: $\quad mkspn \leftarrow$ preverMakeSpanComCPU(j, m, w) + $mkspn_inicial$
10: $\quad cost \leftarrow$ preverCustoCPU(j, m) + $cost_inicial$
11: **Fim-se**
12: $OFValue \leftarrow \varphi$*($mkspn$ / T_M) + (1 - φ)*($cost$ / C_M)
13: **Retorne** $OFValue$

4.4.2 Busca Local e Perturbação

A heurística GRASP-ILS para o PEAW-CPU/GPU utiliza as mesmas vizinhanças de busca e perturbações da versão para o PEAW-CPU/GPU apresentadas anteriormente no Capítulo 3. O método de cálculo de custo apresentado no Algoritmo 12 é utilizado toda vez que é necessário prever o custo de um movimento. O método prevê o custo monetário e temporal levando em conta se uma tarefa é executável em GPU (ou não) e se uma máquina possui GPU (ou não). Vale ressaltar que toda tarefa e toda máquina tem a capacidade de ser realizada e executar em CPU, respectivamente.

Capítulo 5

Resultados Computacionais

O código da heurística híbrida de Teylo et al. [45], cedido pelos autores para a realização dos testes, bem como o da heurística GRASP-ILS proposta neste trabalho, foi implementado na linguagem C++. Testes computacionais foram realizados em um computador equipado com processador Intel Core i7 CPU 8800K @ 3.50GHz, com 16GB de memória RAM e Sistema Operacional Linux Ubuntu versão 18.04 LTS. Todos os experimentos foram executados em uma única *thread* e consideraram os cinco conjuntos de instâncias propostos e utilizados em Teylo et al. [45]: o primeiro denominado *CyberShake* possuindo três instâncias de 30, 50 e 100 tarefas, o segundo denominado *Genome* contendo duas instâncias de 79 e 127 tarefas, o terceiro denominado *Epigenomics* contendo três instâncias de 24, 46 e 100 tarefas, o quarto denominado *Montage* com três instâncias de 25, 50 e 100 tarefas e, o último, chamado *Inspiral* com três instâncias de 30, 50 e 100 tarefas. As instâncias utilizadas são baseadas em *traces* (dados históricos relativos a desempenho) reais de *workflows* científicos. Esses dados foram obtidos por meio da ferramenta Workflow Generator [3], que gera os *traces* baseado em execuções passadas de usuários do Sistema de Gerência de Workflows Pegasus [16]. Os workflows utilizados como estudo de caso são benchmarks bastante conhecidos na área, em especial o workflow Montage que é utilizado como estudo de caso em praticamente todos os trabalhos que estudam escalonamento de workflows [13, 15, 31, 45]. Cada *trace* gerado contém todas as tarefas do *workflow*, o tempo de início e término, os arquivos produzidos e consumidos, os tamanhos dos arquivos e as dependências de dados entre as atividades. Todas as instâncias estão disponibilizadas no GitHub através do link [2].

A execução da heurística de Teylo et al. [45] seguiu os valores de parâmetros de entrada reportados em [45] com os valores para *num_chromosomes*, *num_gerenerations*, *num_elite_set*, *mutation_probability*, *elitism_rate* e *alpha* sendo, respectivamente, 50,

100, 10, 0,10, 0,10 e 0,30. O número de iterações ILS e o valor para α para a heurística GRASP-ILS foram escolhidos por meio de testes apresentados na Seção 5.1.

5.1 Ajuste de Parâmetros

A Tabela 5.1 apresenta os valores de cada parâmetro utilizado nos testes apresentados a seguir. A linha **Instâncias** apresenta o conjunto de instâncias suprimidas do conjunto de comparação, chamadas instâncias de calibração.

Tabela 5.1: Valores e instâncias usados no ajuste de parâmetros.

Parâmetro	**Valores**
Instâncias	[CyberShake_30, Genome.d.35, Epigenomics_24, Montage_25, Inspiral_30]
$\boldsymbol{\alpha}$	[0,1, 0,2, 0,3, 0,4, 0,5, 0,6, 0,7, 0,8, 0,9]
iterILS	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]

A Tabela 5.2, Tabela 5.3 e Tabela 5.4 mostram os resultados dos valores testados para o parâmetro α. Os resultados obtidos são da execução do construtor presente no GRASP-ILS, sem buscas locais ou perturbações. As médias são calculadas sobre dez execuções com sementes aleatórias distintas. Os melhores resultados estão destacados em todas as tabelas. Devido aos melhores resultados em média obtidos por $\alpha = 0.2$, esse valor foi o escolhido para os experimentos realizados a seguir.

Tabela 5.2: Resultados computacionais da parametrização do α.

	$\boldsymbol{\alpha}$					
Instâncias	**0,1**		**0,2**		**0,3**	
	melhor	média	melhor	média	melhor	média
CyberShake_30	17,47	17,48	10,95	**11,32**	**10,93**	11,40
GENOME.d.35	**478,02**	495,39	481,82	**493,70**	490,43	500,50
Epigenomics_24	60,48	62,64	**56,60**	**57,51**	57,02	58,03
Montage_25	**1,20**	**1,26**	1,30	1,37	1,27	1,42
Inspiral_30	15,95	16,88	**15,17**	**15,91**	15,47	16,09

Tabela 5.3: Resultados computacionais da parametrização do α.

	$\boldsymbol{\alpha}$					
Instâncias	**0,4**		**0,5**		**0,6**	
	melhor	média	melhor	média	melhor	média
CyberShake_30	11,48	12,01	11,57	12,45	11,38	12,60
GENOME.d.35	492,15	518,56	534,45	551,41	557,47	581,28
Epigenomics_24	57,30	58,90	57,78	59,16	58,78	61,31
Montage_25	1,45	1,53	1,52	1,62	1,52	1,71
Inspiral_30	15,52	16,35	15,45	17,09	16,12	17,64

Tabela 5.4: Resultados computacionais da parametrização do α.

	α					
Instâncias	**0,7**		**0,8**		**0,9**	
	melhor	média	melhor	média	melhor	média
CyberShake_30	12,15	13,18	12,05	13,85	14,12	15,28
GENOME.d.35	585,72	627,66	621,60	671,33	683,63	784,26
Epigenomics_24	60,08	62,47	60,93	66,14	62,65	68,46
Montage_25	1,68	1,83	1,73	1,91	2,07	2,17
Inspiral_30	17,63	19,23	18,75	20,40	21,68	23,55

Após a escolha do valor de α, o parâmetro $iterILS$ foi avaliado com valores contidos no intervalo apresentado na Tabela 5.1. Os resultados obtidos são oriundos da execução do GRASP-ILS com o mesmo tempo médio gasto pela reprodução do código de Teylo et al. [45]. As iterações GRASP, desta forma, não são limitadas por um parâmetro, mas pelo tempo despendido pela reprodução da abordagem original. A intenção é investigar somente a variabilidade proporcionada pela escolha do número de iterações ILS.

O crescimento do parâmetro $iterILS$ tem relação direta com a intensificação da busca sobre uma solução gerada pelo construtivo. Como as execuções foram parametrizadas sobre o tempo de execução da heurística reproduzida de Teylo et al. [45], aumentar o parâmetro $iterILS$ significa diminuir o número de construções e aumentar os movimentos de perturbação e busca sobre uma mesma solução. Quando $iterILS$ recebe valores acima de 130, uma perda de qualidade de solução foi observada quando comparada aos resultados obtidos com $iterILS = 130$. Devido a este fato, $iterILS = 130$ foi escolhido como a melhor parametrização para o número de iterações ILS. Os valores encontrados nos testes de parametrização do parâmetro $iterILS$ estão nas Tabelas 5.5, 5.6, 5.7 e 5.8

Tabela 5.5: Resultados computacionais da parametrização do iterILS.

	iterILS									
Instâncias	**10**		**20**		**30**		**40**		**50**	
	melhor	média	melhor	média	melhor	média	melhor	média	melhor	média
CyberShake_30	9,22	10,13	9,50	10,14	8,23	9,88	8,70	**9,67**	**8,22**	**9,67**
GENOME.d.35	441,18	441,60	440,95	441,12	**440,57**	441,17	440,63	**440,95**	440,87	441,11
Epigenomics_24	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**
Montage_25	**0,88**	0,90	0,90	**0,91**	0,90	0,92	0,90	**0,91**	0,90	**0,91**
Inspiral_30	13,68	13,70	13,68	13,69	13,68	**13,68**	13,68	13,69	13,68	**13,68**

Tabela 5.6: Resultados computacionais da parametrização do iterILS.

	iterILS									
Instâncias	**60**		**70**		**80**		**90**		**100**	
	melhor	média	melhor	média	melhor	média	melhor	média	melhor	média
CyberShake_30	8,27	9,81	8,75	9,89	**8,22**	9,26	8,68	**9,77**	8,62	9,90
GENOME.d.35	440,82	441,00	440,70	440,90	440,68	**440,89**	**440,52**	**440,89**	**440,52**	440,90
Epigenomics_24	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**
Montage_25	**0,90**	**0,92**	**0,90**	**0,92**	**0,90**	**0,92**	**0,90**	**0,92**	**0,90**	**0,92**
Inspiral_30	**13,67**	**13,68**	**13,67**	**13,68**	**13,67**	**13,68**	13,68	**13,68**	**13,67**	**13,68**

Tabela 5.7: Resultados computacionais da parametrização do iterILS.

	iterILS									
Instâncias	**110**		**120**		**130**		**140**		**150**	
	melhor	média	melhor	média	melhor	média	melhor	média	melhor	média
CyberShake_30	8,32	9,57	8,31	9,57	**8,27**	**9,54**	8,32	9,61	8,68	9,55
GENOME.d.35	440,70	440,90	440,57	440,86	440,57	440,91	**440,45**	440,74	440,58	**440,78**
Epigenomics_24	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**
Montage_25	0,90	0,93	0,90	**0,91**	**0,88**	**0,91**	0,90	**0,91**	0,90	**0,91**
Inspiral_30	13,68	**13,68**	**13,67**	**13,68**	**13,67**	**13,68**	**13,67**	**13,68**	**13,67**	**13,68**

Tabela 5.8: Comparação entre três melhores parametrizações.

	iterILS					
Instâncias	**50**		**90**		**130**	
	melhor	média	melhor	média	melhor	média
CyberShake_30	**8,22**	9,67	8,68	9,77	8,27	**9,54**
GENOME.d.35	440,87	441,11	**440,52**	**440,89**	440,57	440,91
Epigenomics_24	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**	**55,80**
Montage_25	0,90	**0,91**	0,90	0,92	**0,88**	**0,91**
Inspiral_30	13,68	**13,68**	13,68	**13,68**	**13,67**	**13,68**

5.2 Testes Computacionais para o PEAW-CPU

As heurísticas comparadas foram executadas dez vezes para cada instância utilizada, com dez sementes distintas. Foram reportados os custos das melhores soluções, o custo médio das dez soluções e também o tempo médio de execução obtido. A heurística de Teylo et al. [45] foi executada com novas sementes aleatórias, já que as sementes aleatórias aplicadas na execução original em [45] não foram disponibilizadas pelos autores. Embora os resultados atingidos durante a reprodução da heurística tenham sido ligeiramente diferentes dos reportados em Teylo et al. [45] (ver resultados reportados na Tabela 5.9), optou-se por utilizar apenas os resultados alcançados nestas novas execuções, a fim de realizar uma comparação mais justa com a heurística proposta neste trabalho.

Os resultados computacionais serão apresentados na seguinte sequência: (i) a comparação dos valores obtidos pela reprodução do código de Teylo et al. [45] com os valores apresentados em [45]; (ii) a comparação entre a execução da heurística proposta nesta dissertação com a parametrização escolhida, e os valores alcançados pela reprodução do código de Teylo et al. [45], acompanhada de análise de significância estatística; e (iii) os resultados obtidos pela execução da heurística GRASP-ILS para o PEAW-CPU/GPU, em conjunto com os resultados atingidos pela execução do modelo matemático.

Na Tabela 5.9, estão apresentados os resultados obtidos com a reprodução do código de Teylo et al. [45]. Na primeira coluna, estão os nomes das instâncias. Na segunda e terceira, estão os melhores resultados e os tempos de execução da heurística original, apresentadas em [45]. Nas três colunas seguintes, estão os melhores resultados, média de dez execuções

e tempos médios de execução, respectivamente, obtidos com a reprodução. Todos os tempos apresentados são em minutos, como originalmente reportado por Teylo et al. [45]. É possível observar que a reprodução obteve melhora na maioria das instâncias, quando comparado com os resultados apresentados em [45]. De qualquer forma, os resultados da reprodução foram utilizados como base de comparação em todos os testes a seguir.

Tabela 5.9: Resultados computacionais da reprodução do código de Teylo et al. [45]

	Teylo et al.		**Reprodução**		
Instâncias	**Melhor**	**Tempo (m)**	**Melhor**	**Média**	**Tempo (m)**
CyberShake_30	10,25	0,39	**10,15**	10,29	1,22
CyberShake_50	**12,46**	2,19	13,83	13,93	3,53
CyberShake_100	**14,52**	8,11	14,88	22,08	16,79
GENOME.d.35	444,91	4,05	**441,38**	443,65	23,59
GENOME.d.70	833,98	6,79	**827,90**	833,11	36,23
Epigenomics_24	**55,80**	0,03	**55,80**	55,80	36,61
Epigenomics_46	99,27	0,48	**98,43**	99,67	37,82
Epigenomics_100	889,65	4,02	**889,25**	889,47	44,03
Montage_25	0,95	0,05	**0,90**	0,92	44,42
Montage_50	1,88	0,10	**1,82**	1,83	45,76
Montage_100	**3,93**	3,33	**3,93**	4,06	55,70
Inspiral_30	13,95	0,17	**13,70**	13,71	56,37
Inspiral_50	22,41	0,79	**22,38**	22,39	58,17
Inspiral_100	40,76	1,55	**40,23**	40,50	66,92

A Tabela 5.10 mostra os resultados obtidos quando o GRASP-ILS foi executado com os mesmos tempos médios gastos pela reprodução. Na primeira coluna, estão os nomes das instâncias. Na segunda, terceira e quarta, estão os melhores resultados, médias e tempos de execução, respectivamente, obtidos com a reprodução. Nas três colunas seguintes, estão os melhores resultados, média de dez execuções e tempos de execução, respectivamente, obtidos pela nova proposta. É possível observar sete vitórias e duas derrotas em relação à melhor solução e seis vitórias e três derrotas em relação à solução média.

Analisando a parametrização escolhida para o GRASP-ILS ($\alpha = 0.2$ e $iterILS =$

130), é possível verificar uma melhora na qualidade de solução média na maioria das instâncias da literatura. Nos conjuntos de instâncias *CyberShake*, *Inspiral* e *Epigenomics*, o algoritmo alcançou melhora nas médias e melhores soluções nas duas instâncias de cada conjunto. No conjunto *Montage*, o GRASP-ILS obteve melhores resultados em uma de duas instâncias, em relação à melhor solução encontrada, mas Teylo et al. [45] obtiveram melhores médias em ambas as instâncias (*Montage_50* e *Montage_100*). Finalmente, no conjunto *Genome* a heurística GRASP-ILS não superou o desempenho da heurística de [45].

Todos os resultados obtidos foram avaliados usando o teste Mann-Whitney-Wilcoxon [30], com um nível de significância estatística de 0,05. Para a aplicação do teste, foram utilizados os dez valores obtidos na execução de cada instância, separadamente, para cada uma das duas heurísticas. Os *p-values*, apresentados na última coluna da Tabela 5.10, são oriundos da execução do teste usando a linguagem R, versão 3.6.2. [35].

Tabela 5.10: Resultados da nova proposta.

	Reprodução			**Heurística Proposta**		
Instâncias	**Melhor**	**Média**	**Tempo (m)**	**Melhor**	**Média**	***p-value***
CyberShake_50	13,83	13,93	3,53	**11,77**	**11,79**	0,002
CyberShake_100	14,88	22,08	16,79	**12,38**	**13,90**	0,002
GENOME.d.70	**827,90**	**833,11**	36,23	898,63	985,04	0,002
Epigenomics_46	98,43	99,67	37,82	**98,08**	**98,19**	0,002
Epigenomics_100	889,25	889,47	44,03	**885,08**	**888,83**	0,020
Montage_50	1,82	**1,83**	45,76	**1,80**	2,10	0,006
Montage_100	**3,93**	**4,06**	55,70	4,03	5,48	0,004
Inspiral_50	22,38	22,39	58,17	**22,32**	**22,34**	0,008
Inspiral_100	40,23	40,50	66,92	**40,05**	**40,26**	0,014

5.3 Testes Computacionais para o PEAW-CPU/GPU

A formulação matemática apresentada na Seção 4.2 foi implementada com o compilador gcc versão 4.8.5 e resolvida usando o IBM ILOG CPLEX versão 12.5.1. Os experimentos com o CPLEX foram realizados em um computador pessoal Intel®Xeon(R) CPU

E5-2620 v3 @ 2.40GHz, com 64GB RAM usando CentOS 7.6.1810 OS. Os valores dos parâmetros do GRASP-ILS são os mesmos da parametrização adotada para o PEAW-CPU. A heurística proposta para o PEAW-CPU/GPU foi executada em um computador pessoal Intel®Core™ i7-8700K CPU @ 3.70GHz com 16GB RAM usando S.O Linux Ubuntu versão 18.04 LTS com paralelismo desabilitado. Para comparação direta entre o tempo de execução dos testes para os dois processadores supracitados, um fator multiplicativo de 0.62 foi aplicado para o processor mais lento Xeon(R) CPU E5-2620 v3 @ 2.40GHz. Os valores para a extração do fator multiplicativo estão disponíveis em [1].

A função objetivo requer que ambos C_M e T_M sejam calculados previamente para cada instância, e inseridos, como parâmetros de entrada, para a formulação matemática e para a heurística. A abordagem inicial para estimar esses parâmetros foi prever a solução mais cara e com maior *makespan* para cada instância, e usar esses valores encontrados como C_M e T_M. Entretanto, como o crescimento de variáveis e de restrições é exponencial com o aumento da instância, usar o maior *makespan* se tornou inviável. Para contornar este problema, testes preliminares com valores de C_M e T_M foram executados, usando somente o método construtivo da heurística GRASP-ILS com o intuito de encontrar valores menores para os parâmetros, mas ainda, teoricamente, longe dos ótimos. A Tabela 5.11 apresenta os valores utilizados para os parâmetros nos experimentos seguintes, além do número de tarefas e arquivos de cada instância.

No primeiro experimento, um conjunto de instâncias de pequeno porte, chamadas *Toy*, foi usado na comparação do modelo matemático com a heurística, devido à limitação do modelo em executar instâncias com tamanhos maiores, mais próximas dos casos reais. A Tabela 5.12 reporta os experimentos computacionais da heurística GRASP-ILS e da execução do CPLEX com a formulação matemática. A primeira coluna identifica a instância. A segunda reporta, para cada instância, o valor da melhor solução encontrada após dez execuções da heurística. A terceira coluna apresenta a média dessas dez execuções. A quarta e quinta colunas mostram, respectivamente, o *makespan* e o custo financeiro da melhor solução encontrada pela heurística. A sexta coluna indica o tempo médio destas execuções em segundos. Na sétima coluna, estão os valores de soluções ótimos para as instâncias, seguidos do *makespan* e custo financeiro ótimos. A última coluna apresenta o tempo de execução gasto pelo modelo matemático para cada instância após a conversão com o fator multiplicativo de 0,62 obtido por meio do *benchmark* [1]. Os resultados mostram que o modelo matemático consegue reportar soluções ótimas para todas as instâncias. A heurística GRASP-ILS alcançou a mesma solução que o modelo matemático em todas as instâncias tanto em média, quanto em melhor solução. É impor-

tante perceber que os tempos de execução da heurística são significativamente menores do que os reportados pelo CPLEX.

No segundo experimento, somente a heurística foi executada para instâncias maiores. Essas instâncias foram geradas artificialmente usando informações de execuções de *workflows* reais. Como as instâncias *Montage, CyberShake, Inspiral, Epigenomics e Sipht* não consideram execuções em GPUs, elas foram artificialmente modificadas para inserir características de execução e custo de GPUs para tarefas e máquinas, respectivamente. Para a geração desses instâncias artificiais, cada tarefa tem uma probabibilidade de 0,5 de ser executável em GPU. Caso ela seja, o tempo de execução da tarefa em GPU é escolhido aleatoriamente do intervalo $[cpu_{time} * 0,3, cpu_{time} * 0,7]$. Para estipular artificialmente o desempenho das máquinas, cada uma tem probabilidade 0,5 de ter GPU. Caso exista GPU na máquina, o desempenho da execução em GPU é selecionado aleatoriamente do intervalo $[cpu_{slowdown} * 0,3, cpu_{slowdown} * 0,7]$. O preço de aluguel das máquinas também é alterado de acordo com a presença de GPU, com o custo variando no intervalo $[cpu_{cost} * 8,0, cpu_{cost} * 15,0]$. A Tabela 5.13 mostra os experimentos computacionais sobre a heurística GRASP-ILS para essas instâncias artificiais supracitadas. Por serem consideravelmente maiores dos que as instâncias *Toy*, não foi possível a execução do modelo matemático com o resolvedor CPLEX, uma vez que esse resolvedor não conseguiu encontrar soluções inteiras. Para essas instâncias grandes, a heurística proposta teve um limite de tempo de 1800 segundos de execução, encontrando soluções viáveis para todas as instâncias.

No caso da instância Montage_100, embora o tempo necessário para a criação do escalonamento tenha sido até seis vezes maior do que o *makespan* encontrado na solução, os escalonamentos são criados previamente ao uso da nuvem computacional. Dessa maneira, um custo maior do que a execução para a criação de um plano de execução não influencia o custo de aluguel das nuvens computacionais, já que os escalonamentos podem ser feitos *offline* em uma máquina não alugada e enviados para a nuvem computacional em conjunto com o *workflow*.

Tabela 5.11: Características das instâncias

Instâncias	***Tarefas***	***Arquivos***	T_M	C_M
3_toy_5_A	2	3	10	120,00
5_toy_5_A	2	3	10	120,00
3_toy_5_B	2	3	35	120,00
5_toy_5_B	2	3	16	108,00
3_toy_5_C	1	4	6	80,00
5_toy_5_C	1	4	14	96,00
3_toy_10_A	4	6	46	168,00
5_toy_10_A	4	6	82	268,00
3_toy_10_B	3	7	16	198,00
5_toy_10_B	3	7	22	300,00
3_toy_10_C	4	6	42	440,00
5_toy_10_C	4	6	68	450,00
3_toy_15_A	5	10	36	154,00
5_toy_15_A	5	10	38	88,00
3_toy_15_B	5	10	52	54,00
5_toy_15_B	5	10	44	46,00
3_toy_15_C	5	10	54	136,00
5_toy_15_C	5	10	54	56,00
CyberShake_30	30	49	2820	282,20
CyberShake_50	50	79	2332	1758,04
CyberShake_100	100	154	4604	5353,16
Epigenomics_24	24	38	6788	9888,78
Epigenomics_46	46	71	21252	3377,80
Epigenomics_100	100	152	106878	197180,00
Montage_25	25	38	340	454,36
Montage_50	50	53	816	138,62
Montage_100	100	93	1018	1792,26
Inspiral_30	30	47	2456	772,28
Inspiral_50	50	77	3006	2044,60
Inspiral_100	100	151	6140	3008,92
Sipht_30	29	963	5888	669,40
Sipht_60	58	1049	8774	4422,60
Sipht_100	97	1121	15160	2256,00

Tabela 5.12: Resultados computacionais da heurística GRASP-ILS e do modelo matemático para instâncias pequenas

	Heurística					**Modelo**				
Instâncias	**Melhor**	**Média**	**Mksp**	**Custo**	**T(s)**	**Opt**	**Mksp**	**Custo**	**T(s)**	GAP(%)
3_toy_15_A	0,55	0,55	7	60	0,03	0,55	7	60	0,09	0,0%
3_toy_15_B	0,35	0,35	10	54	0,02	0,35	10	54	0,71	0,0%
3_toy_15_C	0,88	0,88	7	60	0,02	0,88	7	60	0,02	0,0%
5_toy_5_A	0,55	0,55	7	60	0,04	0,55	7	60	0,12	0,0%
5_toy_5_B	0,53	0,53	10	54	0,04	0,53	10	54	0,26	0,0%
5_toy_5_C	0,67	0,67	11	60	0,03	0,67	11	60	0,12	0,0%
3_toy_10_A	0,47	0,47	26	65	0,06	0,47	26	65	48,50	0,0%
3_toy_10_B	0,53	0,53	10	98	0,05	0,53	10	98	1,20	0,0%
3_toy_10_C	0,44	0,44	20	190	0,06	0,44	20	190	7,14	0,0%
5_toy_10_A	0,42	0,42	40	96	0,10	0,42	40	96	1294,34	0,0%
5_toy_10_B	0,47	0,47	11	143	0,10	0,47	11	143	3,74	0,0%
5_toy_10_C	0,40	0,40	28	183	0,10	0,40	28	183	139,68	0,0%
3_toy_15_A	0,38	0,38	23	22	0,11	0,38	23	22	42,47	0,0%
3_toy_15_B	0,42	0,42	23	22	0,10	0,42	23	22	23,00	0,0%
3_toy_15_C	0,34	0,34	27	26	0,11	0,34	27	26	192,53	0,0%
5_toy_15_A	0,41	0,41	20	28	0,18	0,41	20	28	222,78	0,0%
5_toy_15_B	0,44	0,44	21	20	0,18	0,44	21	20	193,96	0,0%
5_toy_15_C	0,45	0,45	26	25	0,18	0,45	26	25	3655,89	0,0%

Tabela 5.13: Resultados computacionais da heurística GRASP-ILS para instâncias artificiais grandes

	Heurística				
Instâncias	**Melhor**	**Média**	**Makespan**	**Custo**	**Tempo(s)**
CyberShake_30_gpu	0,48	0,48	934	179,30	28,82
CyberShake_50_gpu	0,24	0,24	566	400,38	451,14
CyberShake_100_gpu	0,20	0,21	964	972,50	1800,06
Epigenomics_24_gpu	0,42	0,42	2,740	4304,11	27,16
Epigenomics_46_gpu	0,44	0,44	11295	1163,70	281,32
Epigenomics_100_gpu	0,48	0,48	48,498	99583,40	1800,04
Montage_25_gpu	0,36	0,36	96	198,81	19,98
Montage_50_gpu	0,36	0,36	268	53,40	367,15
Montage_100_gpu	0,43	0,44	356	934,05	1800,03
Inspiral_30_gpu	0,43	0,43	1,05	160,40	39,42
Inspiral_50_gpu	0,43	0,43	1470	760,85	694,86
Inspiral_100_gpu	0,42	0,43	3058	1055,46	1800,02
Sipht_30_gpu	0,47	0,47	2626	323,90	214,61
Sipht_60_gpu	0,33	0,33	4133	869,90	1800,01
Sipht_100_gpu	0,40	0,41	5053	1066,20	1800,08

Capítulo 6

Conclusões e Trabalhos Futuros

O gerenciamento de dados e o escalonamento de tarefas em WfCs é considerado um problema com grande explosão combinatória, principalmente quando se adiciona o tratamento necessário para cenários com máquinas em nuvens. O problema se intensifica quando se trata de um sistema de máquinas heterogêneas e com capacidade (ou não) de suportar execuções em GPU. Um problema de escalonamento é, conhecidamente, da classe $\mathcal{NP}$-Difícil, mesmo no seu formato mais simples. Este trabalho propõe uma nova heurística, baseada nas metaheurísticas GRASP e ILS, para o problema que considera tanto a alocação de arquivos quanto o escalonamento de tarefas, aumentando a complexidade combinatória. Resultados melhores do que os apresentados na literatura foram encontrados para a versão PEAW-CPU, onde somente máquinas com execução em CPU são consideradas.

Uma nova modelagem matemática, em conjunto com a adaptação da nova heurística híbrida proposta inicialmente para o PEAW-CPU, foi apresentada para a versão PEAW-CPU/GPU, que considera máquinas que podem, ou não, ter processamento em GPU e tarefas que podem, ou não, ser executadas em GPU.

A heurística híbrida GRASP-ILS foi parametrizada por meio de testes empíricos, onde diferentes valores para os parâmetros α e número de iterações ILS foram avaliados. Os tempos de execução foram limitados aos da reprodução da heurística original, por meio da execução do código desenvolvido em [45]. A execução de testes em instâncias da literatura para o PEAW-CPU mostraram uma melhora significativa nos custos encontrados para as melhores soluções (sete dos nove casos) e média de solução após dez execuções (seis de nove casos). Todos os resultados obtidos na comparação das heurísticas tiveram significância estatística avaliada com o teste Mann-Whitney-Wilcoxon.

A mesma parametrização da versão PEAW-CPU foi utilizada na heurística GRASP-ILS para o PEAW-CPU/GPU. Algumas instâncias pequenas denominadas *Toy* foram geradas e utilizadas para a comparação da heurística com o modelo matemático proposto neste trabalho. A heurística proposta alcançou todos os ótimos encontrados pelo modelo exato em um tempo computacional menor. Devido à ausência de instâncias reais, um conjunto de instâncias artificiais com uso de GPU foi gerado sobre as instâncias presentes em [45]. O conjunto de instâncias artificiais não pôde ser utilizado com o modelo matemático, uma vez que a modelagem se tornou inviável computacionalmente. A heurística se mostrou eficaz (e necessária) ao resolver todas as instâncias artificiais propostas, encontrando soluções viáveis.

Como trabalhos futuros, pretende-se iniciar as investigações sobre a inserção da mineração de padrões e as suas aplicações sobre uma solução, tanto para o PEAW-CPU, quanto para o PEAW-CPU/GPU. Seguindo este estudo, será implementada a criação de movimentos de busca e perturbações, em conjunto com o uso de métodos construtivos modificados que se beneficiem dos padrões obtidos. Adquirir novas instâncias baseadas em *workflows* reais também é um foco deste estudo, validando ainda mais a eficiência e usabilidade desta nova abordagem. Sobre as soluções de instâncias reais, a execução do *workflow* gerado pela heurística GRASP-ILS por ambientes e *frameworks* preparadas para execução de WfCs será relevante, uma vez que será possível a comparação com escalonadores já existentes e consolidados.

Referências

[1] Cpu-benchmark. https://www.cpubenchmark.net/compare/Intel-Xeon-E5-2620-v3-vs-Intel-i7-8700K/2418vs3098, Janeiro 2020.

[2] Github repositório de instâncias. https://github.com/d5storm/hea-mestrado, Janeiro 2020.

[3] Workflow generator. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator, Janeiro 2020.

[4] Blattner, T.; Keyrouz, W.; Bhattacharyya, S. S.; Halem, M.; Brady, M. A hybrid task graph scheduler for high performance image processing workflows. *Signal Processing Systems 89* (2017), 457–467.

[5] Blythe, J.; Jain, S.; Deelman, E.; Gil, Y.; Vahi, K.; Mandal, A.; Kennedy, K. Task scheduling strategies for workflow-based applications in grids. *IEEE International Symposium on Cluster Computing and the Grid* (2005), 759–767.

[6] Broder, A.; Garcia-Pueyo, L.; Josifovski, V.; Vassilvitskii, S.; Venkatesan, S. Scalable k-means by ranked retrieval. *Proceedings of the 7th ACM International Conference on Web Search and Data Mining* (2014), 233–242.

[7] Bryk, P.; Malawski, M.; Juve, G.; Deelman, E. Storage-aware algorithms for scheduling of workflow ensembles in clouds. *Journal of Grid Computing 14* (2015), 1–20.

[8] Chen, J.; Chen, Y.; Du, X.; Li, C.; Lu, J.; Zhao, S.; Zhou, X. Big data challenge: a data management perspective. *Frontiers of Computer Science 7* (2013), 157–164.

[9] Chen, W. N.; Zhang, J. An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 39* (2009), 29–43.

[10] Chvátal, V. Linear programming. *W.H. Freeman 1* (1983).

[11] Cook, W.; Cunningham, W.and Pulleyblank, W.; Schrijver, A. Combinatorial optimization. *John Wiley & Sons* (1998).

[12] de Mello Santos, L. F. Metaheurística híbrida GRASP-MD: Novas aplicações e paralelização. *Instituto de Computação, Universidade Federal Fluminense, Programa de Pós-Graduação em Computação, Mestrado* (2006).

[13] de Oliveira, D.; Ocaña, K. A.; Ogasawara, E.; Dias, J.; Gonçalves, J.; Baião, F.; Mattoso, M. Performance evaluation of parallel strategies in public clouds: A study with phylogenomic workflows. *Future Generation Computer Systems 29* (2013), 1816 – 1825.

[14] De Oliveira, D.; Ocaña, K. A. C. S.; Baião, F.; Mattoso, M. A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *Journal of Grid Computing 10* (2012), 521–552.

[15] de Oliveira, D.; Viana, V.; Ogasawara, E.; Ocaña, K.; Mattoso, M. Dimensioning the virtual cluster for parallel scientific workflows in clouds. In *Proceedings of the 4th ACM workshop on Scientific cloud computing* (New York, NY, USA, 2013), Science Cloud '13, ACM, pp. 5–12.

[16] Deelman, E.; Blythe, J.; Gil, Y.; Kesselman, C.; Mehta, G.; Patil, S.; Su, M.-H.; Vahi, K.; Livny, M. Pegasus: Mapping scientific workflows onto the grid. In *undefined* (2004), Springer, pp. 11–20.

[17] Dong, F.; Akl, S. G. Scheduling algorithms for grid computing: State of the art and open problems. *Technical Report* (2006).

[18] Durillo, J. J.; Fard, H. M.; Prodan, R. M. A multi-objective list-based method for workflow scheduling. *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings* (2012), 185–192.

[19] Fakhfakh, F.; Kacem, H. H.; Kacem, A. H. Workflow scheduling in cloud computing: A survey. *IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations* (2014), 372—378.

[20] Feo, T. A.; Resende, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6* (1995), 109–133.

[21] Gendreau, M.; Potvin, J.-Y. *Handbook of Metaheuristics*, 2nd ed., vol. 146 of *International Series in Operations Research & Management Science*. Springer, 2010.

[22] Hu, Y.; Xing, L.; Zhang, W.; Xiao, W.; Tang, D. A knowledge-based ant colony optimization for a grid workflow scheduling problem. *Advances in Swarm Intelligence 6145* (2010), 241–248.

[23] Isard, M.; Prabhakaran, V.; Currey, J.; Wieder, U.; Talwar, K.; Goldberg, A. Quincy: Fair scheduling for distributed computing clusters. In *22Nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), pp. 261–276.

[24] Jiang, H.; Chen, Y.; Qiao, Z.; Weng, T.; Li, K. Scaling up mapreduce-based big data processing on multi-gpu systems. *Cluster Computing 18* (2015), 369–383.

[25] Kennedy, J.; Eberhart., R. Particle swarm optimization. *IEEE International Conference on Neural Networks* (1995), 1942–1948.

[26] Liu, J.; Pacitti, E.; Valduriez, P.; Mattoso, M. A survey of data-intensive scientific workflow management. *Journal of Grid Computing 13* (2015), 457–493.

[27] Liu, J.; Silva, V.; Pacitti, E.; Valduriez, P.; Mattoso, M. Scientific workflow partitioning in multisite cloud. In *Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops* (2007), pp. 105–116.

[28] Mattoso, M.; Werner, C.; Travassos, G. H.; Braganholo, V.; Ogasawara, E.; Oliveira, D.; Cruz, S.; Martinho, W.; Murta, L. Towards supporting the life cycle of large scale scientific experiments. *International Journal of Business Process Integration and Management 5* (2010), 79–92.

[29] Nemhauser, G.; Wolsey, L. Integer and combinatorial optimization. *John Wiley & Sons* (1999).

[30] Neuhäuser, M. *Wilcoxon–Mann–Whitney Test.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 1656–1658.

[31] Ogasawara, E. S.; Dias, J.; Silva, V.; Chirigati, F. S.; de Oliveira, D.; Porto, F.; Valduriez, P.; Mattoso, M. Chiron: A Parallel Engine for Algebraic Scientific Workflows. *Concurrency and Computation: Practice and Experience 25* (2013), 2327–2341.

[32] Oliveira, D. E. D.; Boeres, C.; Fausti, A.; Porto, F. Avaliação da localidade de dados intermediários na execução paralela de workflows big data. *XXX Brazilian Simposyum on Databases* (2015), 29–40.

[33] Pandey, S.; Wu, L.; Guru, S. M.; Buyya, R. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. *24th International Conference on Advanced Information Networking and Applications* (2010), 400–407.

[34] Purawat, S.; Ieong, P. U.; Malmstrom, R. D.; Chan, G. J.; Yeung, A. K.; Walker, R. C.; Altintas, I.; Amaro, R. E. A kepler workflow tool for reproducible amber gpu molecular dynamics. *Biophysical Journal 112* (2017), 2469 – 2474.

[35] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2017.

[36] Ranganathan, K.; Foster, I. Decoupling computation and data scheduling in distributed data-intensive applications. In *11th IEEE International Symposium on High Performance Distributed Computing* (2002), pp. 352–358.

[37] Ribeiro, C.; Urrutia, S. Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research 179* (2007), 775–787.

[38] Ribeiro, M. H.; Plastino, A.; Martins, S. L. Hybridization of GRASP metaheuristic with data mining techniques. *Journal of Mathematical Modelling Algorithms 5* (2006), 23–41.

[39] Ribeiro, M. H.; Trindade, V. F.; Plastino, A.; Martins, S. L. Hybridization of GRASP metaheuristics with data mining techniques. In *Proceedings of the ECAI workshop on hybrid metaheuristics* (2004), pp. 69–78.

[40] Santos, L. F.; Martins, S. L.; Plastino, A. Applications of the DM-GRASP heuristic: a survey. *International Transactions in Operational Research 15* (2008), 387–416.

[41] Santos, L. F.; Ribeiro, M. H.; Plastino, A.; Martins, S. L. A hybrid GRASP with data mining for the maximum diversity problem. In *Proceedings of the International Workshop on Hybrid Metaheuristics* (Barcelona, Spain, 2005), vol. 3636 of *Lecture Notes in Computer Science*, pp. 116–127.

[42] Schmidt, S.; Schulz, V. Pareto-curve continuation in multi-objective optimization. *Pacific Journal of Optimization 4* (2008).

[43] Shirahata, K.; Sato, H.; Matsuoka, S. Hybrid map task scheduling for gpu-based heterogeneous clusters. In *2010 IEEE International Conference on Cloud Computing Technology and Science* (2010), pp. 733–740.

[44] Szabo, C.; Sheng, Q. Z.; Kroeger, T.; Zhang, Y.; Yu, J. Science in the cloud: Allocation and execution of data-intensive scientific workflows. *Journal of Grid Computing 12* (2013), 245–264.

[45] Teylo, L.; de Paula, U.; Frota, Y.; de Oliveira, D.; M.A.Drummond, L. A hybrid evolutionary algorithm for task scheduling and data assignment of data-intensive scientific workflows on clouds. *Future Generations Computer Systems 76* (2017), 1–17.

[46] Topcuoglu, H.; Hariri, S.; Wu, M.-Y. Performance-effective and low- complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems 13* (2002), 260–274.

[47] Ullman, J. D. Polynomial complete scheduling problems. *SIGOPS* (1973), 96–101.

[48] Wang, M.; Zhang, J.; Dong, F.; Luo, J. Data placement and task scheduling optimization for data intensive scientific workflow in multiple data centers environment. *Advanced Cloud and Big Data* (2014), 77–84.

[49] Yu, J.; Buyya, R. R. K. Workflow scheduling algorithms for grid computing. *Springer Berlin Heidelberg 146* (2008), 173–214.

[50] Yuan, D.; Yang, Y.; Liu, X.; Chen, J. A data placement strategy in scientific cloud workflows. *Future Generation Computer Systems* (2010), 1200–1214.