UNIVERSIDADE FEDERAL FLUMINENSE

ELLIOD GARCIA CIEZA

METAHEURÍSTICAS PARALELAS PARA ESCALONAMENTO DE WORKFLOWS CIENTÍFICOS EM AMBIENTES HÍBRIDOS CPU-GPU

NITERÓI 2019 UNIVERSIDADE FEDERAL FLUMINENSE

ELLIOD GARCIA CIEZA

METAHEURÍSTICAS PARALELAS PARA ESCALONAMENTO DE WORKFLOWS CIENTÍFICOS EM AMBIENTES HÍBRIDOS CPU-GPU

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: Sistemas de Computação.

Orientador: Lúcia Maria de Assumpção Drummond

> NITERÓI 2019

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

C569m Cieza, Elliod Garcia METAHEURÍSTICAS PARALELAS PARA ESCALONAMENTO DE WORKFLOWS CIENTÍFICOS EM AMBIENTES HÍBRIDOS CPU-GPU / Elliod Garcia Cieza ; Lúcia Drummond, orientadora. Niterói, 2019. 47 f. Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2019. DOI: http://dx.doi.org/10.22409/PGC.2019.m.10905250796 1. Escalonamento de tarefas. 2. Metaheurística. 3. Computação em nuvem. 4. Produção intelectual. I. Drummond, Lúcia, orientadora. II. Universidade Federal Fluminense. Instituto de Computação. III. Título. CDD -

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

Elliod Garcia Cieza

Metaheurísticas Paralelas para Escalonamento de *Workflows* Científicos em Ambientes Híbridos CPU-GPU

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: Sistemas de Computação.

Aprovada em Julho de 2019.

BANCA EXAMINADORA

() c

Profa. Lúcia M. A. Drummond - Orientador, IC-UFF

RI К.

Profa. Cristiana Barbosa Bentes, UER J

Prof. Luiz Satoru Ochi, IC-UFF

Niterói 2019

Aos meus pais.

Agradecimentos

Agradeço a todos professores do Instituto de Computação por todo conhecimento concedido, em especial à professora Lúcia pela sua inexaurível paciência e confiança depositada em mim durante todas etapas da orientação.

Agradeço a todos colegas e amigos com quem tive o prazer de conviver.

Agradeço ao CNPq pelo apoio financeiro que foi dado.

Resumo

Workflows são amplamente usados por cientistas para automatizar experimentos científicos em larga escala. Um *workflow* científico descreve as dependências entre tarefas, e é comumente descrito por um grafo acíclico dirigido em que os nós são tarefas e as arestas denotam a interdependência de tarefas. O escalonamento de um *workflow* consiste em atribuir suas tarefas a recursos computacionais com o intuito de tornar a execução mais eficiente. Normalmente, workflows científicos são compostos por milhares de tarefas, e avaliar todas possíveis distribuições de tarefas resultantes de diferentes escalonamentos resultaria em tempos computacionais proibitivos, visto que este é um problema NP-Completo. Portanto é importante estudar algoritmos de escalonamento eficientes. Neste trabalho é proposta uma metaheurística Particle Swarm Objective (PSO) para o Task Scheduling and Data File Assignment Problem (TaSDAP), que leva em consideração o tempo de execução das tarefas, os tempos de transferências de arquivos e um conjunto de máquinas virtuais heterogêneas. Como base para este método, foi usada a metaheurística evolutiva Hybrid Evolutionary Algorithm to Task Scheduling and Data File Assignment Problem (HEA-TaSDAP). Adicionalmente, uma versão paralela desta metaheurística (PHEA-TaSDAP) foi desenvolvida a fim de viabilizar o escalonamento de workflows com mais de 100 tarefas em um tempos computacionais razoáveis.

Palavras-chave: Workflow, Escalonamento, Metaheuristica, GPU.

Abstract

Workflows have been widely used by the scientific community to model large-scale scientific experiments. A scientific workflow describes the dependencies between the tasks and is commonly described as a directed acyclic graph, where the nodes are tasks and the edges denote task interdependencies. The scheduling of a workflow consists in assigning its tasks to computational resources with the purpose of improving the execution efficiency. Usually, scientific workflows are comprised of thousands of tasks, and evaluating all possible task scheduling distributions would result in prohibitive computation times, since this is a NP-Complete problem. Hence it becomes important to study efficient scheduling algorithms. In this work we propose a Particle Swarm Objective (PSO) algorithm for the Task Scheduling and Data File Assignment Problem (TaSDAP), that considers the task execution time, data transfer times and a heterogeneous set of virtual machines. As a basis for this method, we used the algorithm Hybrid Evolutionary Algorithm to Task Scheduling and Data File Assignment Problem (HEA-TaSDAP). Additionally, a parallel version of this metheuristic (PHEA-TaSDAP) was developed in order to enable the scheduling of workflows with more than 100 tasks in reasonable computation times.

Keywords: Workflow, Scheduling, Metaheuristic, GPU.

Lista de Figuras

2.1	Modelos da definição do problema (obtidos de [32])	5
3.1	Exemplo de uma partícula representando um possível escalonamento de um $workflow$ com 3 arquivos e 4 tarefas em um $cluster$ com 3 MVs	14
3.2	Exemplo de vizinhanças <i>move-element</i> and 2-opt <i>move-element</i> para um <i>workflow</i> de 7 tarefas escalonado em um cluster com 3 MVs	20
3.3	Exemplo de vizinhanças <i>swap-element</i> and 2-opt <i>move-element</i> para um workflow de 7 tarefas escalonado em um cluster com 3 MVs	21
4.1	Comparação do makespan e tempo de programação para o Montage 1000	25
4.2	Comparação da qualidade da solução para instâncias Montage	26
4.3	Comparação da qualidade da solução para os <i>workflows</i> Montage, Cy- bershake, Epigenomics e Sipht.	26
4.4	Comparação do tempo de escalonamento para diferentes <i>workflows</i>	28
4.5	Comparação do tempo de programação para instâncias de montage	28

Lista de Tabelas

2.1	Algoritmos de escalonamento	7
2.2	Heurísticas de busca local na GPU	10
3.1	Distribuição do tempo de execução para a versão sequencial do algoritmo	16
4.1	Especificação da MV	23
4.2	Alocação de MVs para cada cluster	24
4.3	Tempos de escalonamento de GPU (em segundos) e os speedups para ins- tâncias pequenas	29
4.4	Tempos de escalonamento de GPU (em segundos) e os speedups para ins- tâncias de montage	30

Lista de Abreviaturas e Siglas

- MV : Máquina Virtual;
- WfC : *Workflow* Científico;
- GPU : Graphics Processing Unit;
- CPU : Central Processing Unit;

Sumário

1	Intr	odução		1
		1.0.1	Objetivos	2
		1.0.2	Organização	3
2	Defi	nição d	o Problema e Trabalhos Relacionados	4
	2.1	Defini	ção do Problema	4
	2.2	Traba	lhos Relacionados	6
		2.2.1	Escalonamento estático de <i>workflows</i> em <i>clouds</i>	6
		2.2.2	Métodos de Busca Local em Vizinhança Orientados para Platafor- mas CPU-GPU	9
3	Algo	oritmos	propostos para o escalonamento de WfCs	12
	3.1	PSO		12
	3.2	Identi	ficando <i>hotspots</i> da versão sequencial	15
	3.3	PHEA	-TaSDAP	16
		3.3.1	Buscas locais implementadas em GPU para o PHEA-TaSDAP $\ .$	18
			3.3.1.1 move-element	19
			3.3.1.2 2-opt <i>move-element</i>	20
			3.3.1.3 <i>swap-vm</i>	21
			3.3.1.4 4-opt <i>move-element</i>	21
		3.3.2	Buscas locais paralelas implementadas em CPU	21

4 Experimentos

5 (Conclusões e Trabalhos Futuros	31
Refe	erências	32

Capítulo 1

Introdução

Um *workflow* científico (WfC) é uma abstração usada para modelar e otimizar todas as etapas de um experimento computacional. Nos últimos anos, experimentos científicos têm aumentado em escala e complexidade, e consequentemente cada vez mais experimentos científicos [6, 15, 3, 20] estão sendo modelados como WfCs. Ao mesmo tempo, a adoção de ambientes de computação em nuvem para executar estes tipos de aplicações tem aumentado, e vários cientistas [36, 24] estão migrando seus experimentos para a nuvem.

Ambientes de computação em nuvem tem sido amplamente empregados para executar WfCs devido à facilidades de aquisição e baixo custo monetário [2, 31, 12, 11]. Ao contrário de sistemas de computação em *grid*, os serviços de computação em nuvem oferecem infra estrutura sob demanda e um ambiente de execução customizado, o que possibilita a execução de *workflows* com um grande volume de dados sem limitações de escalabilidade.

WfCs são normalmente compostos por milhares de tarefas interconectadas que são executadas de acordo com sua dependência de dados ou controle. Para executar um WfC na nuvem, suas tarefas devem ser escalonadas nas máquinas virtuais alugadas, o tempo total de execução do *workflow* (*makespan*) varia de acordo com a distribuição de tarefas nas máquinas virtuais. O número total de possíveis combinações de diferentes distribuições de tarefas nas máquinas virtuais é enorme, encontrar a solução com o menor *makespan* é um problema que pertence à classe NP-Completo [16] que requer algoritmos de otimização.

Dentre vários algoritmos para o escalonamento estático de WfCs, o *Hybrid Evolutio*nary Algorithm To Task Scheduling and Data File Assingment Problem (HEA-TaSDAP) [32] é um algoritmo que se destaca porque é o primeiro algoritmo que considera, ao mesmo tempo, ambos os problemas de escalonamento de tarefas e alocação de dados. HEA-TaSDAP é uma metaheurística evolucionária que inclui métodos de busca local e *path relinking*, e produz bons escalonamentos para *workflows* com até 100 tarefas em um tempo relativamente curto. No entanto, *workflows* deste tamanho são considerados pequenos, dado que *Workflows Cybershake*, por exemplo, podem ser modelados com até 420.000 tarefas como é visto em [35]. Apesar do HEA-TaSDAP ser capaz de escalonar *workflows* de qualquer tamanho, seu *overhead* de execução é bastante significativo e o escalonamento de instâncias com mais de 500 tarefas pode levar vários dias para terminar.

Pandey *et al.* [25] enfatizam a importância de desenvolver algoritmos de escalonamento para *workflows* que atendam os seguintes critérios: (i) baixo *overhead* de execução; (ii) capaz de fornecer soluções com recursos computacionais heterogêneos; e (iii) considera o custo de transferências de dados entre máquinas. Embora o HEA-TaSDAP atenda os critérios (ii) e (iii), seu *overhead* de execução não é ideal para escalonar instâncias do problema com mais de 100 tarefas.

1.0.1 Objetivos

Um dos objetivos é reduzir o *overhead* de execução do HEA-TaSDAP para permitir o escalonamento de *workflows* com mais de 100 tarefas em tempos computacionais razoáveis. Portanto, foi desenvolvida uma versão paralela desta metaheurística: *Parallel Hybrid Evolutionary Algorithm to Task Scheduling and Data File Assignment Problem* (PHEA-TaSDAP). O algoritmo foi paralelizado através da GPU nas plataformas de computação paralela CUDA e OpenMP. A nova implementação reduziu o *overhead* de execução em 98,83% e não houve um impacto (negativo) significativo na qualidade das soluções.

O objetivo primário deste trabalho é desenvolver um algoritmo evolucionário eficiente, baseado na metaheurística HEA-TaSDAP, para o problema de Escalonamento de Tarefas e Alocação de Arquivos (ETAA) (em inglês *Task Scheduling and Data Assingment Problem* (TaSDAP) [32]), e portanto neste trabalho foi desenvolvido o algoritmo evolucionário *Particle Swarm Optimization to Task Scheduling and Data File Assingment Problem* (PSO-TaSDAP). Esta metaheurística foi escolhida pois é tradicionalmente usada para o escalonamento de *workflows* devido ao seu curto tempo de convergência [23]. Os algoritmos propostos PHEA-TaSDAP e PSO-TaSDAP, foram avaliados quanto ao seu *overhead* de execução e qualidade da solução (*makspan*) em instâncias clássicas do problema (*Montage, Cybershake, SIPHT* e *Epigenomics*) com até 1000 tarefas. Estes algoritmos foram testados com outro algoritmo da literatura (HEFT [33]) e obtiveram em média 12% de melhoria na qualidade das soluções.

1.0.2 Organização

Este trabalho está organizado em 5 Capítulos, incluindo a introdução.

O Capítulo 2 apresenta a definição do problema e contém uma revisão dos trabalhos relacionados: tanto sobre algoritmos de otimização para o problema abordado quanto sobre metaheurísticas que foram paralelizadas através da GPU. O Capítulo 3 traz a descrição completa dos algoritmos desenvolvidos: PHEA-TaSDAP e PSO-TaSDAP. No Capítulo 4 são apresentados os resultados computacionais dos experimentos realizados. Por último, no Capítulo 5 são apresentadas as considerações finais e trabalhos futuros.

Capítulo 2

Definição do Problema e Trabalhos Relacionados

Neste capítulo são apresentados os trabalhos relacionados e a definição do problema *Task Scheduling and Data Assingment Problem* (TaSDAP) nas seções 2.2 e 2.1, respectivamente. Na subseção 2.2.1 é feita uma uma revisão da literatura sobre algoritmos de escalonamento estático desenvolvidos para ambientes de computação em nuvem. Na subseção 2.2.2 é feito um levantamento sobre estratégias eficientes de busca local para arquiteturas híbridas CPU-GPU, e são descritos métodos de busca local em vizinhança orientados para essas plataformas.

2.1 Definição do Problema

Comumente, um WfC é definido como um grafo acíclico dirigido, em que os nós são tarefas e as arestas denotam a interdependência de tarefas. De acordo com os autores Teylo *et al.* [32], essa representação é fundamental para o TaSDAP porque permite que o algoritmo de escalonamento determine não somente a alocação de tarefas mas também em que máquinas os arquivos de dados, gerados durante a execução do *workflow*, são alocados.

Neste problema, as tarefas e arquivos de dados devem ser alocados em um *cluster* de máquinas virtuais (MVs) heterogêneas, com o objetivo de minimizar o *makespan* do *workflow*. A Figura 2.1 mostra o modelo da aplicação e arquitetura. Como pode ser visto, o *workflow* é modelado como um DAG denotado por $G = (V, A, a, \omega)$, onde o conjunto de nós $V = N \cup D$ consiste de tarefas $i \in N$ e arquivos de dados $d \in D$; A é o conjunto de arcos, que dá a relação de precedência entre tarefas e arquivos de dados, a_i é o montante de trabalho associado com cada tarefa $i \in N$, e ω_{ij} representa o custo associado com o arco $(i, j) \in A$. Observe que, no grafo, uma tarefa sempre é precedida e sucedida por um arquivo, conforme ilustrado na Figura 2.1(a) em que $task_1$, lê os arquivos $data_1$ e $data_2$, e escreve $data_3$, que será lido posteriormente por $task_2$.



Figura 2.1: Modelos da definição do problema (obtidos de [32]).

O modelo da arquitetura ilustrado na Figura 2.1(b) representa as principais características do ambiente de execução. Para cada MV $j \in M$, onde M é o conjunto de todas MVs, as seguintes características são consideradas; (i) a capacidade de armazenamento (cm); (ii) o valor computacional de *slowdown* (cs); (iii) o custo de comunicação para operações de escrita (cdw); e (iv) o custo da comunicação para operações de leitura (cdr). Neste modelo o tempo de execução de uma tarefa $i \in N$ em uma MV $j \in M$ é dado por $t_{ij} = a_i \cdot cs_j$; o tempo de comunicação de uma tarefa $i \in N$ que executa na MV $j \in M$, para a escrita do dado $d \in D$ em MV $p \in M$, onde j e p são conectados por um enlace l, é dado por $\overleftarrow{t}_{djp} = \omega_{id} \cdot cdw_l$, e o tempo de comunicação para a leitura é dado por $\overrightarrow{t}_{djp} = \omega_{di} \cdot cdr_l$.

Em [32] os autores descrevem uma formulação matemática para o TaSDAP, como um problema de programação inteira chamado TaSDAP-IP. Seja $D = D_s \cup D_d$ o conjunto de todos dados, onde cada dado $d \in D$ tem um tamanho w(d) que pode ser estático (D_s) , com origem na máquina $O(d) \in M$, ou dinâmico, gerado durante a execução do workflow (D_d) . Para cada tarefa $i \in N$, é considerado um conjunto de entrada de dados $\Delta_{in}(i) \cdot D$ necessário para a execução e um conjunto de saída de dados $\Delta_{out}(i) \cdot D_d$. Além disso, T_M é definido como o tempo máximo de execução para o workflow e $T = \{1...T_M\}$ como o conjunto de intervalos de tempo de uma execução. Neste sentido, o TaSDAP é definido como um problema de escalonamento de tarefas e alocação de dados em MVs, respeitando a capacidade de armazenamento disponível e tentando minimizar o *makespan*. A definição matemática do problema está descrita em [32].

2.2 Trabalhos Relacionados

As subseções seguintes apresentam as abordagens mais comuns para o escalonamento estático de *workflows* em *clouds*, e métodos de busca local em vizinhança tendo em vista que uma das abordagens propostas, para o TaSDAP, neste trabalho é uma metaheurística paralela orientada para arquiteturas CPU-GPU. No entanto, não existem trabalhos na literatura para este problema de escalonamento que fazem uso de heurísticas GPU ou CPU-GPU. Portanto os métodos de busca local descritos na subseção 2.2.2 são referentes a problemas variados.

2.2.1 Escalonamento estático de workflows em clouds

Escalonamento de WfCs em sistemas com recursos computacionais distribuídos é considerado um problema pertencente à classe NP-difícil [34]. Diversos algoritmos foram desenvolvidos para realizar o escalonamento em recursos distribuídos. No entanto, algoritmos com baixo *overhead* de execução geralmente produzem soluções longe da otimalidade, e por outro lado técnicas que produzem soluções com boa qualidade costumam ter um *overhead* de execução enorme.

Algoritmos baseados em heurísticas e metaheurísticas tem sido amplamente aplicados para o escalonamento de *workflows* científicos em sistemas de computação em *grid* e sistemas computação em nuvem. O foco do escalonamento nestes dois ambientes é diferente, as soluções para ambientes *grid* assumem que os recursos computacionais são limitados e podem aumentar ou diminuir ao longo da execução do *workflow* portanto algoritmos para estes ambientes buscam minimizar o tempo de execução do *workflow* sem se preocupar em reduzir o custo da execução. Por outro lado, não existe a restrição da disponibilidade de recursos computacionais em ambientes de computação em nuvem, logo os algoritmos para estes ambientes são mais focados em minimizar o custo da execução ou cumprir os prazos de execução.

Segundo [37], algoritmos de escalonamento estático não são apropriados para ambientes grid porque os recursos computacionais podem ser agregados ou tomados por outra aplicação em qualquer instante durante a execução, o que torna difícil estimar de forma precisa os custos de computação e comunicação de cada tarefa. Algoritmos de escalonamento dinâmico devem ser aplicados para estes ambientes. Estes algoritmos empregam técnicas de tolerância a falha e refazem o escalonamento sempre que há algum tipo de mudança no ambiente de execução.

Nos últimos anos, foram propostos vários trabalhos que exploram a utilidade de algoritmos baseados na natureza, como algoritmos genéticos, *Particle Swarm Optimization* PSO e *Ant Colony Optimization* ACO, para gerar soluções eficientes para o problema. Entretanto, esta área de otimização ainda tem espaço para ser aprimorada. Nesta seção é feita uma revisão da literatura sobre algoritmos de otimização para o escalonamento estático de *workflows* em ambientes de computação em nuvem. Na Tabela 2.1 se encontra um resumo de algoritmos de escalonamento descritos nesta seção.

Trabalho	Estratégia	Ambiente	Alocação de dados
Teylo <i>et al.</i> , 2017 [32]	metaheurística	heterogêneo	sim
Rodriguez e Buyya, 2014 [29]	metaheurística	heterogêneo	não
Abrishami $et al.$, 2013 [2]	heurística	heterogêneo	não
Casas <i>et al.</i> , 2016 [8]	metaheurística	homogêneo	\sin
Casas <i>et al.</i> , 2017 [9]	metaheurística	homogêneo	indireta
Malaski <i>et al.</i> , 2012 [22]	heurística	homogêneo	não

Tabela 2.1: Algoritmos de escalonamento

Em Teylo *et al.*, é proposto um novo modelo de representação de *workflow*, em que tarefas e arquivos são representados como nós em um DAG. Diferente do modelo tradicional, no qual somente as tarefas são consideradas na representação do modelo, a representação proposta nesse trabalho permite que seja feita uma distribuição mais eficiente de arquivos que resulta em uma redução significativa nas transferências de dados entre máquinas virtuais, especialmente em *data-intensive workflows*. O problema abordado nesse trabalho é definido como *Task Scheduling and Data Assignment Problem* (TaSDAP), também é proposto algoritmo híbrido evolucionário para esse problema (HEA-TaSDAP), que inclui métodos de busca local e *path relinking*, e tem como objetivo reduzir o *makespan* do *workflow*. Na análise experimental foram conduzidos testes teóricos com 6 *workflows* sintéticos e um *workflow* real, nos quais o algoritmo obteve uma melhora em relação às soluções oferecidas pelo HEFT(11.15%) e MinMin-TSH(22.72%). Seu *overhead* de execução é bastante significativo, chegando a 9 minutos para instâncias com 100 tarefas.

Casas et al. [8] desenvolveram um algoritmo genético (GA-ETI) com objetivo de

minimizar o makespan e custos monetários da execução do workflow. Esta abordagem emprega operadores de mutação que adicionam/removem MVs durante a execução do algoritmo, o que resultou em soluções com custos monetários bastante reduzidos. Este algoritmo foi testado com outros três algoritmos de escalonamento, incluindo o HEFT, e obteve soluções com makespan menor do que o HEFT em 4 das 5 instâncias testadas (para mais de 13 MVs, este algoritmo empatou com o HEFT no workflow Epigenomics). O ambiente usado para os testes foi uma nuvem privada (VMware-vSphere) com 25 máquinas virtuais, e o Pegasus-WMS foi usado como middleware para executar o GA-ETI. Os tempos de transferências de dados entre MVs são considerados pelo algoritmo no cálculo do makespan.

Em Casas *et al.* [9], o *Particle Swarm Optimization with Discrete Adaptation* (PSO-DS) é proposto, este algoritmo possui um *overhead* de execução baixo de, e segundo os autores, o algoritmo converge para soluções em que tarefas com dependências são executadas na mesma MV, reduzindo assim o número de transferências entre MVs.

Em Malaski *et al.* [22] é proposto o algoritmo *Static Provisioning Static Scheduling* (SPSS) com o objetivo de minimizar o custo monetário da execução e definir prazos de execução. Foram usadas instâncias Montage, CyberShake, SIPHT, Epigenomics e LIGO com até 1000 tarefas nos testes realizados para esse algoritmo. O ambiente de execução em que o algoritmo foi estado é um *cluster* com MVs homogêneas pois o algoritmo não foi implementado para escalonar tarefas em recursos computacionais heterogêneos, e tampouco considera o tempo das transferências de dados, e armazenamento. Seu *overhead* de execução é significativo, chegando a 15 minutos para as maiores instâncias.

Abrishami et al. [2] propuseram dois algoritmos (IC-PCP e IC-PCPD2) para o escalonamento estático de *workflows*. Esta abordagem leva em consideração o caminho critico de um *workflow*, o custo monetário dos recursos computacionais oferecidos pela nuvem e maquinas virtuais heterogêneas. Ambos algoritmos propostos tem como objetivo minimizar o custo monetário de execução respeitando um prazo de execução definido e a disponibilidade de recursos. Nesse trabalho foram feitos testes sintéticos com 10 máquinas virtuais heterogêneas com características similares às oferecidas pelos serviços da Amazon EC2. Nestes testes foram usadas instâncias de *workflows* Montage, CyberShake, SIPHT, Epigenomics e LIGO, variando entre 30 e 1000 tarefas.

Rodriguez e Buyya [29] apresentaram um algoritmo PSO com o objetivo de minimizar o custo monetário da execução e ao mesmo tempo atender os prazos de execução. Este algoritmo incorpora princípios básicos de modelos de nuvem IaaS, como o modelo de pagamento pré-pago, heterogeneidade e elasticidade dos recursos, e também considera características típicas de plataformas IaaS, tais quais; variações de desempenho e tempos de inicialização das máquinas virtuais. Este algoritmo foi comparado com outros dois algoritmos, sendo um destes o IC-PCP, e foram realizados testes com diversos *workflows* sintéticos avaliando o desempenho do algoritmo em diversos aspectos; *makespan*, custo monetário de execução e porcentagem de prazos de execução cumpridos. O algoritmo PSO proposto atingiu uma porcentagem de prazos cumpridos maior e encontrou soluções com um *makespan* menor em todos testes em que os prazos de execução foram cumpridos. Esta abordagem não leva em conta os tempos de transferências de dados entre MVs no cálculo do *makespan*. Tampouco considera a capacidade de armazenamento das MVs, o que pode levar a soluções inviáveis.

2.2.2 Métodos de Busca Local em Vizinhança Orientados para Plataformas CPU-GPU

Nos últimos anos, as unidades de processamento gráfico (GPUs) tornaram-se parte integrante de sistemas de computação de alto desempenho, foram bastante difundidas na comunidade científica por apresentar uma alternativa econômica em relação às CPUs, em termos de desempenho. Arquiteturas CPU-GPU tem se tornado mais relevantes por oferecer bastante desempenho com uma boa relação de custo/beneficio.

A busca local é um componente fundamental para muitas metaheurísticas, este método heurístico vem sendo usado há bastante tempo [1]. Nesse contexto, a utilização de GPUs para acelerar procedimentos de busca local ganhou destaque no campo de otimização e metaheurísticas devido à sua capacidade de encontrar soluções quase ótimas em problemas difíceis.

Heurísticas de busca local consistem em mudar a solução inicial com incrementos sistemáticos na qualidade da solução por meio da exploração de soluções vizinhas. A vizinhança de uma solução s é denotada por N(s). As soluções nesta vizinhança são obtidas a partir de operações de movimento aplicadas à solução inicial s. Os movimentos alteram uma pequena parte da solução inicial s seguindo um critério específico. Segundo [1], encontrar operadores de movimento que conduzam a ótimos locais de alta qualidade pode ser considerado um dos desafios da busca local.

Como neste trabalho é aplicada um heurística CPU-GPU de busca local para o problema abordado, a seguir é feito um levantamento bibliográfico sobre heurísticas GPU e CPU-GPU de busca local voltadas para problemas de otimização. A Tabela 2.2 apresenta um resumo dos trabalhos descritos a seguir.

Tabela 2.2: Heurísticas de busca local na GPU								
Trabalho	Heurística	Processamento						
Coelho <i>et al.</i> , 2016 [10]	VND	híbrido CPU-GPU						
Zhou <i>et al.</i> , 2016 [38]	VNS	híbrido CPU-GPU						
Campeotto $et al.$, 2014 [7]	VNS	híbrido CPU-GPU						

Em Coelho et al. [10] é proposta uma heurística CPU-GPU, Four-Neighborhood Variable Neighborhood Search (FN-VNS) com 4 métodos de busca local em vizinhança orientados para GPU, para o Single Vehicle Routing Problem with Deliveries and Selective Pickups (SVRPDSP). No FN-VNS, a solução inicial é gerada a partir de soluções exatas de dois subproblemas do SVRPDSP (TSP e Problema da Mochila), a solução é então diversificada em quatro estruturas de vizinhanças (1-OrOpt, 2-OrOpt, Swap e 2-Opt) onde é realizada a busca local pela GPU. A GPU verifica se solução encontrada é viável e atualiza a solução atual caso encontre uma solução melhor. Uma estratégia de perturbação híbrida CPU-GPU é usada para diversificar as soluções. O movimento de perturbação é gerado na CPU e aplicado no vetor de soluções da CPU, apenas as modificações neste vetor são transferidas para a GPU. Esta implementação híbrida tem como objetivo reduzir o tamanho das transferências de dados entre CPU e GPU. O testes realizados comparam o FN-VNS com outros dois algoritmos SVRPDSP, usando 68 instâncias da literatura como benchmark. O algoritmo encontrou soluções melhores para 51 destas instâncias da e empatou em 7. As soluções tiveram um speedup de 14.49 em instâncias grandes.

Campeotto *et al.* [7] propõem um *framework* orientado para GPUs, para o *Constraint Satisfaction Problem* (CSP), que explora o paralelismo da propagação e exploração de diversas estruturas de vizinhanças de grande porte (RL, RP, 2P, GS, ICM e CE). Foi obtido um *speedup* de 38 na maior instância testada.

Uma heurística de busca local iterada para o problema do caixeiro viajante é proposta por Zhou *et al.* [38], esta heurística possui 5 métodos de busca local em vizinhanças (2opt-glm, 2-opt-tex, 2-opt-tex-shm, 2-opt-shm e 2-opt-coord). Os *kernels* correspondentes às vizinhanças 2-opt-shm e 2-opt-coord obtiveram os melhores resultados em termos de desempenho. No *kernel* 2-opt-shm, as coordenadas das cidades e o vetor de soluções são armazenados na memória local dos multiprocessadores da GPU, no entanto este *kernel* é limitado pelo tamanho da memória local e nos testes conduzidos não foi capaz de executar instâncias com mais 4096 cidades. O *kernel* 2-opt-coord faz acesso coalescente à memoria global, resultando em um melhor aproveitamento da largura de banda máxima da GPU usada nos testes. Foram testadas instâncias clássicas do problema do caixeiro viajante, o algoritmo obteve *speedups* de ate 279 para o *kernel* 2-opt-coord na maior instância testada com 4461 cidades. A qualidade das soluções fornecidas por esta heurística foi deteriorada em até 3.67%, nenhum *kernel* encontrou a solução ótima para as instâncias testadas.

Capítulo 3

Algoritmos propostos para o escalonamento de WfCs

Neste capítulo são apresentados os algoritmos de escalonamento estático de *workflows*: PSO-TaSDAP e PHEA-TaSDAP. Estes dois algoritmos são baseados em metaheurísticas evolutivas, e foram desenvolvidos especificamente para o problema TaSDAP. Na seção 3.1 é apresentado PSO-TaSDAP e é detalhada sua implementação. O PHEA-TaSDAP é apresentado na seção 3.2, são descritas as funções paralelizadas de busca local e geração de população do HEA-TaSDAP nas subseções 3.2.1 e 3.2.2.

3.1 PSO

O algoritmo de otimização por enxame de partículas PSO (em inglês, *Particle Swarm Optimization*) foi introduzido por Kennedy e Ebehart [19], e tem sido amplamente usado desde então. Esta metaheurística foi inspirada no comportamento social de um bando de aves durante a procura de alimentos e locais para criar seus ninhos. Este algoritmo evolucionário é uma técnica de otimização estocástica em que a partícula representa um individuo que tem a capacidade de se mover pelo espaço definido do problema. A representação de uma partícula *i* é dada pela sua posição x_i e velocidade v_i . Partículas armazenam os valores da sua melhor posição *pbest* e da melhor posição global *gbest*; os valores são obtidos por uma função que avalia a qualidade da solução com base na posição da partícula. A cada iteração do algoritmo são atualizados os valores da velocidade e posição das partículas, estes valores são alterados de acordo com as variáveis de aceleração aleatórias $rand_1, rand_2$ na direção das posições *pbest* e *gbest*. A posição e velocidade das partícula são atualizadas de acordo nas equações 3.1 e 3.2, respectivamente.

$$v_i^{k+1} = \omega v_i^k + c_1 rand_1 \cdot (pbest_i - x_i^k) + c_2 rand_2 \cdot (gbest - x_i^k), \qquad (3.1)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{3.2}$$

 ω = inércia c_1, c_2 = coeficientes de aceleração $rand_1, rand_2$ = números aleatórios em que $rand_i \in [0, 1]$ x_i = posição atual de uma partícula i

O parâmetro de inércia ω tem um impacto significativo na convergência do algoritmo, este parâmetro determina o quanto as velocidades anteriores vão impactar na velocidade atual e possui uma relação inversa entre o componente cognitivo (aprendizado local) e o componente social (aprendizado global) das partículas. Por um lado, um valor de inércia alto vai aumentar a velocidade das soluções e portanto vai favorecer a exploração global. Por outro lado, um valor menor causaria a desaceleração e como consequência favoreceria exploração local. Desta forma, um valor ω que equilibra buscas locais e globais resulta em poucas iterações necessárias para que o algoritmo convirja.

Em contra partida, os parâmetros c_1 e c_2 não têm um efeito significativo na convergência do algoritmo. No entanto, ajustar estes valores adequadamente pode resultar em uma convergência mais rápida e prevenir que o algoritmo fique preso em um local mínima de baixa qualidade. O parâmetro c_1 denota o componente cognitivo e o valor c_1rand_1 na 3.1 define, de forma aleatória, o peso da melhor posição anterior (*pbest*). O componente social é denotado por c_2 , onde c_2rand_2 define o comportamento da partícula relativa às outras partículas vizinhas.

Outros três parâmetros que não estão representados nas equações 3.1 e 3.2 afetam o desempenho do algoritmo: o número de partículas, a dimensão da partícula e a distância máxima que as partículas podem se mover a cada iteração. Geralmente, quanto maior for o número de partículas, maior é a probabilidade de encontrar a solução global ótima. A quantidade ideal de partículas depende da complexidade do problema de otimização mas tipicamente varia entre 20 e 100 partículas [5, 30]. Em testes conduzidos de forma empírica com 20 - 150 partículas, foi observado que para uma quantidade pequena de partículas, ocorre uma convergência prematura do algoritmo. Constatamos que uma população de 150 partículas torna o algoritmo mais robusto, e portanto este número foi escolhido. A quantidade de dimensões das partículas depende do problema de otimização a ser resolvido e como é modelado, no problema de escalonamento abordado neste trabalho a quantidade de dimensões é igual à soma de arquivos de dados e tarefas, variando entre 193

e 1843 dimensões para a menor e maior instância testada (Montage100 e Montage1000) respectivamente. Cada dimensão de uma partícula x_i é denotada x_{jk} , onde j é um arquivo ou tarefa alocado em uma máquina virtual k como ilustrado na Figura 3.1.

O valor máximo (V_{max}) para as velocidades define a alteração máxima que uma partícula pode ter a cada iteração, este valor normalmente é a metade do valor da posição máxima que a partícula pode ocupar.



Figura 3.1: Exemplo de uma partícula representando um possível escalonamento de um *workflow* com 3 arquivos e 4 tarefas em um *cluster* com 3 MVs.

As partículas do enxame são inicializadas com as posições correspondentes às soluções geradas anteriormente pelo HEFT, e as velocidades são escolhidas de forma aleatória. Em seguida, cada partícula é avaliada por uma função *fitness* que determina a qualidade da solução *makespan* de cada partícula *i*, e compara o resultado com o melhor resultado obtido por esta partícula, denominada $pbest_i$. Se a partícula tiver qualidade superior a $pbest_i$, a posição $pbest_i$ é atualizada com a posição atual da partícula x_i . A partícula *gbest* é atualizada com os parâmetros de x_i caso tenha um *makespan* maior que x_i .

A seguir, a velocidade das partículas é calculada de acordo com a equação 3.1 e suas posições são determinadas seguindo a equação 3.2. Neste problema, a posição das partículas é representada pela ordem de execução de tarefas, e as máquinas virtuais em que estão distribuídas. Durante a execução do programa o enxame de partículas circula em torno das melhores posições anteriores $pbest_i$ e gbest, e à medida que estas variáveis são atualizadas, a trajetória da partícula muda para outras regiões mais promissoras do espaço de busca. Após 20.000 iterações o algoritmo retorna a posição da partícula com menor makespan (gbest).

O Algoritmo 1 representa o algoritmo geral do PSO-TasDAP. Nesta metaheurística, a população inicial com 150 indivíduos é gerada pela heurística HEFT [33].

Algoritmo 1 PSO-TaSDAP

Entrada: Workflow, |M|: conjunto de máquinas virtuais.

```
Saída: Melhor solução encontrada(gbest).
```

```
1: P \leftarrow \text{HEFT}()
 2: E \leftarrow inicializaEnxame(P)
 3: iter \leftarrow 0
 4: Enquanto iter \leq MAX Faça
          Para \forall partícula \ i \in E Faça
 5:
               Se fitness(x_i^k) < fitness(pbest_i) Então
 6:
                    pbest_i \leftarrow x_i^k
 7:
               Se fitness(x_i^k) < fitness(qbest) Então
 8:
                    qbest \leftarrow x_i^k
 9:
               Para \forall n \circ j \in N Faça
10:
                    v_{ij}^{k+1} = \omega \cdot v_{ij}^k + c_1 rand_1 \times (pbest_{ij} - x_{ij}^k) + c_2 rand_2 \times (gbest - x_{ij}^k)
11:
                   x_{ij}^{k+1} = x_{ij}^k + v_{ij}^{k+1}
12:
13: retorna gbest
```

3.2 Identificando *hotspots* da versão sequencial

O primeiro passo na redução do *overhead* de execução do algoritmo sequencial HEA-TaSDAP é identificar os *hotspots* do programa. Os *hotspots* são funções e rotinas que consomem a maior parte do tempo computacional, e portanto são os principais alvos para otimização. Para identificar os *hotspots* foram usados *timers* no código, que foi em três partes: *Local Search*, *Crossover & Mutation* e *Other Functions*. Os testes foram conduzidos com 16 WfCs variando entre 100 e 1000 tarefas.

A Tabela 3.1 mostra os tempos de execução (em segundos) para cada parte do código e a sua respectiva porcentagem em relação ao tempo total de execução do HEA-TaSDAP. Os resultados são exibidos para diferentes WfCs escalonados em *clusters* de 5 MVs. Também foram conduzidos testes com *clusters* de diferentes tamanhos e os resultados foram bastante similares.

Considerando que cerca de 99% do tempo de escalonamento está concentrado nas funções de busca local, se espera que a paralelização desta parte oferece mais potencial na redução do *overhead* de execução.

Workflow ($\#$ of tasks)	Local search	Crossover & Mutation	Other functions
Montage (100)	929s~(97.58%)	10s (1.05%)	13s (1.37%)
Montage (200)	7680s~(98.12%)	15s~(0.19%)	132s (1.69%)
Montage (300)	$15054s \ (96.85\%)$	16s (0.1%)	473s (3.04%)
Montage (400)	16980s (98.02%)	11s~(0.06%)	332s~(1.92%)
Montage (500)	40180s (97.38%)	17s~(0.04%)	1062s~(2.57%)
Cybershake (100)	247s (97.24%)	3s (1.18%)	4s (1.57%)
Cybershake (200)	986s~(98.21%)	4s (0.4%)	14s (1.39%)
Cybershake (300)	4252s~(98.75%)	5s~(0.12%)	49s~(1.14%)
Cybershake (400)	10627s~(99.74%)	7s~(0.07%)	21s~(0.2%)
Cybershake (500)	124013s (99.68%)	10s~(0.04%)	68s~(0.28%)
Sipht (100)	1802s~(98.52%)	3s~(1.91%)	23s~(1.26%)
Sipht (300)	13491s (98.86%)	10s~(1.09%)	133s~(0.97%)
Sipht (400)	60537s (98.41%)	12s~(0.58%)	947s~(1.54%)
Sipht (500)	43491s (97.26%)	15s~(0.63%)	1188s (2.66%)
Epigemonics (100)	650s~(92.46%)	1s (5.88%)	42s~(5.97%)

Tabela 3.1: Distribuição do tempo de execução para a versão sequencial do algoritmo.

3.3 PHEA-TaSDAP

Algoritmos genéticos são métodos de busca e otimização inspirados na evolução biológica, em que é feita uma analogia entre indivíduos, ou cromossomos, e soluções viáveis para um problema. A qualidade da solução é dada pela função *fitness* que computa o valor *fitness* de cada indivíduo. O algoritmo inicia com uma população de indivíduos e as soluções correntes são modificadas ao longo das iterações por meio de operadores genéticos como mutação e *crossover*. É uma técnica robusta bastante usada [13, 27, 18, 17, 26] para resolver problemas complexos em que o espaço de soluções é muito abrangente.

Em [32], foi desenvolvido o HEA-TaSDAP, um algoritmo genético que inclui um método de *path relinking* e procedimentos de busca local. A representação de um cromossomo no algoritmo é composta de duas estruturas. A primeira estrutura é um vetor que representa a alocação tarefas e arquivos de dados denominado, AV. Em AV, cada índice *i* representa uma tarefa ou um arquivo, e cada elemento AV[i] contém a máquina virtual onde a tarefa ou arquivo foi atribuído. A segunda estrutura é uma lista encadeada, denotada por OV, que representa a ordem de execução das tarefas.

A complexidade espacial dessas duas estruturas é de $O(|V|) \in O(|N|)$, respectivamente. Buscas locais aplicadas a estas estruturas costumam lidar com um vasto espaço de busca, dado que WfCs podem ser modelados com centenas de milhares de tarefas e arquivos.

Três procedimentos de busca local foram usados no algoritmo sequencial (HEA-TaSDAP): (i) *swap-vm*: troca dois elementos do *AV*; (ii) *swap-position*: troca dois elementos com as mesmas ordens de precedência na lista de execução de tarefas; e (iii) *move-element*: move uma tarefa ou arquivo para uma máquina diferente. Em todos os procedimentos de busca local, o critério de *First-Improvement* foi adotado como condição de parada, ou seja, o procedimento de busca local é executado até que se obtenha uma melhoria ou até que todas as combinações tenham sido testadas.

A versão paralela do HEA-TaSDAP (PHEA-TaSDAP) é apresentada no Algoritmo 2. Após a geração da população inicial, é executada a busca local com probabilidade de 30% a cada iteração. A principal diferença deste algoritmo para o original é o procedimento de busca local, que neste algoritmo inclui quatro estruturas vizinhança distintas (implementadas em GPU) usadas para gerar os movimentos necessários para aplicar a estratégia de *Multi Improvement*. Outros procedimentos como a geração de população (feita a cada iteração) e seleção por torneio foram paralelizados em CPU.

Algoritmo 2 PHEA-TaSDAP

Entrada: Workflow, |M|: conjunto de máquinas virtuais.

Saída: Melhor solução encontrada(best_global).

```
1: P \leftarrow populaçãoInicial()
```

- 2: best global \leftarrow encontraBest(P)
- 3: *ConjElite* $\leftarrow \emptyset$
- 4: $i \leftarrow 0$

5: Enquanto $i \leq MAX$ Faça

```
Se fazBuscasLocais? Então
 6:
           Para \forall solução \in P'//P' \subset P Faça
 7:
               move A \leftarrow buscaLocalSwapVM(solução)
 8:
               move B \leftarrow buscaLocal1optMoveElement(solução)
 9:
               move C \leftarrow \mathbf{buscaLocal2optMoveElement}(solução)
10:
               move D \leftarrow buscaLocal4optMoveElement(solução)
11:
               solução \leftarrow multiImprovement(move A, move B, move C, move D)
12:
13:
        best atual \leftarrow encontraBest(P)
       Se fitness(best_atual) melhor que fitness(best_global) Então
14:
           best global \leftarrow best atual
15:
           Se ConjElite \neq \emptyset Então
16:
               best global \leftarrow pathRelinking(best global, conjElite)
17:
           Se \forallsolução \in ConjElite, distância(best global, solução) > \alpha Então
18:
               ConjElite \leftarrow ConjElite \cup best \ global
19:
               Se |conjElite| > \beta Então
20:
                   removeCromossomo(ConjElite)
21:
       P \leftarrow \text{geraPopulaçãoEmParalelo}(P)
22:
       i = i + 1
23:
24: retorna best global
```

3.3.1 Buscas locais implementadas em GPU para o PHEA-TaSDAP

As GPUs estão cada vez mais populares nos dias de hoje devido ao seu maior desempenho e menor consumo de energia em comparação com CPUs *multicore* tradicionais. As GPUs suportam uma enorme quantidade de paralelismo a um custo relativamente baixo. No caso das GPUs NVIDIA e do modelo de programação CUDA, *kernels* são transferidos para a GPU, onde o código do *kernel* é executado por múltiplas *threads* executando em diferentes núcleos da GPU.

Como discutido anteriormente, os procedimentos atuais de busca local produzem um overhead de escalonamento significativo, tornando o HEA-TaSDAP muito ineficiente ao escalonar WfCs com mais de 100 tarefas. A fim de aumentar a eficiência metaheurística, permitindo que o escalonador lide com instâncias maiores, propomos o algoritmo Paralelo HEA-TaSDAP (PHEA-TaSDAP). A ideia principal do nosso algoritmo é implementar os procedimentos de busca local na GPU. Em PHEA-TaSDAP, propomos diferentes procedimentos de busca local para tirar máximo proveito do paralelismo de alta granularidade oferecido pela GPU.

O PHEA-TaSDAP implementa em GPU as buscas locais *move-element* e *swap-vm*, e também inclui 2-opt *move-element* e 4-opt *move-element*.

Os vizinhanças aceleradas pela GPU foram implementadas com dois *kernels*. O primeiro *kernel* é responsável por avaliar todas as possíveis operações de movimento em AV. O segundo *kernel* realiza uma redução na memória compartilhada para encontrar o movimento que resultou na melhor melhoria da solução. Uma vez que todas as vizinhanças tenham terminado sua execução, os melhores movimentos de cada vizinhança são combinados em uma única solução, seguindo a estratégia *Partial-GPU-CPU Multi Improvement* [28], na qual a combinação das soluções é executada na CPU. Esta estratégia é representada pelo Algoritmo 3.

```
Algoritmo 3 Multi ImprovementEntrada: multi_improvement (Cromossomo).Saída: Cromossomo (s).s^* \leftarrow s2: Para cada combinação diferente de movs FaçaSe possível_combinar(movs) Então4: s^{**} \leftarrow combina(movs)Se (custo(s^{**}) < custo(s^*)) Então6: s^* \leftarrow s^{**}Se (custo(s^*) < custo(s)) Então8: s \leftarrow s^*retorna s
```

$3.3.1.1 \quad move-element$

Nesta busca local, um novo cromossomo é gerado ao alterar uma posição no AV, movendo uma tarefa ou arquivo para uma máquina virtual diferente. No *kernel* de avaliação, uma *thread* é atribuída para cada posição do AV e é responsável por avaliar qual é a melhor máquina virtual para aquela posição. Para um *workflow* de n tarefas e um conjunto de |M| MVs, o primeiro *kernel* executa um total de $|M| \cdot n$ avaliações, onde n avaliações são realizadas em paralelo.

A Figura 3.2(a) mostra um exemplo da vizinhança *move-element* com um *workflow* de 7 tarefas e 3 MVs. Nesse caso, 7 *threads* farão o cálculo em paralelo do custo para mover um elemento no AV de uma MV para outra.

3.3.1.2 2-opt move-element

Semelhante à vizinhança move-element, em 2-opt move-element, um novo cromossomo é obtido movendo duas tarefas ou arquivos para máquinas virtuais diferentes no AV, como mostra a Figura 3.2(b), que ilustra apenas 3 dos 21 movimentos possíveis. O número total de movimentos é dado por $C_{n,2} = \frac{n!}{2!(n-2)!} = \frac{n \cdot (n-1)}{2}$, sendo este o número total de combinações dos pares de diferentes movimentos em um workflow de n tarefas e arquivos de dados, o primeiro kernel é disparado com o mesmo número threads. Assim, cada par de posições distintas no AV é atribuído a uma thread que realiza $|M|^2$ avaliações. Para garantir que não haja repetição nos movimentos, uma função de mapeamento eficiente, descrita em [21], é usada, onde as posições x, y são calculadas em função do índice i de cada thread, e do número total de tarefas e arquivos de dados n:



Figura 3.2: Exemplo de vizinhanças *move-element* and 2-opt *move-element* para um *workflow* de 7 tarefas escalonado em um cluster com 3 MVs.

3.3.1.3 *swap-vm*

A busca local *swap-vm* troca a posição de dois elementos no AV. Analogamente à vizinhança 2-opt *move-element*, existem $C_{n,2}$ possíveis trocas de pares de elementos no AV, mas executar a troca exige apenas uma operação por *thread*, independente do número de VMs. Portanto o *kernel* de avaliação é disparado com $C_{n,2}$ *threads*. A Figura 3.3(a) ilustra um exemplo de 3 possíveis movimentos.

3.3.1.4 4-opt move-element

Na vizinhança 4-opt move-element quatro tarefas ou arquivos são movidos para máquinas virtuais diferentes, como é mostrado na Figura 3.3(b) que ilustra um pequeno subconjunto de 3 movimentos. Como o número total de movimentos para esta vizinhança é enorme $(C_{n,4})$, foi necessário estabelecer um número arbitrário de movimentos, armazenados em uma estrutura de dados gerada na CPU e copiada para a GPU. Estes movimentos são armazenados em ordem lexicográfica, para garantir que não haja repetição nos movimentos. Uma vez que esta estrutura é copiada para a GPU, o kernel de avaliação calcula a melhor alocação para estes 4 elementos, realizando um total de $|M|^4$ por thread.



Figura 3.3: Exemplo de vizinhanças *swap-element* and 2-opt *move-element* para um work-flow de 7 tarefas escalonado em um cluster com 3 MVs.

3.3.2 Buscas locais paralelas implementadas em CPU

Em testes realizados preliminarmente, foi observado que o algoritmo PHEA-TaSDAP converge para uma solução melhor em menos iterações quando são aplicadas as buscas locais sequenciais da estratégia *First Improvement* nas primeiras gerações do algoritmo. Portanto, os três procedimentos de busca local presentes no algoritmo sequencial HEA-TaSDAP também foram paralelizados em CPU (usando a biblioteca OpenMP). Estes procedimentos são aplicados à melhor solução quando; (i) há melhora no cromossomo *best_global* e (ii) o número total de iterações (gerações) for menor que 90. A partir da 90-ésima iteração, passam a ser usadas as buscas locais implementadas em GPU.

O pseudocódigo de um dos procedimentos de busca local (swap-vm) é estruturado pelo Algoritmo 4.

Algoritmo 4 swap-vm_paralelo
Entrada: swap-vm_paralelo (Cromossomo).
Saída: Cromosomo (s).
$s^* \leftarrow s$
$size \leftarrow num_tasks + num_files$
3: $offset \leftarrow 0$
$i \leftarrow 0$
$n \leftarrow n_threads$
6: $found_improvement \leftarrow false$
$improvement \leftarrow 0$
$improvement_thread \leftarrow -1$
9: Enquanto $(i < size - 2)$ Faça
$\mathbf{Enquanto} \ (\textit{offset} < size - i) \ \mathbf{Faça}$
${f Se} \ n/2 <= (size - offset - i - 1) < n \ {f Ent ilde a}$
12: $n \leftarrow n/2$
Para cada thread Faça
Se s*.allocation[$i + 1 + thread_id + offset$] ! = s*.allocation[i] Então
15: $s^* \leftarrow \operatorname{swap}(i, (i+1+thread_id+offset))$
$\mathbf{Se} \ (\mathtt{custo}(s^*) < \mathtt{custo}(s)) \ \mathbf{Ent}\mathbf{\tilde{ao}}$
Se $(custo(s^*) - custo(s) > improvement)$ Então
18: $found_improvement \leftarrow true$
$improvement \leftarrow \texttt{atomic_write}(\texttt{custo}(s^*)) - \texttt{custo}(s)$
$improvement_thread \leftarrow thread_id$
21: Se (found improvement) Então
$s^{**} \leftarrow s$
$s^{**} \leftarrow swap(i, (i+1+improvement \ thread + offset))$
24: retorna s^{**}
$offset \leftarrow offset + n$
found improvement \leftarrow false
27: $improvement \leftarrow 0$
$improvement_thread \leftarrow -1$
$i \leftarrow i + 1$
30: $offset \leftarrow 0$
$n \leftarrow n$ threads
retorna s

Capítulo 4

Experimentos

Neste capítulo, avaliamos as soluções produzidas pelos algoritmos PHEA-TaSDAP e PSO-TaSDAP quanto ao tempo de escalonamento e qualidade da solução. Estes algoritmos foram comparados com a versão sequencial (HEA-TaSDAP) proposta em [32], e com uma heurística de escalonamento usada pelo sistema Pegasus [14], chamada *Heterogenous Earliest Finish Time* (HEFT) [33].

HEFT é um dos quatro algoritmos de escalonamento suportados pelo módulo *Pega*sus Site Selector. Os outros três algoritmos (*Random*, Group e RoundRobin) não foram incluídos nos experimentos porque não obtiveram soluções com qualidade superior ao HEFT.

O teste experimental compreende um conjunto heterogêneo de máquinas virtuais sintéticas baseadas no Amazon EC2 detalhadas nas Tabelas 4.1 e 4.2, e um gerador de *workflow* sintético proposto em [4] para gerar instâncias com 100 a 1000 tarefas.

The second se									
Tipo da VM	Slowdown	Armazenamento	Bandwidth						
m3.medium	1.53	8 GB	4 Mbps						
m3.large	0.77	32 GB	$9 { m ~Mbps}$						
m3.xlarge	0.38	$80~\mathrm{GB}$	$10 { m ~Mbps}$						
m3.2xlarge	0.19	$160~\mathrm{GB}$	$10 { m Mbps}$						

Tabela 4.1: Especificação da MV.

#de MVs	Distribuição de MVs
2	2 x m3.2xlarge
3	$3 \ge m3.2 x large$
4	$3 \ge m3.2x$ large, $1 \ge m3.x$ large
5	$3 \ge m3.2x$ large, $2 \ge m3.x$ large
6	$3 \ge m3.2x$ large, $3 \ge m3.x$ large
7	$3 \ge m3.2 x large, 3 \ge m3.x large, 1 \ge m3.large$
8	$3 \ge m3.2x$ large, $3 \ge m3.x$ large, $2 \ge m3.large$
9	3 x m3.2xlarge, 3 x m3.xlarge, 2 x m3.large, 1 x m3.medium
10	3 x m3.2xlarge, 3 x m3.xlarge, 2 x m3.large, 2 x m3.medium
11	3 x m3.2xlarge, 3 x m3.xlarge, 3 x m3.large, 2 x m3.medium
12	3 x m3.2xlarge, 3 x m3.xlarge, 3 x m3.large, 3 x m3.medium
13	4 x m3.2xlarge, 3 x m3.xlarge, 3 x m3.large, 3 x m3.medium
14	4 x m3.2xlarge, 4 x m3.xlarge, 3 x m3.large, 3 x m3.medium
15	4 x m3.2xlarge, 4 x m3. xlarge, 4 x m3.large, 3 x m3.medium
16	4 x m3.2xlarge, 4 x m3. xlarge, 4 x m3.large, 4 x m3.medium
17	5 x m3.2xlarge, 4 x m3.xlarge, 4 x m3.large, 4 x m3.medium
18	5 x m3.2xlarge, 5 x m3.xlarge, 4 x m3.large, 4 x m3.medium
19	5 x m3.2xlarge, 5 x m3.xlarge, 5 x m3.large, 4 x m3.medium
20	5 x m3.2xlarge, 5 x m3.xlarge, 5 x m3.large, 5 x m3.medium

Tabela 4.2: Alocação de MVs para cada cluster.

Para as instâncias de referência, selecionamos quatro aplicações de WfCs: o Montage, uma aplicação de astronomia usada para gerar mosaicos customizados do céu; Cybershake, uma aplicação usada para caracterizar o risco de terremotos; Epigenomics, uma aplicação de biologia projetada para executar operações de sequenciamento do genoma; SIPHT, uma aplicação de biotecnologia criada para procurar pequenos RNAs não traduzidos. Montage, Cybershake e Epigenomics são WfCs que fazem uso intensivo de dados e SIPHT é um *workflow* com uso intensivo de processamento [4].

A versão paralela do algoritmo foi implementada em C ++, OpenMP e CUDA versão 9.1.85. Todos os experimentos (paralelos e sequenciais) foram executados em um computador com processador AMD Ryzen R7 1800X de 3.9 GHz, com 32 GB de memória e GPU NVIDIA GeForce RTX 2080 Ti (arquitetura Turing), e sistema operacional Ubuntu 18.04.

Primeiro analisamos a qualidade do escalonamento quanto ao *makespan* (qualidade) da solução. Considerando que a maioria dos WfCs com aplicações reais são modelados com milhares de tarefas, avaliamos os algoritmos propostos PHEA-TaSDAP e PSO-TaSDAP com uma instância de 1000 tarefas (Montage 1000). Os resultados do *makespan* e de escalonamento são mostrados na Figura 4.1. Em comparação com os resultados do algoritmo HEA-TaSDAP e PSO-TaSDAP, o PHEA-TaSDAP apresentou *makespan* menores em todos os casos.



Figura 4.1: Comparação do makespan e tempo de programação para o Montage 1000.

Quatro conjuntos de instâncias com um grande número de tarefas (entre 200 a 500) foram analisadas. A Figura 4.2 apresenta os resultados do makespan em minutos para instâncias grandes do Montage executando em até 20 MVs. Podemos observar nos gráficos que as versões CPU e GPU geraram consistentemente melhores resultados de makespan do que o PSO-TaSDAP. Comparando os resultados das versões sequencial e paralela, observamos resultados similares no makespan para mais de 4 MVs.

O PHEA-TaSDAP e PSO-TaSDAP também conseguem fornecer soluções boas com makespan baixo para WfCs menores. A Figura 4.3 mostra os resultados do makespan em minutos para instâncias pequenas que executam em *clusters* de até 20 MVs. Podemos observar nos gráficos desta figura, que para *data-driven* WfCs como Montage e SIPHT, tanto o algoritmo sequencial HEA-TaSDAP (CPU) quanto a versão paralela, geraram melhores resultados de *makespan* do que o PSO-TaSDAP. Para Cybershake e Epigenomics, os resultados produzidos pelo algoritmo PSO-TaSDAP ficaram mais próximos do PHEA-TaSDAP e HEA-TaSDAP. Comparando as soluções geradas pelas versões CPU(HEA-TaSDAP) e GPU(PHEA-TaSDAP), podemos observar que elas são semelhantes, exceto nos casos em que um pequeno número de MVs é usado.

Em seguida, apresentamos uma análise do *overhead* de escalonamento e o *speedup* obtido pela implementação paralela. O HEFT não foi incluído nesta análise porque obteve soluções de baixa qualidade em relação aos outros algoritmos. Quando o problema







Figura 4.3: Comparação da qualidade da solução para os *workflows* Montage, Cybershake, Epigenomics e Sipht.

aumenta em complexidade e tamanho, escalonar de forma eficiente um grande número de tarefas e arquivos se torna mais difícil.Os algoritmos HEA-TaSDAP, PHEA-TaSDAP e

PSO-TaSDAP exploram o espaço de soluções de forma mais inteligente e portanto consomem mais tempo do que algoritmos simples como o HEFT.

A figura 4.4 mostra o tempo de escalonamento em segundos dos algoritmos propostos e do HEA-TaSDAP, para instâncias pequenas executadas em *clusters* de até 20 MVs. Os gráficos mostram que o tempo de escalonamento do PHEA-TaSDAP e PSO-TaSDAP aumentam proporcionalmente ao número de MVs, enquanto o tempo de escalonamento do HEA-TaSDAP apresenta um comportamento irregular. No *workflow* Epigenomics, por exemplo, os tempos de escalonamento do HEA-TaSDAP são surpreendentemente baixos com 9 MVs, se aproximando dos tempos da versão paralela e PSO-TaSDAP.

Isto ocorre porque existe um *overhead* adicional em todos os procedimentos de busca local implementados na GPU, que usam a estratégia de *Multi Improvement*, testando todas as combinações possíveis de movimentos gerados pelas diferentes vizinhanças exploradas, enquanto a busca local da CPU usa a estratégia *First Improvement*, em que os tempos de execução tendem a variar muito mais devido à natureza desta estratégia; e.g. um método de busca local termina sua execução após explorar o primeiro movimento que resulta em uma melhora na solução, enquanto que outra estratégia (*Multi Improvement*) explora e mescla todos possíveis movimentos antes de retornar uma solução. Se observa que esta diferença nos critérios de parada das buscas locais tem um impacto na velocidade de convergência do algoritmo, sendo o PHEA-TaSDAP o algoritmo que converge mais rápido.

A figura 4.5 apresenta os resultados do tempo de escalonamento para a instância Montage variando entre 200 a 500 tarefas. Resultados semelhantes foram obtidos para instâncias maiores, mas a diferença no tempo de escalonamento da versão sequencial para a paralela é significativamente maior. Para a instância Montage com 500 tarefas, a diferença é tão grande que é difícil ver o tempo de escalonamento da versão paralela no gráfico. Para ambos conjuntos de instâncias, o PSO-TaSDAP obteve em média um tempo de execução menor do que o PHEA-TaSDAP.

A fim de melhor divulgar a diferença nos tempos computacionais do escalonamento das versões CPU e GPU, as Tabelas 4.3 e 4.4 mostram os tempos de execução e o *speedup* da versão da GPU comparada à versão da CPU para as instâncias pequenas e grandes, respectivamente. A tabela 4.3 mostra que os *clusters* com mais MVs obtiveram *speedups* menores. Isso acontece porque a vizinhança 4-opt *m*ove-element foi implementada apenas na GPU, e seu tempo de execução é fortemente influenciado pelo número de MVs, como discutido na seção anterior. Os ganhos de desempenho do Montage dependem do tamanho



Figura 4.4: Comparação do tempo de escalonamento para diferentes workflows.



Figura 4.5: Comparação do tempo de programação para instâncias de montage.

da instância, observamos. Essa diferença ocorre porque instâncias pequenas não tiram todo proveito das capacidades da GPU. Como a versão da CPU pode levar dias, ou mesmo semanas, para concluir o escalonamento, os *speedups* da GPU para a instância Montage

1000 na Tabela 4.4 consideram um tempo de escalonamento de 12 horas para a versão CPU. Para este teste o tempo do PSO-TaSDAP não foi limitado pois seu *overhead* de execução é pequeno.

# of	Montage 100		Cybers	shake 100	Epigeno	omics 100	Sipht 100	
MVs	Sched	Speedup	Sched	Speedup	Sched	Speedup	Sched	Speedup
	time	Speedup	time	Speedup	time	Speedup	time	Speedup
2	10s	52.2	14s	39.1	7s	43.3	19s	89.8
3	14s	52.2	12s	33.5	16s	36.0	51s	44.6
4	25s	21.4	19s	14.1	29s	24.9	64s	28.0
5	28s	17.9	20s	12.3	36s	27.3	96s	23.4
6	25s	24.8	31s	9.2	49s	19.5	87s	37.8
7	34s	17.0	40s	8.9	68s	12.1	128s	26.7
8	51s	7.9	46s	8.2	84s	10.4	166s	22.9
9	72s	6.5	92s	2.0	90s	1.2	139s	11.9
10	74s	6.7	62s	4.1	116s	4.4	213s	20.6
11	105s	11.6	76s	16.7	138s	11.9	250s	27.3
12	93s	10.1	73s	42.7	179s	6.6	351s	25.4
13	139s	7.8	106s	11.5	15292s	10.0	251s	34.1
14	141s	9.4	138s	8.9	218s	5.6	281s	25.7
15	181s	8.9	107s	11.1	186s	8.0	332s	18.9
16	157s	7.5	92s	11.3	297s	5.4	248s	35.8
17	209s	6.3	124s	11.8	318s	3.4	314s	20.7
18	150s	6.5	198s	5.9	297s	6.8	431s	18.5
19	280s	3.7	261s	5.9	253s	9.8	417s	13.9
20	253s	5.3	221s	8.8	388s	5.8	514s	18.4

Tabela 4.3: Tempos de escalonamento de GPU (em segundos) e os speedups para instâncias pequenas.

Tabela 4.4: Tempos de escalonamento de GPU (em segundos) e os speedups para instâncias de montage.

# of	Montage 200		Mont	age 300	Montage 400		Montage 500		Montage 1000	
MVs	Sched	Speedup	Sched	Speedup	Sched	Speedup	Sched	Speedup	Sched	Speedup
	Time	Speedup	Time	Speedup	Time	Speedup	Time	Speedup	Time	Speedup
2	10s	52.2	14s	39.1	7s	43.3	19s	89.8	47s	1074.8
3	14s	52.2	12s	33.5	16s	36.0	51s	44.6	77s	396.2
4	25s	21.4	19s	14.1	29s	24.9	64s	28.0	122s	759.5
5	28s	17.9	20s	12.3	36s	27.3	96s	23.4	179s	465.1
6	25s	24.8	31s	9.2	49s	19.5	87s	37.8	119s	347.9
7	34s	17.0	40s	8.9	68s	12.1	128s	26.7	184s	269.6
8	51s	7.9	46s	8.2	84s	10.4	166s	22.9	340s	345.1
9	72s	6.5	92s	2.0	90s	1.2	139s	11.9	469s	127.8
10	74s	6.7	62s	4.1	116s	4.4	213s	20.6	357s	120.4
11	105s	11.6	76s	16.7	138s	11.9	250s	27.3	530s	85.7
12	93s	10.1	73s	42.7	179s	6.6	351s	25.4	625s	71.3
13	139s	7.8	106s	11.5	192s	10.0	251s	34.1	1216s	35.6
14	141s	9.4	138s	8.9	218s	5.6	281s	25.7	809s	56.7
15	181s	8.9	107s	11.1	186s	8.0	332s	18.9	1267s	35.7
16	157s	7.5	92s	11.3	297s	5.4	248s	35.8	1255s	34.5
17	209s	6.3	124s	11.8	318s	3.4	314s	20.7	2172s	20.8
18	150s	6.5	198s	5.9	297s	6.8	431s	18.5	1868s	24.0
19	280s	3.7	261s	5.9	253s	9.8	417s	13.9	1941s	22.7
20	253s	5.3	221s	8.8	388s	5.8	514s	18.4	1818s	24.3

Capítulo 5

Conclusões e Trabalhos Futuros

Neste trabalho, foi abordado o problema de escalonamento de tarefas e alocação de dados. Por ser um problema NP-Completo propusemos uma metaheurística PSO para este problema. Apesar deste algoritmo não ter sido capaz de encontrar soluções melhores do que o PHEA-TaSDAP, é um algoritmo bastante eficiente em encontrar resultados melhores do que o HEFT e não necessita de uma GPU para executar.

Além de uma implementação paralela para o HEA-TaSDAP (PHEA-TaSDAP) para melhorar o tempo de decisão do escalonamento. Os resultados computacionais demonstraram que a implementação da GPU resultou em resultados satisfatórios, reduzindo o tempo computacional em duas ordens de grandeza, e preservando a qualidade da solução. Com a nova implementação, agora é possível escalonar WfCs maiores que levariam dias com a implementação anterior na CPU. Quanto ao PSO-TaSDAP, este algoritmo teve um *overhead* de escalonamento relativamente baixo, sendo inclusive menor do que o do PHEA-TaSDAP para a maioria das instâncias de 200 a 500 tarefas. No entanto para 1000 tarefas, o PHEA-TaSDAP obteve soluções melhores em um menor tempo.

Como trabalho futuro, é interessante testar instâncias com mais tarefas e tornar o algoritmo dinâmico, incorporando mecanismos de tolerância a falhas, e flexibilizar a elasticidade dos recursos.

Referências

- AARTS, E.; LENSTRA, J. K., Eds. Local Search in Combinatorial Optimization, 1st ed. John Wiley & Sons, Inc., New York, NY, USA, 1997.
- [2] ABRISHAMI, S.; NAGHIBZADEH, M.; EPEMA, D. H. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems 29*, 1 (2013), 158–169.
- [3] BERRIMAN, G. B.; DEELMAN, E.; GOOD, J. C.; JACOB, J. C.; KATZ, D. S.; KESSELMAN, C.; LAITY, A. C.; PRINCE, T. A.; SINGH, G.; SU, M.-H. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In *Optimizing Scientific Return for Astronomy through Information Technologies* (2004), vol. 5493, International Society for Optics and Photonics, pp. 221–233.
- [4] BHARATHI, S.; CHERVENAK, A.; DEELMAN, E.; MEHTA, G.; SU, M.-H.; VAHI, K. Characterization of scientific workflows. In Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on (Nov 2008), pp. 1–10.
- BRATTON, D.; KENNEDY, J. Defining a standard for particle swarm optimization. In 2007 IEEE swarm intelligence symposium (2007), IEEE, pp. 120–127.
- [6] CALLAGHAN, S.; MAECHLING, P.; DEELMAN, E.; VAHI, K.; MEHTA, G.; JUVE, G.; MILNER, K.; GRAVES, R.; FIELD, E.; OKAYA, D., ET AL. Reducing timeto-solution using distributed high-throughput mega-workflows-experiences from scec cybershake. In 2008 IEEE Fourth International Conference on eScience (2008), IEEE, pp. 151–158.
- [7] CAMPEOTTO, F.; DOVIER, A.; FIORETTO, F.; PONTELLI, E. A gpu implementation of large neighborhood search for solving constraint optimization problems. In *ECAI* (2014), pp. 189–194.
- [8] CASAS, I.; TAHERI, J.; RANJAN, R.; WANG, L.; ZOMAYA, A. Y. Ga-eti: An enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments. *Journal of computational science 26* (2016), 318–331.
- [9] CASAS, I.; TAHERI, J.; RANJAN, R.; ZOMAYA, A. Y. Pso-ds: a scheduling engine for scientific workflow managers. *The Journal of Supercomputing* 73, 9 (2017), 3924– 3947.
- [10] COELHO, I.; MUNHOZ, P.; OCHI, L.; SOUZA, M.; BENTES, C.; FARIAS, R. An integrated cpu–gpu heuristic inspired on variable neighbourhood search for the single vehicle routing problem with deliveries and selective pickups. *International Journal* of Production Research 54, 4 (2016), 945–962.

- [11] DE OLIVEIRA, D.; BAIÃO, F. A.; MATTOSO, M. Towards a taxonomy for cloud computing from an e-science perspective. In *Cloud Computing*. Springer, 2010, pp. 47–62.
- [12] 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, 7 (2013), 1816–1825.
- [13] DEB, K.; AGRAWAL, S.; PRATAP, A.; MEYARIVAN, T. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International* conference on parallel problem solving from nature (2000), Springer, pp. 849–858.
- [14] 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 *Grid Computing* (2004), Springer, pp. 11–20.
- [15] DEELMAN, E.; KESSELMAN, C.; MEHTA, G.; MESHKAT, L.; PEARLMAN, L.; BLACKBURN, K.; EHRENS, P.; LAZZARINI, A.; WILLIAMS, R.; KORANDA, S. Griphyn and ligo, building a virtual data grid for gravitational wave scientists. In Proceedings 11th IEEE International Symposium on High Performance Distributed Computing (2002), IEEE, pp. 225–234.
- [16] GAREY, M. R.; JOHNSON, D. S. Computers and intractability, vol. 29. wh freeman New York, 2002.
- [17] JIAO, L.; WANG, L. A novel genetic algorithm based on immunity. *IEEE Transac*tions on Systems, Man, and Cybernetics-part A: systems and humans 30, 5 (2000), 552-561.
- [18] KAZARLIS, S. A.; BAKIRTZIS, A.; PETRIDIS, V. A genetic algorithm solution to the unit commitment problem. *IEEE transactions on power systems* 11, 1 (1996), 83–92.
- [19] KENNEDY, J.; EBERHART, R. Particle swarm optimization. In Neural Networks, 1995. Proceedings., IEEE International Conference on (1995), vol. 4, IEEE, pp. 1942– 1948.
- [20] LIVNY, J.; TEONADI, H.; LIVNY, M.; WALDOR, M. K. High-throughput, kingdomwide prediction and annotation of bacterial non-coding rnas. *PloS one 3*, 9 (2008), e3197.
- [21] LUONG, T. V.; MELAB, N.; TALBI, E.-G. Neighborhood structures for gpu-based local search algorithms. *Parallel Processing Letters* 20, 04 (2010), 307–324.
- [22] MACIEJ, M. Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press (2012).
- [23] MASDARI, M.; VALIKARDAN, S.; SHAHI, Z.; AZAR, S. I. Towards workflow scheduling in cloud computing: a comprehensive analysis. *Journal of Network and Computer Applications 66* (2016), 64–82.

- [24] OCAÑA, K. A.; DE OLIVEIRA, D.; OGASAWARA, E.; DÁVILA, A. M.; LIMA, A. A.; MATTOSO, M. Sciphy: a cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. In *Brazilian Symposium on Bioinformatics* (2011), Springer, pp. 66–70.
- [25] PANDEY, S.; WU, L.; GURU, S. M.; BUYYA, R. A particle swarm optimizationbased heuristic for scheduling workflow applications in cloud computing environments. In 2010 24th IEEE international conference on advanced information networking and applications (2010), IEEE, pp. 400–407.
- [26] RAZALI, N. M.; GERAGHTY, J., ET AL. Genetic algorithm performance with different selection strategies in solving tsp. In *Proceedings of the world congress on engineering* (2011), vol. 2, International Association of Engineers Hong Kong, pp. 1– 6.
- [27] REEVES, C. R. A genetic algorithm for flowshop sequencing. Computers & operations research 22, 1 (1995), 5–13.
- [28] RIOS, E.; COELHO, I. M.; OCHI, L. S.; BOERES, C.; FARIAS, R. A benchmark on multi improvement neighborhood search strategies in cpu/gpu systems. In 2016 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW) (2016), IEEE, pp. 49–54.
- [29] RODRIGUEZ, M. A.; BUYYA, R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE transactions on Cloud Computing 2*, 2 (2014), 222–235.
- [30] SHI, Y.; EBERHART, R. C. Parameter selection in particle swarm optimization. In International conference on evolutionary programming (1998), Springer, pp. 591–600.
- [31] 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, 2 (2013), 245–264.
- [32] TEYLO, L.; DE PAULA, U.; FROTA, Y.; DE OLIVEIRA, D.; DRUMMOND, L. M. A hybrid evolutionary algorithm for task scheduling and data assignment of dataintensive scientific workflows on clouds. *Future Generation Computer Systems 76* (2017), 1–17.
- [33] TOPCUOUGLU, H.; HARIRI, S.; WU, M.-Y. Performance-effective and lowcomplexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* 13, 3 (Mar. 2002), 260–274.
- [34] ULLMAN, J. D. Polynomial complete scheduling problems. In ACM SIGOPS Operating Systems Review (1973), vol. 7, ACM, pp. 96–101.
- [35] VAHI, K. Executing computing pipelines using pegasus wms, 2016. Disponível em https://research.cs.wisc.edu/htcondor/HTCondorWeek2016/ presentations/TueVahi_PegasusTutorial.pdf.
- [36] VÖCKLER, J.-S.; JUVE, G.; DEELMAN, E.; RYNGE, M.; BERRIMAN, B. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd international workshop on Scientific cloud computing* (2011), ACM, pp. 15–24.

- [37] YU, Z.; SHI, W. An adaptive rescheduling strategy for grid workflow applications. In 2007 IEEE International Parallel and Distributed Processing Symposium (2007), IEEE, pp. 1–8.
- [38] ZHOU, Y.; HE, F.; QIU, Y. Optimization of parallel iterated local search algorithms on graphics processing unit. *The Journal of Supercomputing* 72, 6 (2016), 2394–2416.