

UNIVERSIDADE FEDERAL FLUMINENSE

Luan Teylo Gouveia Lima

**Escalonamento de Tarefas e Alocação de Arquivos de
Dados de Workflows Científicos em Nuvens
Computacionais**

NITERÓI

2017

UNIVERSIDADE FEDERAL FLUMINENSE

Luan Teylo Gouveia Lima

Escalonamento de Tarefas 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: Sistemas de Computação

Orientador:

Lúcia Maria de Assumpção Drummond

Co-orientador:

Yuri Abitbol de Menezes Frota

NITERÓI

2017

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

L732 Lima, Luan Teylo Gouveia

Escalonamento de tarefas e alocação de arquivos de dados de *workflows* científicos em nuvens computacionais / Luan Teylo Gouveia Lima. – Niterói, RJ : [s.n.], 2017.

73 f.

Dissertação (Mestrado em Computação) - Universidade Federal Fluminense, 2017.

Orientadores: Lucia Maria de Assumpção Drummond, Yuri Abitbol de Menezes Frota.

1. Escalonamento de tarefas. 2. Metaheurística. 3. Fluxo de trabalho. 4. Trabalho científico. 5. Computação em nuvem. 6. Sistema de computação. I. Título.

CDD 005.136

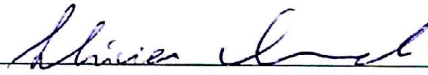
Luan Teylo Gouvêia Lima

Escalonamento de Tarefas e Alocação de Arquivos de Dados de *Workflows* Científicos
em Nuvens Computacionais

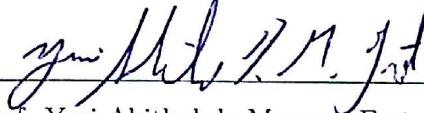
Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Sistemas de Computação

Aprovada em 17 de março 2017.

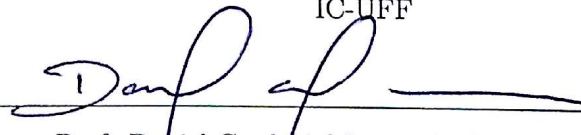
BANCA EXAMINADORA



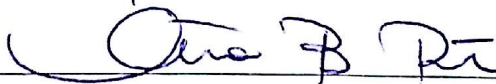
Prof. Lúcia M. A. Drummond - Orientador, IC-UFF



Prof. Yuri Abitbol de Menezes Frota - Co-orientador,
IC-UFF



Prof. Daniel Cardoso Moraes de Oliveira, IC-UFF



Prof. Cristiana Barbosa Bentes, UERJ

Niterói

2017

À minha família.

Agradecimentos

À professora Lúcia pela orientação, e por toda a confiança depositada na elaboração deste trabalho. Sou grato pela oportunidade de trabalhar com uma pessoa tão comprometida com a orientação de seus alunos e com os resultados de seu trabalho.

Ao professor Yuri pela co-orientação e pelo esclarecimento de várias ideias que fizeram deste um trabalho no qual me orgulho muito. Sem a sua ajuda e o seu bom humor o meu desempenho não seria o mesmo.

Ao professor Daniel por toda ajuda e paciência em suas explicações. Sou grato pela dedicação que você tem com os seus alunos, e por sempre ter arrumado tempo para esclarecer as minhas dúvidas.

Ao Ubiratam, sem o qual uma parte importantíssima deste trabalho não teria a mesma qualidade. Sou muito grato por poder contar com a sua experiência e dedicação.

À Pamela, com quem compartilho as minhas conquistas. Obrigado pelo relacionamento duradouro e construtivo.

Aos meus companheiros de laboratório Leonardo, Maicon e Rodrigo, que entre cafés e conversas, me proporcionaram um ambiente de trabalho rico e prazeroso.

A todos os professores e colegas que de alguma forma colaboraram para a minha formação.

Ao CNPq, pela bolsa de mestrado concedida.

Resumo

Na última década, um número crescente de experimentos computacionalmente intensivos envolvendo grandes volumes de dados têm sido modelados na forma de *workflows* científicos. Ao mesmo tempo, as nuvens computacionais surgem como um ambiente promissor para executar esse tipo de aplicação. Neste cenário, a investigação de estratégias de escalonamentos se tornaram essenciais, sendo este um campo de pesquisa extremamente popular. No entanto, poucos trabalhos consideram o problema da alocação de dados durante a resolução do problema de escalonamento de tarefas. Um *workflow* é geralmente representado como um grafo, no qual os nós equivalem às tarefas e, neste caso, o problema de escalonamento consiste em alocar essas tarefas a máquinas que as executarão em um tempo pré definido. O objetivo é reduzir o tempo total de execução de todo o *workflow*. Neste trabalho é mostrado que o escalonamento de *workflows* científicos pode ser melhorado quando os problemas de escalonamento de tarefa e alocação de dados são tratados de forma conjunta. Para isso, uma nova representação, na qual os nós do grafo representam tanto tarefas como dados, é proposta. Além disso, o problema de Escalonamento de Tarefas e Alocação de Dados é definido, considerando esse novo modelo. Esse problema foi formulado como um problema de programação inteira. Por fim, um algoritmo evolucionário híbrido capaz de escalonar tarefas e alocar os dados em ambientes de nuvens computacionais também é apresentado.

Palavras-chave: Problema de escalonamento, Alocação de Dados, *Workflow* Científico, Metaheurística.

Abstract

A growing number of data- and compute-intensive experiments have been modeled as scientific workflows in the last decade. Meanwhile, clouds have emerged as a prominent environment to execute this type of applications. In this scenario, the investigation of workflow scheduling strategies, aiming at reducing its execution times, became a top priority and a very popular research field. However, few works consider the problem of data file assignment when solving the task scheduling problem. Usually, a workflow is represented by a graph where nodes represent tasks and the scheduling problem consists in allocating tasks to machines to be executed at a predefined time aiming at reducing the makespan of the whole workflow. In this work, we show that the scheduling of scientific workflows can be improved when both task scheduling and the data file assignment problems are treated together. Thus, we propose a new workflow representation, where nodes of the workflow graph represent either tasks or data files, and define the Task Scheduling and Data Allocation Problem, considering this new model. We formulated this problem as an integer programming problem. Moreover, a hybrid evolutionary algorithm for solving it is also introduced.

Keywords: Scheduling Problem, Data Allocation, Scientific Workflow, Metaheuristic.

Lista de Figuras

| | | |
|-----|---|----|
| 2.1 | Representação das tarefas de um <i>Workflow</i> Científico. | 6 |
| 4.1 | Exemplo dos modelos de ambiente e aplicação. | 19 |
| 5.1 | Codificação do cromossomo. | 28 |
| 5.2 | Representação dos operadores de <i>crossover</i> utilizados pelo AEH-ETAA. . . | 31 |
| 5.3 | Operador de Mutação. | 33 |
| 5.4 | Procedimentos de Buscas Locais Implementados. | 35 |
| 5.5 | Cálculo da distância entre os cromossomos p_1 e p_2 , $dist(p_1, p_2) = 6$ | 36 |
| 6.1 | Estruturas básicas dos <i>workflows</i> (adaptado de [6]). | 42 |
| 6.2 | Um grafo representando uma única execução do SciPhy. | 48 |
| 6.3 | Diagrama de caixa comparando as execuções do SciPhy utilizando os algoritmos de escalonamento AEH-ETAA, Greedy-SC, HEFT e MinMin. . . . | 51 |

Lista de Tabelas

| | | |
|-----|---|----|
| 3.1 | Resumo dos trabalhos relacionados. | 17 |
| 4.1 | Descrição dos dados e das variáveis utilizadas no modelo matemático. . . . | 21 |
| 6.1 | Descrição das Instâncias Utilizadas na comparação com a formulação matemática. | 41 |
| 6.2 | Resultados da metaheurística AEH-ETAA e da formulação matemática utilizando o CPLEX - tempos em segundos. | 43 |
| 6.3 | Atributos dos <i>workflows</i> utilizados na avaliação teórica entre o AEH-ETAA, HEFT e MinMin (Adaptado de [79]). | 44 |
| 6.4 | Resultados da comparação entre AEH-ETAA, HEFT e MinMin. | 45 |
| 6.5 | Volume de dados transferidos (em MBytes) com os planos de escalonamento dados pelos algoritmos MinMin, HEFT e AEH-ETAA. | 46 |
| 6.6 | Comparação dos valores estimados e obtidos pelos algoritmos de escalonamento. | 51 |

Lista de Abreviaturas e Siglas

| | |
|----------|---|
| WfC | : <i>Workflow Científico</i> ; |
| SGWfC | : Sistema de Gerenciamento de <i>Workflow Científico</i> ; |
| HPC | : <i>High Performance Computing</i> ; |
| MV | : Máquina Virtual; |
| DAG | : <i>Directed Acyclic Graph</i> ; |
| PaaS | : <i>Platform as a Service</i> ; |
| SaaS | : <i>Software as a Service</i> ; |
| IaaS | : <i>Infrastructure as a Service</i> ; |
| ETAA | : Escalonamento de Tarefas e Alocação de Arquivos; |
| AEH-ETAA | : Algoritmo Evolutivo Híbrido para Escalonamento de Tarefas e Alocação de Arquivos; |

Sumário

| | | |
|----------|---|-----------|
| 1 | Introdução | 1 |
| 1.1 | Objetivo e Contribuições do Trabalho | 3 |
| 1.2 | Organização do Trabalho | 4 |
| 2 | Conceitos Preliminares | 5 |
| 2.1 | WfCs | 5 |
| 2.2 | Nuvens Computacionais | 6 |
| 2.3 | O Problema de Escalonamento de WfCs | 8 |
| 3 | Revisão Bibliográfica | 11 |
| 3.1 | Estratégias de Escalonamento de tarefas de <i>workflows</i> em Ambientes Distribuídos | 11 |
| 3.1.1 | Heurísticas | 11 |
| 3.1.2 | Metaheurísticas | 14 |
| 3.2 | Estratégias de Escalonamento de Tarefas e Alocação de Dados para WfCs . | 15 |
| 4 | Definição do Problema de Escalonamento de Tarefas e Alocação de Dados de WfCs em Nuvem | 18 |
| 4.1 | Descrição do Modelo Matemático | 18 |
| 4.1.1 | Modelo da Aplicação e do Ambiente | 19 |
| 4.1.2 | Formulação Matemática | 20 |
| 5 | Algoritmo Evolutivo Híbrido Para o Escalonamento de Tarefas e Alocação de Arquivos | 25 |

| | | |
|----------|--|-----------|
| 5.1 | AEH-ETAA | 25 |
| 5.1.1 | Representação do Cromossomo | 27 |
| 5.1.2 | População Inicial | 30 |
| 5.1.3 | Operador de <i>Crossover</i> e Fase de Seleção | 30 |
| 5.1.4 | Operador de Mutação | 32 |
| 5.1.5 | Procedimentos de buscas locais | 33 |
| 5.1.6 | <i>Path Relinking</i> | 36 |
| 5.1.7 | Heurística <i>move-arquivos</i> | 37 |
| 5.1.8 | Função de <i>fitness</i> | 38 |
| 6 | Resultados Experimentais | 40 |
| 6.1 | Experimentos Teóricos | 40 |
| 6.1.1 | Comparação do AEH-ETAA com a Abordagem Exata IP-ETAA . . | 41 |
| 6.1.2 | Comparação do AEH-ETAA com as Heurísticas HEFT e MinMin . | 44 |
| 6.2 | Avaliação do AEH-ETAA com <i>Workflow</i> Real | 47 |
| 6.2.1 | SciPhy: Um <i>workflow</i> para análises filogenéticas | 47 |
| 6.2.2 | SGWfC SciCumulus | 48 |
| 6.2.3 | Configuração do Ambiente | 49 |
| 6.2.4 | Configuração do Experimento | 50 |
| 6.2.5 | Resultados Experimentais do SciPhy | 50 |
| 7 | Conclusões e Trabalhos Futuros | 52 |
| | Referências | 54 |

Capítulo 1

Introdução

Avanços recentes na ciência da computação têm permitido que diferentes campos científicos se beneficiem de simulações computacionais. Estudos em dinâmica de fluidos [43], astronomia [27], análises filogenéticas [67] e bioinformática [58], são alguns dos exemplos nos quais os chamados experimentos *in silico* [61, 81] desempenham um papel fundamental no desenvolvimento e na aquisição de novos conhecimentos. Porém, essas aplicações estão produzindo e consumindo um volume inédito de dados, fazendo com que problemas relacionados as limitações computacionais e ao gerenciamento de recursos sejam constantemente enfrentados pelos cientistas [15].

Os experimentos científicos são comumente representados como uma cadeia de aplicações, nas quais a saída de um programa é a entrada para outro. Neste contexto, os *workflows* científicos (WfCs) destacam-se como uma solução promissora para elaborar e gerenciar esses experimentos. 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 elas [61]. Esses *workflows* são gerenciados pelos Sistemas de Gerenciamento de *Workflows* Científicos (SGWfCs), que são utilizados para definir, executar e monitorar essas atividades. Alguns exemplos de SGWfCs são: Swift/T [91], Pegasus [28], VisTrails [13], Apache Taverna [90] e Kepler [2].

Conforme a complexidade dos WfCs cresce, em termos de quantidade de dados processados e tempo de execução, seus requisitos de performance excedem as capacidades dos sistemas sequenciais (computadores pessoais com poucos processadores, por exemplo). Se executados sequencialmente, esses *workflows* podem demorar dias ou até mesmo meses para finalizar, o que não é desejado, devido à propensão a erros de execução, e até mesmo pela competitividade existente no meio científico hoje em dia [35]. Consequentemente, os ambientes de computação de alto desempenho (HPC, do inglês *High Performance Com-*

puting) aliados às técnicas de paralelismo, tornaram-se essenciais para a obtenção de resultados em tempo aceitável.

Ambientes como *clusters*, super computadores e *grids* computacionais, são tradicionalmente utilizados para a execução de aplicações HPC. No entanto, a computação em nuvem [86] vem se destacando como uma opção promissora para a execução dessa classe de aplicações [56]. A computação em nuvem é um tipo de serviço baseado na Internet no qual a infraestrutura computacional é virtualmente ilimitada, a plataforma e o *software* são providos sob demanda, seguindo o modelo de cobrança *pay-per-use* [92, 21], no qual os usuários são cobrados pelos recursos efetivamente utilizados. Ao contratar esses serviços, a necessidade de aquisição de infraestruturas de alto custo (como *clusters*), e os esforços despendidos com configurações complexas (como ocorre nos *grids*), são substituídos pela aquisição de máquinas virtuais (MVs) pré configuradas e prontas para o uso. A facilidade em obter recursos computacionais que atendem a diferentes necessidades e custo monetário flexível são alguns dos atrativos que despertam o interesse da comunidade científica por esses ambientes [23].

Para permitir a execução de WfCs em ambientes de nuvens computacionais é necessário escalonar cada tarefa que compõe o *workflow* para uma das MVs disponíveis. Assim, a grosso modo, um algoritmo de escalonamento busca mapear tarefas a recursos, de forma que os requisitos definidos pelo usuário, pelas aplicações ou pelo provedor sejam atendidos [7]. O escalonamento de tarefas em recursos distribuídos é um problema NP-Difícil [85] e há algumas características das nuvens computacionais que fazem com que esse processo seja ainda mais complicado.

Primeiramente, os serviços de nuvem disponibilizam vários tipos de MVs, cada uma com diferentes capacidades de processamento, armazenamento, taxas de transferência e custo financeiro, sendo que algumas dessas MVs não são adequadas para HPC (por exemplo as MVs do tipo micro e nano na Amazon EC2). Além disso, muitos dos *workflows* existentes consomem e produzem vários GB ou mesmo TB de dados, e esses dados (ou pelo menos uma parte deles) são transferidos de uma MV para outra, o que pode impactar o tempo total de execução do *workflow*. Por exemplo, uma única execução do Montage [27], um *workflow* utilizado para gerar mosaicos customizados de imagens astronômicas, pode processar em torno de 200 GB de dados [50]. Se, durante a execução, esses dados forem transferidos inúmeras vezes entre as MVs, uma parte considerada do tempo total do experimento será gasta em transferência ao invés de processamento (que é o foco do experimento). Sendo assim, ao escalonar as tarefas de um *workflow* é necessário evitar

transferências desnecessárias ou, quando a transferência é inevitável, ao menos minimizar o seu impacto no tempo total de execução.

O escalonamento de dados e tarefas em sistemas distribuídos é um tópico largamente discutido nos ambientes de *grids* e *clusters* [30, 82, 49, 71]. Em [71], 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 mostram a importância de tratar o escalonamento de tarefas e dados de forma conjunta. Nos últimos anos, várias abordagens heurísticas para o escalonamento de *workflows* científicos em ambientes de nuvens foram propostas [57, 69, 96, 79, 11, 22, 68]. No entanto, soluções que consideram tanto a distribuição dos dados, quanto a alocação de tarefas foram pouco exploradas.

Neste trabalho, uma nova abordagem para o escalonamento de *workflows* científicos em ambiente de nuvem é apresentada e avaliada. A abordagem considera o escalonamento das tarefas e a distribuição dos dados de forma conjunta, como partes de um mesmo problema. Além disso, para validar a solução, é apresentado um algoritmo evolutivo híbrido [65], que escala tarefas e dados considerando a heterogeneidade das MVs, as características do ambiente e as restrições impostas aos dados.

1.1 Objetivo e Contribuições do Trabalho

Este trabalho propõe uma solução para o problema de escalonamento de tarefas e alocação de arquivos de WfCs executados em ambientes de nuvens computacionais. Para isso, um modelo matemático que considera as características das aplicações, do ambiente e a junção entre o escalonamento de tarefas e alocação de arquivos é proposto. Além disso, também é apresentado um algoritmo evolutivo híbrido, que foi avaliado utilizando instâncias de *workflows* sintéticos e reais. O objetivo é minimizar o tempo total de execução (*makespan*) dos *workflows* nestes ambientes e demonstrar a viabilidade da nova abordagem.

Sendo assim, as principais contribuições deste trabalho se resumem nos seguintes pontos:

1. Um modelo de representação para *workflows* científicos utilizando grafo, no qual os nós correspondem às atividades de processamento ou arquivos de dados, e os arcos representam as operações de leitura e escrita
2. Formulação do problema de escalonamento de tarefas e alocação de arquivos de

dados como um problema de programação inteira mista, nomeado IP-ETAA

3. Implementação de um algoritmo evolutivo híbrido para o escalonamento de tarefas e dados em ambientes de nuvem, chamado de AEH-ETAA
4. Avaliação experimental baseada em *workflows* sintéticos, e em execuções reais utilizando o serviço da Amazon EC2.

1.2 Organização do Trabalho

O trabalho está dividido em 7 capítulos, incluindo a introdução.

O Capítulo 2 introduz os conceitos principais dos *workflows* científicos, das nuvens computacionais e do problema de escalonamento. No Capítulo 3 os trabalhos relacionados são discutidos. A definição do problema, que inclui o modelo da aplicação, do ambiente e a formulação matemática, é abordada no Capítulo 4.

No Capítulo 5, o algoritmo evolutivo híbrido proposto é apresentado. No Capítulo 6, os resultados das avaliações teóricas e práticas são apresentados e discutidos. Por fim, o Capítulo 7 finaliza o trabalho com as conclusões e os trabalhos futuros.

Capítulo 2

Conceitos Preliminares

Este capítulo apresenta os conceitos relacionados aos tópicos de estudo deste trabalho. Na Seção 2.1, os *workflows* científicos são apresentados. Em seguida, na Seção 2.2, as nuvens computacionais são abordadas. Por fim, na Seção 2.3 o problema de escalonamento de *workflows* científicos em ambientes de nuvens é apresentado.

2.1 WfCs

As tecnologias de *workflow* foram primeiramente adotadas pela comunidade empresarial, nos chamados *business workflows*. Em 1995 a *Workflow Management Coalition* [47] definiu essa tecnologia como a automação parcial ou total de um processo, durante o qual documentos, informações ou tarefas são passados de um recurso (humano ou computacional) a outro, para que sejam executadas ações de acordo com um conjunto procedural de regras. Essa definição, embora considere o cenário empresarial, concede a visão dos conceitos originais nos quais os *workflows* científicos foram baseados. O termo *workflow* científico (WfC) foi cunhado para descrever a automatização de experimentos científicos executados em ambientes computacionais utilizando as tecnologias de *workflow* [5].

Um WfC é composto por uma série de etapas analíticas, que descrevem o processo de experimentação computacional. Esses sistemas fornecem um ambiente para auxiliar a descoberta científica através da combinação entre gerenciamento de dados, análises, simulação, e visualização [5, 26]. Alguns exemplos de WfC são: Cybershake [25], utilizado pela *Southern California Earthquake Center* para caracterizar o risco de terremoto de uma região; Epigenomics [50], usado na automação de várias operações de sequenciamento de genomas; e Ligo *Inpiral Analysis* [10], empregado pelo *Laser Interferometer Gravitational Wave Observatory* (LIGO) para analisar os dados obtidos nas observações

de ondas gravitacionais.

Um *workflow* é definido geralmente como um grafo acíclico dirigido (em inglês *Directed Acyclic Graph*, DAG), no qual os vértices representam as etapas do experimento, também chamadas de atividades, e as arestas descrevem as relações de dependências entre estas. As atividades de um WfC consistem em várias tarefas executáveis, que consomem partes diferentes dos dados (paralelismo de dados) [57]. Um WfC pode ter centenas ou mesmo milhares de tarefas, que além de processarem uma grande quantidade de dados (*data-intensive*), também podem executar durante várias horas, ou mesmo dias (*compute-intensive*) [50].

A Figura 2.1 representa um *workflow* composto por três atividades (A_i , A_k e A_y). Como pode ser visto, as atividades A_i e A_y são executadas por n tarefas ($n > 0$), e a atividade A_k é executada por uma única tarefa. A relação de precedência entre as tarefas é definida por meio de troca de dados. Portanto, como os dados de saída de uma tarefa são os de entrada de outra, a tarefa só pode ser executada quando todos os seus dados de entradas estiverem disponíveis. Por exemplo, na Figura 2.1 a tarefa t_{k1} tem relação de dependência com todas as tarefas de A_i e, portanto, só poderá ser executada quando todas as tarefas de A_i estiverem finalizadas.

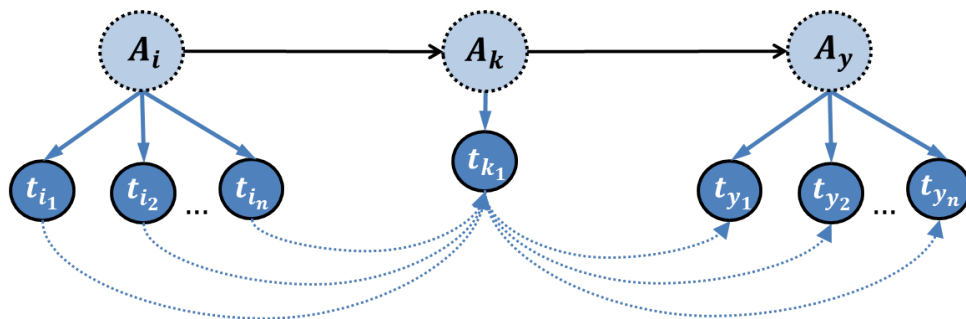


Figura 2.1: Representação das tarefas de um *Workflow* Científico.

2.2 Nuvens Computacionais

Ao longo dos últimos anos, várias definições para nuvens computacionais foram apresentadas [29, 4, 87]. Essas definições variam entre o ponto de vista técnico, que aborda os conceitos relacionados a arquitetura e a implementação do ambiente físico e virtual, e o ponto de vista de usabilidade, focado essencialmente nos serviços prestados e nos modelos de cobrança [52]. Segundo a ótica de serviço, as nuvens computacionais são um conjunto de servidores virtuais que trabalham interligados através da Internet, e podem ser dinamicamente gerenciados e monitorados [45]. Esses ambientes se assemelham aos *clusters*

nos aspectos técnicos, porém, diferente destes, os recursos computacionais são oferecidos através de virtualização e seguem o modelo de pagamento pague-pelo-uso (do inglês, *pay-per-use*), no qual os usuários são cobrados pelo tempo de uso dos recursos contratados [52].

As nuvens computacionais oferecem várias vantagens técnicas e econômicas em relação as outras plataformas (como *grids* e *clusters*), pois combinam virtualização e escalabilidade em modelos de serviços economicamente viáveis [52]. Entre as principais vantagens do uso desse ambiente estão [44]: processamento paralelo sob demanda; ilusão de recursos infinitos; e serviços integrados de processamento de dados e armazenamento escalável. Segundo Hashem *et al.* [44], além do potencial em diminuir os custos ligados a automação e processamento de dados para indivíduos e empresas, as nuvens computacionais podem reduzir significativamente os custos de aquisição, manutenção, gerenciamento e acesso a infraestruturas computacionais.

Os serviços prestados pelas nuvens são divididos em três categorias [44]:

- Plataforma como serviço (PaaS, do inglês *Platform as a Service*), que consiste em diferentes recursos (sistemas operacionais, bibliotecas, compiladores, etc.) operando em conjunto para fornecer uma plataforma ao usuário final. Google Apps Engine [40] e Microsoft Azure [62], são exemplos deste serviço.
- *Software* como serviço (SaaS, do inglês *Software as a Service*), consiste em aplicações operadas remotamente em uma infraestrutura de nuvem e oferecidas como um serviço. Exemplos deste modelo são Google Docs [42], Gmail [39], Sharelatex [75], etc.
- Infraestrutura como serviço (IaaS, do inglês *Infrastructure as a Service*, refere-se a equipamentos de *hardware* virtualizados que são fornecidos sob demanda por um provedor de nuvem. Neste modelo, o cliente tem controle total de configuração e instalação de *softwares* no *hardware* adquirido. Amazon EC2 [3], Google Cloud Platform [41] e Microsoft Azure [62] são exemplos de IaaS.

A virtualização é umas das principais tecnologias por trás do sucesso das nuvens computacionais, e pode ser definida como um processo de compartilhamento de recursos computacionais (como CPU, espaço de armazenamento e rede) que isola o *hardware* físico, diminuindo a ineficiência na alocação e distribuição de seus recursos [44]. O uso mais comum de virtualização é através das máquinas virtuais (MVs), que criam ambientes de *hardware* e *software* configurados de forma totalmente independente do recurso

físico, o que permite que várias MVs independentes sejam executadas ao mesmo tempo sob um mesmo *hardware* [45]. A Amazon EC2, por exemplo, utiliza MVs para oferecer *hardware* virtualizado com diferentes capacidades de memória, largura de banda e poder de processamento. As MVs são classificadas conforme o propósito de uso e capacidade de recursos, e apresentam custo monetário variável. Atualmente, A MV mais barata oferecida pela Amazon EC2 é chamada de t2.nano, e custa \$0.0059 hora. A máquina contém um processador virtual (vCPU) baseado no Intel Xeon e 500MB de memória RAM. Já a máquina mais cara, chamada de d2.8xlarge, tem custo de \$5.52 hora, 36 vCPUs (também baseadas no Intel Xeon) e 244 GB de memória RAM [3].

O uso dos serviços de nuvem para o processamento de aplicações distribuídas se baseia no modelo IaaS e é feito através da alocação de um *cluster* virtual, que pode ser composto por máquinas de um mesmo tipo (ambiente homogêneo) ou por máquinas com diferentes configurações (ambiente heterogêneo). Especificamente no caso dos WfCs, o SGWfC é responsável pela alocação das MVs necessárias e pelo planejamento de execução das tarefas do *workflow*, feito pelos algoritmos de escalonamento.

2.3 O Problema de Escalonamento de WfCs

Segundo Yu *et al.* [94], escalonamento é o processo de mapear e gerenciar a execução de tarefas em um ambiente computacional. Portanto, é papel do escalonador definir onde e quando uma tarefa deverá ser executada, sendo que essa decisão está sujeita às restrições impostas às aplicações e ao ambiente. Segundo Topcuoglu *et al.* [82], em sistemas distribuídos, o escalonamento eficiente das tarefas é um fator chave para atingir um alto desempenho computacional.

Formalmente, o objetivo do escalonador é alocar um conjunto de tarefas $N = \{tf_1, tf_2, \dots, tf_n\}$ em um conjunto de máquinas $M = \{mv_1, mv_2, \dots, mv_n\}$. O escalonador deve otimizar uma função objetivo f , que qualifica a solução encontrada e está sujeito a restrições que, caso não sejam satisfeitas, inviabilizam a solução de escalonamento [83]. Os objetivos mais comuns no escalonamento de WfCs são: minimizar o tempo de execução (*makespan*); minimizar o custo monetário; maximizar o uso dos recursos (balanceamento de carga); e minimizar a ocorrência de falhas. No caso das restrições, limitar o orçamento e definir um prazo de execução, são comumente empregadas [60].

O escalonamento de tarefas em sistemas distribuídos faz parte dos chamados problemas NP-Difícil [85]. Isso significa que não há algoritmo conhecido capaz de produzir

soluções ótimas em tempo polinomial (a menos que $P = NP$). Por conta disso, as abordagens de escalonamento utilizam geralmente algoritmos de resultado aproximados, chamados de heurísticas de escalonamento [82, 60, 94]. Essas abordagens não garantem que a solução encontrada seja ótima, isto é, a melhor solução possível para o problema. No entanto, são capazes de encontrar soluções com qualidade aceitável, em tempo viável de execução [54]. As heurísticas são classificadas em diferentes categorias, conforme o critério utilizado na manipulação das tarefas. Uma dessas categorias são os métodos randômicos de busca guiados, chamados também de metaheurísticas [54, 8, 83]. Segundo Boussaïd *et al.* [8], metaheurísticas são algoritmos elaborados para resolver, de forma aproximada, um grande número de problemas difíceis de otimização sem que seja necessário realizar adaptações profundas no algoritmo para cada um destes problemas.

Os algoritmos de escalonamento podem ser estáticos ou dinâmicos [34]. Nos algoritmos estáticos todo o plano de escalonamento é definido antes de qualquer execução. Para isso, os algoritmos consideram as condições iniciais do ambiente (como capacidades de processamento, número de máquinas e taxas de transferência) e não preveem mudanças destas condições ao longo da execução. Portanto, todas as informações relacionadas ao *workflow* e ao ambiente, como tempo de execução das tarefas, tamanho dos dados e relações de dependências, devem ser conhecidas *a priori* e fazem parte da entrada do problema. A acurácia dessas informações influencia diretamente no resultado do escalonamento, já que informações discrepantes induzem o escalonador ao erro e resultam em soluções de baixa qualidade. As formas mais comuns de obter essas informações é através de execuções prévias do *workflow*, ou com estimativas por modelos matemáticos e testes de *benchmark*.

Nos algoritmos dinâmicos, o escalonamento é realizado durante a execução do *workflow*. Nessa abordagem, as tarefas são escalonadas por etapas, conforme a disponibilidade do ambiente e das tarefas. Para isso, o escalonador monitora o status de execução e das máquinas e, caso haja tarefas prontas para serem escalonadas e máquinas ociosas, submete as tarefas para a execução conforme um critério de escalonamento que atenda ao objetivo desejado. Normalmente, essa abordagem é local, isto é, o escalonador utiliza as informações das tarefas prontas e das máquinas livres para determinar a alocação, e não considera todo o *workflow* na decisão [34, 54]. Diferente do escalonamento estático, as informações utilizadas no escalonamento são atualizadas de tempos em tempos, o que melhora a acurácia delas, já que as oscilações de qualidade do ambiente são facilmente detectadas e o escalonamento pode ser adaptado. Embora o escalonamento dinâmico seja mais flexível e exija menos parâmetros de entrada, seu *overhead* de execução e a complexi-

dade na implementação são maiores quando comparados aos algoritmos de escalonamento estático [34].

Capítulo 3

Revisão Bibliográfica

Este capítulo apresenta o levantamento bibliográfico efetuado ao longo do trabalho. Os artigos apresentados foram divididos em dois tópicos principais: estratégias de escalonamento de tarefas de *workflows* em ambientes distribuídos (Seção 3.1); e estratégias de escalonamento com alocação de dados (Seção 3.2).

3.1 Estratégias de Escalonamento de tarefas de *workflows* em Ambientes Distribuídos

Os algoritmos de escalonamento podem ser divididos em dois grupos de interesse [77]: algoritmos direcionados ao usuário e algoritmos direcionados ao provedor. A principal diferença entre esses grupos é em relação aos objetivos do escalonamento, isto é, enquanto os algoritmos do lado provedor concentram-se principalmente nas camadas físicas, como distribuição da carga entre os *data centers* e a minimização dos gastos energéticos, os orientados aos usuários concentram-se nos atributos relacionados a contratação dos serviços, como o tempo de execução das aplicações e os custos monetários associados ao uso dos recursos. Como a proposta deste trabalho é voltada aos interesses dos cientistas, este levantamento bibliográfico aborda apenas as soluções de escalonamento voltadas aos usuários. Para auxiliar na leitura, um resumo dos trabalhos discutidos é apresentado na Tabela 3.1.

3.1.1 Heurísticas

Segundo o levantamento feito em [60], a maioria das propostas de escalonamento de *workflows* são baseadas em heurísticas. Neste contexto, uma das heurísticas mais utilizadas é

o *Heterogeneous Earliest-Finish-Time* (HEFT) [82]. O HEFT é uma extensão para ambientes heterogêneos dos algoritmos clássicos de *list scheduling* [74]. O algoritmo executa o escalonamento em duas fases: *fase de ranqueamento*, que calcula a prioridade de cada tarefa; e *fase de seleção de máquina*, que escolhe uma tarefa de acordo com o valor de prioridade, e a atribui para a máquina na qual seu tempo de execução será o menor. Inúmeras variações do HEFT foram propostas para o escalonamento de WfCs [95, 17, 32, 33]. Em [95] um algoritmo de escalonamento adaptativo para ambientes de *grids* é apresentado. O algoritmo, chamado de AHEFT, utiliza o escalonamento definido pelo HEFT e executa reescalamentos caso sejam detectadas mudanças no ambiente, como a adição ou remoção de máquinas. O objetivo do AHEFT é minimizar o *makespan*. Em Chopra e Singh [17], o escalonamento dado pelo HEFT é também combinado a técnicas de reescalamento. Porém, diferente de [95], os autores executam o escalonamento de *workflows* em ambientes híbridos, que combinam nuvens públicas e privadas. O HEFT é utilizado na execução inicial (realizada na nuvem privada), e caso o *makespan* resultante seja maior que o tempo máximo de execução definido pelo usuário, a heurística seleciona um conjunto de tarefas para serem processadas na nuvem pública.

Em Durillo, Fard e Prodan [32], o *Multi-Objective* HEFT (MOHEFT) é proposto. O MOHEFT computa um conjunto de soluções, chamadas de Fronteira de Pareto. Essas soluções apresentam uma condição de balanceamento, na qual a melhora de um dos objetivos implica na piora de outra. Em [32], o MOHEFT foi aplicado a ambientes de nuvens públicas, mais especificamente nos serviços da Amazon EC2. O objetivo era minimizar o *makespan* e diminuir os custos financeiros. Em outro trabalho [33], o MOHEFT foi avaliado em *clusters* heterogêneos compostos por 100 nós, o objetivo do trabalho foi minimizar o gasto energético e o *makespan*. Em ambas as avaliações o MOHEFT apresentou soluções viáveis e resultados melhores que heurísticas mais simples, como o próprio HEFT por exemplo. Embora os algoritmos baseadas no HEFT considerem aspectos relacionados ao ambiente, como a capacidade de processamento das máquinas e as taxas de transferências, as restrições de armazenamento não são consideradas. Além disso, os algoritmos também não levam em conta a localização dos dados para decidir o escalonamento das tarefas.

O algoritmo guloso MinMin [7] também foi utilizado no escalonamento de *workflows*. O algoritmo escalona as tarefas prontas para serem executadas com base nas informações locais, como tempo de execução das tarefas e a capacidade de processamento das máquinas. A cada iteração, o tempo de término de cada tarefa em relação as máquinas disponíveis é estimado, e a tarefa que apresentar o menor tempo é escalonada primeiro.

Essas etapas são repetidas até que não haja mais tarefas para serem escalonadas. Segundo Blythe *et al.* [7] a ideia por trás da heurística é garantir que o tempo total de execução seja incrementado aos poucos, de modo que o *makespan* resultante seja minimizado. Porém, como a heurística não considera o grafo na decisão de escalonamento e o modelo de ambiente considera apenas o poder de processamento das máquinas e as taxas de transferências, a solução resultante pode ser de baixa qualidade e até mesmo inviável, caso haja restrições de armazenamento.

Heurísticas baseadas em particionamento de grafo também foram propostas para este problema [12, 1]. A ideia geral dos algoritmos de particionamento é definir e alocar subgrafos, de modo que restrições e objetivos sejam satisfeitos. Byun *et al.* [12] apresentaram a heurística PBTS (*Partitioned Balanced Time Scheduling*), cujo objetivo é minimizar o tempo de execução, respeitando um tempo limite definido pelo usuário. O algoritmo constrói o escalonamento iterativamente, com base no período mínimo de contratação das MVs, que é definido pelo provedor da nuvem. No Amazon EC2, por exemplo, esse período é de 60 minutos, ou seja, se a aplicação for executada por 61 minutos, o usuário é cobrado por 2 períodos (120 minutos). O PBTS define, para cada período, o número mínimo de MVs suficiente para executar as tarefas de uma partição. Nessa abordagem, o usuário define um tempo limite de execução para todo o *workflow*, que é então dividido em sub-limites e atribuídos para cada partição (subgrafos do *workflow*). O algoritmo considera os tempos de execução das tarefas e de transferência dos dados, e assume que a comunicação entre tarefas é feita exclusivamente via o Amazon S3 (<https://aws.amazon.com/pt/s3/>), que é um serviço de armazenamento compartilhado oferecido pela Amazon [3]. Embora o uso do Amazon S3 facilite as trocas de dados entre as tarefas do *workflow*, as alocações individuais dos arquivos entre as máquinas não podem ser exploradas. Além disso, conforme demonstrado por Juve *et al.* [53], o Amazon S3 apresenta uma baixa performance para *workflows* compostos por pequenos arquivos, o que é comum em WfCs.

A heurística IC-PCP (*IaaS Cloud Partial Critical Paths*) [1] constrói subconjuntos de tarefas com base nos caminhos críticos do *workflow*. Cada um desses subconjuntos é escalonado para uma MV, escolhida conforme seu valor financeiro e sua capacidade de processamento. Durante a escolha das máquinas virtuais, o algoritmo dá preferência às máquinas que já estão em uso. Caso seja necessário alocar novas máquinas, a MV mais barata com capacidade de processamento suficiente para executar o subgrupo de tarefas dentro do tempo limite de execução é escolhida. A busca por caminhos críticos e todo o processo de escalonamento é repetido até que não haja tarefas para serem escalonadas. Embora o agrupamento de tarefas seja capaz de reduzir os custos de transferências, a

localização dos dados não é implicitamente explorado por essa abordagem.

3.1.2 Metaheurísticas

As metaheurísticas foram largamente empregadas no escalonamento de WfCs em diferentes ambientes distribuídos. Entre as principais metaheurísticas adotadas destacam-se as baseadas em população, tais como, *Particle Swarm Optimization* (PSO) [55], *Ant Colony Optimization* (ACO) [31] e Algoritmos Genéticos (AG) [38].

PSO é uma metaheurística baseada no comportamento social de animais, como por exemplo um bando de pássaros buscando comida ou um cardume protegendo-se de predadores. No PSO, o movimento das chamadas partículas é análogo ao "caminhar" de um indivíduo sobre o espaço de busca e sua posição, em um dado intervalo de tempo, é baseada na melhor posição conhecida e na posição da melhor partícula do grupo [69]. Pandey *et al.* [69] utilizam o PSO como parte de uma heurística dinâmica de escalonamento, cujo objetivo é 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 a heurística avaliada. Em Rodriguez e Buyya [73] é apresentado um PSO para o escalonamento estático de WfCs em ambientes de nuvem, cujo objetivo é minimizar o custo financeiro atendendo à restrição de tempo de execução imposta pelo usuário. Diferente de [69], que considera recursos homogêneos, a solução de [73] define as MVs que serão contratadas, o período de contratação e o plano de escalonamento. Embora ambos considerem os tempos de transferências dos arquivos na decisão de escalonamento, a distribuição dos dados no ambiente não é levada em consideração pelos escalonadores.

ACO é uma metaheurística inspirada no comportamento cooperativo desempenhado pelas formigas durante a busca de alimentos [31]. Essa metaheurística tem sido aplicada com sucesso em vários problemas reais, inclusive em problemas de otimização combinatória. Chen e Zhang [16] apresentam uma solução de escalonamento multi-objetivo para *workflows* executados em *grids*. Os autores propõem uma metaheurística baseada em ACO para atender a três dos principais parâmetros de QoS: confiabilidade dos serviços; custo financeiro; e *makespan*. São definidas três classes de otimização: otimização de confiabilidade; otimização de *makespan*; e otimização de custo. Essas classes buscam maximizar (ou minimizar, no caso do *makespan*) uma métrica de QoS específica e atender as restrições impostas pelas demais métricas. Para a atualização do feromônio, os autores propuseram sete diferentes heurísticas que são selecionadas em tempo de execução, con-

forme um esquema adaptativo de própria autoria. O algoritmo obteve em média valores de custo financeiro entre 20% e 30% menores do que as heurísticas avaliadas. Em [48] o algoritmo *knowledge-based ant colony optimization* (KBACO) é apresentada. O KBACO constrói escalonamentos estáticos para *workflows* em *grids* utilizando a metaheurística ACO juntamente com heurísticas que utilizam o aprendizado acumulado a cada etapa da construção para melhorar as decisões de escalonamento. O escalonamento deve garantir que um limite de tempo de execução imposto pelo usuário ao *workflow* seja atendido. Ambas as abordagens não consideram as características da rede (taxa de transferência, por exemplo) na decisão de escalonamento.

Yu e Buyya [93] apresentaram um GA para o problema de escalonamento estático de *workflows* para ambientes de nuvem. Nessa proposta, o usuário pode definir qual restrição (tempo máximo de execução ou custo monetário máximo) a metaheurística deverá atender. A solução é representada por uma codificação bidimensional (máquinas x tarefas), e os métodos *Best-fit* e *Round-Robin* são empregados para gerar a população inicial. As abordagens clássicas de troca e recombinação em dois pontos são empregadas no operador de mutação e no de *crossover*, respectivamente.

3.2 Estratégias de Escalonamento de Tarefas e Alocação de Dados para WfCs

Alguns trabalhos consideraram o impacto causado pela transferência de dados no tempo de execução dos *workflow*, outros, chamados de *data-aware*, executam o escalonamento dos dados e das tarefas, de forma separada.

Em Szabo *et al.* [79], o impacto das transferências de dados no escalonamento de WfCs é discutido. Os autores argumentam que por conta do aumento no volume de dados transferidos entre as tarefas de um *workflow*, as soluções de escalonamento devem considerar a relação entre dados e tarefas para definir o plano de escalonamento. Os autores apresentaram um algoritmo evolutivo que otimiza o escalonamento de tarefas tendo em vista a diminuição das transferências entre essas e a minimização do tempo de execução do *workflow*. Nesse trabalho, um modelo de transferência e de armazenamento de dados baseado no Amazon S3 é empregado. Além disso, a distribuição dos dados é definida conforme a alocação das tarefas, isto é, os arquivos de saída das tarefas são escritos tanto no S3, quanto na máquina associada a tarefa. Como resultado, apenas as tarefas alocadas para essa mesma máquina tiram vantagem da alocação dos arquivos,

sendo que as outras tarefas devem fazer o *download* diretamente do S3.

Yuan *et al.* [96] utilizam uma técnica de clusterização [9] baseada em uma matriz de dependência para identificar os arquivos que serão consumidos (isto é, compartilhados) pelas mesmas tarefas do *workflow*. A ideia é armazenar cada arquivo no *data-center* que contenha o maior número de tarefas que depende deles e, dessa forma, reduzir as transferências entre *data-centers*. O problema de alocação de tarefas e a distribuição dos arquivos não é profundamente explorado pelo algoritmo de Yuan *et al.*, pois os escalonamentos de tarefas e arquivos para as MVs dentro de um mesmo *data-center* não são realizados.

Wang *et al.* [89] apresentam uma solução cujo objetivo é minimizar as transferências de dados em *workflows* alocados a múltiplos *data-centers*. O trabalho define uma localização inicial para os dados estáticos utilizando o algoritmo de clusterização k-means [9]. Em seguida, uma técnica de replicação de tarefas é utilizada para reduzir as transferências entre diferentes *data-centers* dos dados produzidos durante a execução do *workflow* (dados dinâmicos). Além da complexidade inerente às técnicas de replicação, como garantir a consistência dos dados, a abordagem proposta por Wang *et al.* não trata dos problemas de transferências dentro de um mesmo *data-center*.

Bryk *et al.* [11] propõem um modelo para execução de múltiplos *workflows* em ambientes de nuvem. O algoritmo *File Locality-Aware scheduling* (FLA-S), que se beneficia da localização dos dados para aumentar a performance de execução dos *workflows* é apresentado. O algoritmo executa uma alocação dinâmica de tarefas, na qual, a cada etapa, as tarefas prontas para a execução são escalonadas para as MVs disponíveis. O escalonador prioriza algumas tarefas, considerando a localização dos dados, com o objetivo de minimizar as transferências. O modelo considera que os dados sejam armazenados em um sistema de arquivos centralizado e compartilhado. Porém, cada MV mantém localmente uma cópia dos arquivos que foram gerados nela, isto é, pelas tarefas alocadas à ela. Portanto, embora a alocação dos arquivos não seja definida pelo algoritmo, as tarefas que consomem os mesmo arquivos, ou que apresentam relação de dependência, são preferencialmente alocadas à mesma MV, evitando assim transferências entre as máquinas e o sistema centralizado de arquivos.

Çatalyürek *et al.* [14] apresentam um algoritmo para a alocação de dados e tarefas de *workflow* executados na nuvem. O *workflow* é modelado como um hiper-grafo. Um algoritmo de particionamento é proposto, cujo objetivo é dividir o *workflow* em k partes (onde k é o número de máquinas virtuais do ambiente), e associa cada uma delas a

uma MV diferente, e minimiza o número de transferências de dados. Além disso, o particionamento deve atingir um balanceamento previamente definido pelo usuário. Os autores não consideram característica do ambiente, tal como capacidade de processamento e de armazenamento e taxas de transferências. Apenas o tamanho total dos arquivos transferidos é utilizado para avaliar a qualidade da solução.

Tabela 3.1: Resumo dos trabalhos relacionados.

| Ref. | Algoritmo | Objetivos | Restrições | Ambiente |
|-------------------------------|-------------------|--|---------------------------|----------------|
| Yu e Shi [95] | AHEFT | <i>Makespan</i> | - | <i>Grids</i> |
| Chopra e Singh [17] | HEFT-Based | Custo financeiro | Tempo de execução | Híbrido |
| Durillo, Fard e Prodan [32] | MOHEFT | <i>Makespan</i> e custo financeiro | - | Nuvem |
| Durillo, Nae e Prodan [33] | MOHEFT | <i>Makespan</i> e consumo de energia | - | <i>Cluster</i> |
| Blythe <i>et al.</i> [7] | MinMin | <i>Makespan</i> | - | <i>Grids</i> |
| Byun <i>et al.</i> [12] | PBTS | <i>Makespan</i> | Tempo de execução | Nuvem |
| Abrishami <i>et al.</i> [1] | IC-PCP | Custo financeiro | Tempo de execução | Nuvem |
| Pandey <i>et al.</i> [69] | PSO | Custo financeiro | - | Nuvem |
| Rodriguez e Buyya [73] | PSO | Custo financeiro | Tempo de execução | Nuvem |
| Chen e Zhang [16] | ACO | Vários objetivos | Várias restrições | <i>Grids</i> |
| Hu <i>et al.</i> [48] | KACO | - | Tempo de execução | <i>Grids</i> |
| Yu e Buyya [93] | GA | - | Tempo de execução e custo | Nuvem |
| Szabo <i>et al.</i> [79] | GA | <i>Makespan</i> e tempo de transferência | - | Nuvem |
| Yuan <i>et al.</i> [96] | <i>Clustering</i> | Número de transferências | - | Nuvem |
| Wang <i>et al.</i> [89] | k-means | Número de transferências | - | Nuvem |
| Bryk <i>et al.</i> [11] | FLA-S | Número de transferências | - | Nuvem |
| Çatalyürek <i>et al.</i> [14] | Partition | Número de transferências | - | Nuvem |

Capítulo 4

Definição do Problema de Escalonamento de Tarefas e Alocação de Dados de WfCs em Nuvem

Neste capítulo, o modelo matemático para o problema de escalonamento de tarefas e alocação de arquivos é apresentado. Primeiramente serão discutidos os modelos relacionados à aplicação e ao ambiente de execução. Em seguida, a formulação matemática proposta é apresentada.

4.1 Descrição do Modelo Matemático

Como apresentado no Capítulo 2 (Subseção 2.1), um WfC é comumente definido como um DAG, no qual as tarefas são representadas como vértices e as dependências entre elas são definidas pelos arcos. Nesse contexto, um algoritmo de escalonamento mapeia a execução de tarefas interdependentes para recursos compartilhados (por exemplo, MVs) [66]. Geralmente, os métodos de escalonamento encontrados na literatura relacionada consideram que todos os arquivos estarão disponíveis em uma máquina, ou sincronizados entre todas. Diferente dessas abordagens, neste trabalho é proposto um novo modelo para o *workflow* e, baseado neste, o problema de Escalonamento de Tarefas e Alocação de Arquivos de Dados (ETAA) é apresentado.

O ETAA considera que determinar a máquina na qual os dados gerados serão alocados durante a execução do *workflow* é uma etapa crucial para o problema de escalonamento, pois permite diminuir não só o tempo de execução das tarefas, mas também os tempos de transferências. Além disso, essa abordagem também possibilita que o espaço de armazenamento seja levado em consideração, permitindo assim que cenários mais realísticos (isto

é, com espaço de armazenamento finito) sejam considerados. Além disso, motivado pela crescente migração de experimentos científicos para ambientes de nuvens computacionais, o ETAA foi formulado considerando as características desses ambientes.

4.1.1 Modelo da Aplicação e do Ambiente

O Problema de Escalonamento de Tarefas e Alocação de Arquivos de Dados (ETAA) considera uma classe de aplicações paralelas representadas por um DAG, denotado por $G = (V, A, a, \omega)$. Diferente dos trabalhos atuais, os arquivos de dados não são representados como arcos do grafo e sim como parte do conjunto de vértices, da mesma forma como as tarefas. Sendo assim, $V = N \cup D$ consiste no conjunto de tarefas $i \in N$ e de arquivos $d \in D$. Já o conjunto de arcos, que dá a relação de precedência entre tarefas e arquivos, é representado por A . Por fim, a_i é a quantidade de trabalho associada com a tarefa $i \in N$, e ω_k representa o custo associado ao arco $k \in A$.

O conjunto de tarefas predecessores imediatas da tarefa $i \in N$ é definido como $pred(i) = \{j \in N \mid \exists d \in D, \text{ tal que } (j, d) \in A \wedge (d, i) \in A\}$. De forma similar, o conjunto de sucessores imediatos é dado por $succ(i) = \{j \in N \mid \exists d \in D, \text{ tal que } (i, d) \in A \wedge (d, j) \in A\}$. No grafo, as tarefas são sempre precedidas e sucedidas por arquivos, como ilustrado na Figura 4.1a, na qual a tarefa tf_1 lê o arquivo d_1 e d_2 que são necessários para a sua execução e escreve d_3 , que é então lido pela tarefa tf_2 . Por fim, o arquivo d_4 é escrito por tf_2 .

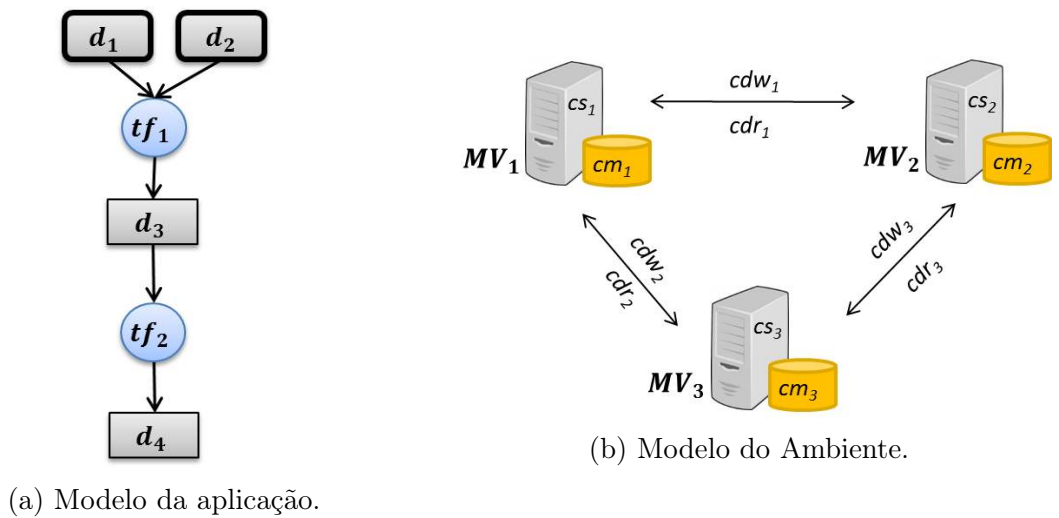


Figura 4.1: Exemplo dos modelos de ambiente e aplicação.

O modelo do ambiente, apresentado na Figura 4.1b, representa os recursos utilizados durante a execução das aplicações. Nessa representação, o conjunto de todas as MVs dis-

poníveis para execução e armazenamento é dado por M . Cada MV $j \in M$ tem capacidade de armazenamento cm_j e um valor computacional de *slowdown* definido como cs_j . O *slowdown* representa o grau de diferença da capacidade de processamento das diferentes MVs disponíveis, e é uma alternativa à representação do tempo de execução das tarefas por meio de matrizes [76]. Neste trabalho, o valor de *slowdown* é inversamente proporcional à capacidade de processamento cp_j de uma MV j . Porém, nos testes teóricos realizados com *workflows* sintéticos disponíveis na literatura, o valor foi definido em termos dos tempos de execução das tarefas. Essa diferença é explicada no Capítulo 6.

Seguindo a mesma ideia do *slowdown*, cd_l representa o custo de latência associado ao enlace l . Dois tipos de retardo de comunicação são propostos: um para a operação de escrita, cdw_l , e outro para a de leitura, cdr_l . Por conta disso, duas matrizes de comunicação são construídas, cada uma relacionada a uma das operações de comunicação. Dessa forma, o tempo de execução da tarefa $i \in N$ na MV $j \in M$ é dado por $t_{ij} = a_i \times cs_j$. Já o tempo de comunicação da tarefa $i \in N$ executando na MV $j \in M$, para escrever o dado $d \in D$ na MV $p \in M$, sendo j e p conectados por um enlace l , é dado por $\overleftarrow{t}_{djp} = \omega_{id} \times cdw_l$. De modo similar, o tempo de comunicação para leitura é dado por $\overrightarrow{t}_{djp} = \omega_{di} \times cdr_l$. A Figura 4.1b ilustra um ambiente composto por 3 MVs, contendo as seguintes informações: (i) capacidade de armazenamento cm , (ii) valor computacional de *slowdown* cs , (iii) o custo de comunicação para operação de escrita (cdw) e (iv) o custo de comunicação para operações de leitura (cdr).

A fim de simplificar o modelo, neste trabalho assume-se que um arquivo lido por uma tarefa será mantido na memória principal (volátil). Assim, a operação de leitura não necessita de espaço de armazenamento e não mantém cópia dos dados em diferentes máquinas.

4.1.2 Formulação Matemática

O ETAA pode ser formulado como um problema de programação inteira mista, nomeado IP-ETAA, como descrito a seguir. Primeiro, é necessário definir duas classes distintas de arquivos: os arquivos estáticos; e os arquivos dinâmicos. Os arquivos estáticos são aqueles que não são produzidos durante a execução do *workflow* e servem principalmente como entrada para as primeiras tarefas executadas (embora também possam ser utilizados pelas demais tarefas). Um arquivo estático é armazenado antes da execução do *workflow*, em uma das MVs cujo espaço de armazenamento seja suficiente. As alocações desses arquivos não são alteradas em nenhum momento da execução.

Tabela 4.1: Descrição dos dados e das variáveis utilizadas no modelo matemático.

| Dados | Descrição |
|---------------------------------|--|
| D_s | Conjunto de arquivos estáticos. |
| D_d | Conjunto de arquivos dinâmicos. |
| $D = D_s \cup D_d$ | Conjunto de arquivos. |
| $O(d)$ | Máquina de origem do arquivo estático $d \in D_s$. |
| $W(d)$ | Tamanho do arquivo $d \in D$. |
| N | Conjunto de tarefas. |
| a_i | Quantidade de trabalho da tarefa $i \in N$. |
| M | Conjunto de Mvs. |
| t_{ij} | Tempo de processamento da tarefa $i \in N$ na Mv $j \in M$. |
| \overrightarrow{t}_{djp} | Tempo gasto pela Mv $j \in M$ para ler o arquivo $d \in D$ armazenado na Mv $p \in M$ |
| \overleftarrow{t}_{djp} | Tempo gasto pela Mv $j \in M$ para escrever o arquivo $d \in D_d$ na Mv $p \in M$. |
| $\Delta_{in}(i) \subseteq D$ | Conjunto de arquivos de entrada necessários para a execução da tarefa $i \in N$. |
| $\Delta_{out}(i) \subseteq D_d$ | Conjnto de arquivos de saída gerados pela tarefa $i \in N$. |
| cm_j | Capacidade de armazenamento da Mv $j \in M$. |
| Variáveis | Descrição |
| x_{ijt} | Variável binária que indica se a tarefa $i \in N$ iniciou sua execução na Mv $j \in M$ no período $t \in T$ ou não. |
| $\overrightarrow{x}_{idjpt}$ | Variável binária que indica se a tarefa $i \in N$ executando na Mv $j \in M$ começou a ler o arquivo $d \in \Delta_{in}(i)$, que está armazenado na Mv $p \in M$, no período $t \in T$ ou não. |
| \overleftarrow{x}_{djpt} | Variável binária que indica se o arquivo $d \in D_d$ começou a ser escrito a partir da Mv $j \in M$ para a Mv $p \in M$ no periodo $t \in T$ ou não. |
| y_{djt} | Variável binária que indica se o arquivo $d \in D$ está armazenado na Mv $j \in M$ no período $t \in T$ ou não. |
| z_T | Variável contínua que indica o tempo total de execução (<i>makespan</i>) do <i>workflow</i> . |

Já os arquivos dinâmicos são gerados pelas tarefas como resultado dos processamentos realizados durante a execução do *workflow*. Os arquivos dinâmicos podem ser armazenados em qualquer MV disponível, desde que haja espaço de armazenamento suficiente. A

alocação de arquivos, definida como parte do problema deste trabalho, tem a classe de arquivo dinâmicos como foco e, portanto, é papel do escalonador definir a localização desses arquivos.

Sendo assim, $D = D_s \cup D_d$ é definido como sendo o conjunto de todos os arquivos, onde cada arquivo $d \in D$ tem tamanho $W(d)$ e pode ser estático $d \in D_s$ com uma máquina de origem $O(d) \in M$, ou dinâmico $d \in D_d$. Além disso, para cada tarefa $i \in N$ são associados um conjunto de arquivos de entrada $\Delta_{in}(i) \subseteq D$ necessário para sua execução, e um conjunto de arquivos de saída $\Delta_{out}(i) \subseteq D_d$. Por fim, um tempo T_M é definido como o tempo máximo de execução do *workflow*, sendo $T = \{1 \dots T_M\}$ o conjunto de intervalos de tempo de uma execução.

Como o objetivo do escalonamento é minimizar o tempo total de execução da aplicação, a função objetivo, definida em (4.1), minimiza o *makespan* (z_T) da aplicação.

$$\min z_T \tag{4.1}$$

A restrição (4.2) garante que cada tarefa seja executada. As restrições (4.3) e (4.4) certificam que todas as operações de leitura e escrita sejam realizadas, respectivamente. Já a inequação (4.5) garante que o dado $d \in \Delta_{out}(i)$ seja escrito apenas se a tarefa i tenha sido executada no tempo correto. Além disso, as restrições definidas em (4.6) asseguram que o dado d não possa ser escrito antes do tempo de processamento da tarefa i (responsável pela sua escrita). Note que ambas as restrições (4.5 e 4.6) trabalham em conjunto para garantir um tempo de escrita factível.

Sujeito a

$$\sum_{j \in M} \sum_{t \in T} x_{ij t} = 1, \quad \forall i \in N \tag{4.2}$$

$$\sum_{j, p \in M} \sum_{t \in T} \vec{x}_{idj p t} = 1, \quad \forall i \in N, \forall d \in \Delta_{in}(i) \tag{4.3}$$

$$\sum_{j, p \in M} \sum_{t \in T} \overleftarrow{x}_{dj p t} = 1, \quad \forall d \in D_d \tag{4.4}$$

$$\begin{aligned} \overleftarrow{x}_{djpt} &\leq x_{ij(t-t_{ij})}, & \forall d \in D_d, \forall j, p \in M, \\ \forall t &= (t_{ij} + 1) \cdots T_M \text{ tal que } d \in \Delta_{out}(i) \end{aligned} \quad (4.5)$$

$$\begin{aligned} \overleftarrow{x}_{djpt} &= 0 & \forall d \in D_d, \forall j, p \in M, \\ 1 &\leq t \leq t_{ij} \text{ tal que } d \in \Delta_{out}(i) \end{aligned} \quad (4.6)$$

A restrição definida em (4.7) assegura que uma tarefa só possa ser executada quando todas as operações de leitura estiverem concluídas. Além disso, a desigualdade (4.8) garante que apenas uma ação (execução, leitura ou escrita) possa ser realizada em cada período de tempo em cada MV. Ou seja, a MV não pode executar uma tarefa e escrever, ou ler, um dado ao mesmo tempo.

$$\begin{aligned} x_{ij t} &\leq \sum_{p \in M} \overrightarrow{x}_{idjp(t - \overrightarrow{t}_{djp})}, & \forall i \in N, \forall d \in \Delta_{in}(i), \forall j \in M, \\ \forall t &\in T, \text{ tal que } (t - \overrightarrow{t}_{djp}) \geq 1 \end{aligned} \quad (4.7)$$

$$\begin{aligned} \sum_{i \in N} \sum_{q=\max(1, t-t_{ij}+1)}^t x_{ijq} + \sum_{d \in D_d} \sum_{p \in M} \sum_{r=\max(1, t-\overleftarrow{t}_{djp}+1)}^t \overleftarrow{x}_{djpr} + \\ \sum_{i \in N} \sum_{d \in \Delta_{in}(i)} \sum_{p \in M} \sum_{r=\max(1, t-\overrightarrow{t}_{djp}+1)}^t \overrightarrow{x}_{idjpr} \leq 1, & \quad \forall j \in M, \forall t \in T \end{aligned} \quad (4.8)$$

A restrição (4.9) estabelece que não há arquivos dinâmicos no tempo inicial. Por outro lado, a restrição (4.10) garante que todos os arquivos estáticos estejam prontamente armazenados em suas máquinas de origem. Já as restrições (4.11) e (4.12) relacionam a variável de armazenamento y com as variáveis de escrita \overleftarrow{x} e de leitura \overrightarrow{x} , garantindo um processo viável de escrita e leitura. A restrição (4.12) garante que os arquivos serão lidos apenas se estes estiverem previamente armazenados em uma MV, e a restrição (4.11) assegura que um arquivo seja armazenado em MV apenas caso ele tenha sido produzido (escrito).

$$y_{dj1} = 0, \quad \forall d \in D_d, \forall j \in M \quad (4.9)$$

$$y_{djt} = 1, \quad \forall d \in D_s \mid j \in O(d), \forall t \in T \quad (4.10)$$

$$y_{dp(t+1)} \leq y_{dpt} + \sum_{j \in M} \overleftarrow{x}_{dj p(t - \overleftarrow{t}_{dj p})}, \quad \forall d \in D, \forall p \in M, \\ \forall t \in T, \text{ tal que } (t - \overrightarrow{t}_{dj p}) \geq 1 \quad (4.11)$$

A capacidade de armazenamento das MVs é estabelecida pela restrição (4.13). A restrição (4.14) relaciona a última operação de escrita com o tempo total de execução da aplicação (*makespan*). Note que no modelo da aplicação, uma tarefa sempre escreve ao menos um arquivo. Além disso, a restrição operacional (4.15) deve ser satisfeita: uma tarefa i pode iniciar um processo de leitura se todos os arquivos $d \in \Delta_{in}(i)$ estiverem disponíveis (isto é, se todos os arquivos $d \in (\Delta_{in}(i) \cap D_d)$ forem escritos). Por fim, as restrições restantes são as de integralidade e de não negatização.

$$\sum_{j \in M} \overrightarrow{x}_{idjpt} \leq y_{dpt}, \quad \forall i \in N, \forall d \in \Delta_{in}(i), \forall p \in M, \forall t \in T \quad (4.12)$$

$$\sum_{d \in D} y_{djt} W(d) \leq cm_j, \quad \forall j \in M, \forall t \in T \quad (4.13)$$

$$\overleftarrow{x}_{djpt} \cdot (t + \overleftarrow{t}_{dj p}) \leq z_T, \quad \forall d \in D_d, \forall j, p \in M, \forall t \in T \quad (4.14)$$

$$\overrightarrow{x}_{idjpt} \cdot |\Delta_{in}(i) \cap D_d| \leq \sum_{g \in \{\Delta_{in}(i) \cap D_d\}} \sum_{l, o \in M} \sum_{u=1}^{t - \overleftarrow{t}_{glo}} \overleftarrow{x}_{glou}, \quad \forall i \in N, \forall d \in \Delta_{in}(i), \\ \forall j, p \in M, \forall t \in T \quad (4.15)$$

Capítulo 5

Algoritmo Evolutivo Híbrido Para o Escalonamento de Tarefas e Alocação de Arquivos

Este capítulo apresenta o AEH-ETAA, um algoritmo evolutivo híbrido que soluciona o problema de escalonamento de WfCs em ambientes de nuvens computacionais conforme o modelo matemático proposto neste trabalho. O algoritmo é apresentado em função de seus principais métodos, discutidos em detalhes nas seções que seguem.

5.1 AEH-ETAA

Algoritmos Evolutivos (AE) [64] são métodos de otimização inspirados nos mecanismos de evolução biológica observados na natureza. No AE, cada cromossomo é um indivíduo de uma população e representa uma possível solução para o problema. A busca pela melhor solução é guiada por uma função de *fitness*, que atribui qualidade aos cromossomos. A cada iteração do algoritmo, novos indivíduos são gerados através da operação de *crossover* e a diversidade da população é obtida através da função de mutação. Neste trabalho, um Algoritmo Evolutivo Híbrido (AEH) que combina os operadores do AE, buscas locais e um método de *path relinking* [72] foi desenvolvido. De acordo com Moscato e Cotta [65], diferente do AE tradicional o AEH explora os conhecimentos disponíveis sobre o problema para atingir melhores resultados.

O Algoritmo Evolutivo Híbrido para Escalonamento de Tarefas e Alocação de Arquivos (AEH-ETAA) é uma metaheurística que escalona WfCs em ambientes de nuvens computacionais. Diferente da maioria das abordagens apresentadas na literatura, o AEH-

ETAA é responsável por determinar tanto a alocação das tarefas, quando a localização dos arquivos gerados durante a execução do *workflow*. Essa abordagem dá maior flexibilidade para o escalonador, pois permite que o tempo total de execução do *workflow* possa ser minimizado considerando tanto a execução das tarefas, como também os tempos gastos em transferências de arquivos.

O Algoritmo 1 representa o procedimento *Principal* do AEH-ETAA, que é responsável pela chamada dos demais procedimentos. Como pode ser visto, a metaheurística é composta pelas seguintes operações: (i) geração da população inicial (Subseção 5.1.2); (ii) buscas locais (Subseção 5.1.5); e (iii) *path relinking* (Subseção 5.1.6). Além disso, também é definido o procedimento *GeraPopulação* (apresentado no Algoritmo 2), que é responsável pela criação de novas soluções durante a execução da metaheurística. Esse procedimento é composto pelas operações de *crossover* e mutação (Subseções 5.1.3 e 5.1.4, respectivamente).

Algoritmo 1 Procedimento Principal

Entrada: Informações do *workflow* e do Ambiente.

Saída: Melhor solução encontrada (*best_global*).

```

1:  $P \leftarrow \text{populaçãoInicial}()$ 
2:  $\text{best\_global} \leftarrow \text{encontraBest}(P)$ 
3:  $\text{ConjElite} \leftarrow \emptyset$ 
4:  $i \leftarrow 0$ 
5: enquanto  $i \leq \text{MAX}$  faça
6:   se fazBuscasLocais? então
7:      $P \leftarrow \text{buscasLocais}(P)$  ▷ Algoritmos 4, 5, 6.
8:      $\text{best\_atual} \leftarrow \text{encontraBest}(P)$ 
9:     se  $\text{fitness}(\text{best\_atual})$  melhor que  $\text{fitness}(\text{best\_global})$  então
10:       $\text{best\_global} \leftarrow \text{best\_atual}$ 
11:      se  $\text{ConjElite} \neq \emptyset$  então
12:         $\text{best\_global} \leftarrow \text{pathRelinking}(\text{best\_global}, \text{conjElite})$  ▷ Algoritmo 7.
13:        se  $\forall \text{solução} \in \text{ConjElite}, \text{distância}(\text{best\_global}, \text{solução}) \geq \alpha$  então
14:           $\text{ConjElite} \leftarrow \text{ConjElite} \cup \text{best\_global}$ 
15:          se  $|\text{conjElite}| > \beta$  então
16:             $\text{removeCromossomo}(\text{ConjElite})$ 
17:       $P \leftarrow \text{geraPopulação}(P)$  ▷ Algoritmo 2
18:       $i = i + 1$ 
19: retorna  $\text{best\_global}$ 

```

Algoritmo 2 Procedimento GeraPopulação**Entrada:** População Anterior (P).**Saída:** Próxima População (P').

```

1:  $Filhos \leftarrow \emptyset$ 
2: para  $i \leftarrow 1$  a  $NUM\_OFFSPRING$  faça
3:    $p_1 \leftarrow torneio(P)$ 
4:    $p_2 \leftarrow torneio(P)$ 
5:    $novo \leftarrow \text{crossover}(p_1, p_2)$  ▷ Algoritmo 3
6:    $novo' \leftarrow mutação(novo)$ 
7:    $calculafitness(novo')$  ▷ Algoritmo 8
8:    $Filhos \leftarrow Filhos \cup novo'$ 
9:  $Soluções \leftarrow Filhos \cup P$ 
10:  $P' \leftarrow selecionaBests(Soluções)$  ▷ Componente elitista.
11: enquanto  $|P'| < POPULAÇÃO\_MAX$  faça
12:    $cromossomo \leftarrow torneio(Soluções)$ 
13:    $P' \leftarrow P' \cup cromossomo$ 
14:    $Soluções \leftarrow Soluções \setminus cromossomo$ 
15: retorna  $P'$ 

```

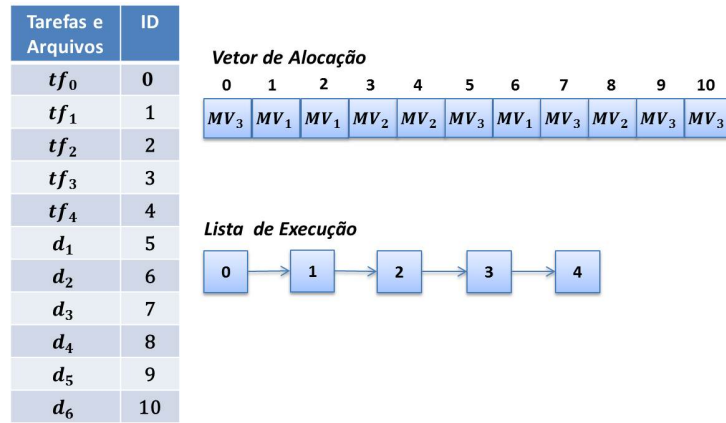
5.1.1 Representação do Cromossomo

Como apresentado no Capítulo 4, no problema de escalonamento de tarefas e alocação de arquivos de dados (ETAA), uma solução viável deve respeitar as ordens de precedência entre as tarefas, que são definidas através das relações de leitura e escrita presentes no *workflow*. Em outras palavras, uma tarefa tf_i só pode ser executada quando todos os seus arquivos de entrada estiverem disponíveis. Essa situação ocorre em dois cenários: (i) quando todas as tarefas predecessoras de tf_i estiverem finalizadas e, portanto, todos os seus arquivos de saída já estiverem disponíveis; ou (ii) quando tf_i tiver como entrada arquivos estáticos que, conforme a definição do modelo, estão disponíveis durante toda a execução do *workflow*. Por exemplo, na Figura 5.1b, a tarefa tf_2 depende do arquivo gerado por tf_0 . Sendo assim, tf_2 só poderá ser executada após a finalização de tf_0 .

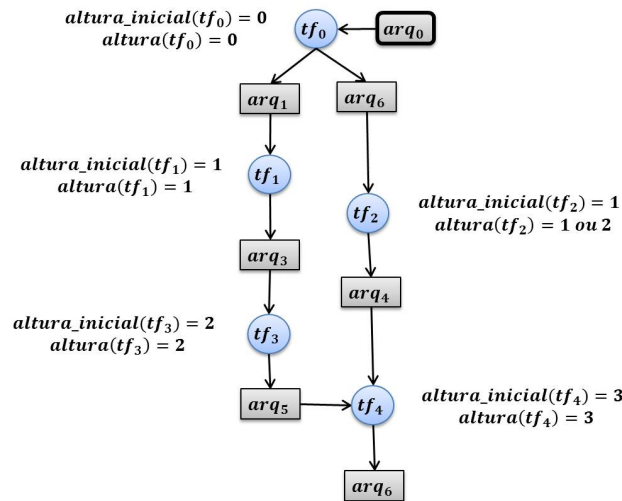
Neste trabalho, um cromossomo é composto por duas estruturas que representam: (i) a alocação das tarefas e dos dados e (ii) a ordem de execução das tarefas. Essa representação foi inspirada nas ideias apresentadas em Szabo *et al.* [79], e permite que os procedimentos de buscas locais e o cálculo do *fitness* sejam facilmente realizados. Como pode ser visto

na Figura 5.1a, a primeira estrutura é um vetor no qual os índices representam tarefas ou arquivos dinâmicos, e cada elemento representa a MV na qual a tarefa, ou o dado, foi alocado. Essa estrutura é chamada de *vetor de alocação*.

A representação da alocação de dados nesse vetor é uma importante diferença entre este trabalho e a literatura relacionada. É essa a estrutura que permite que a metaheurística proposta trate não apenas do problema da alocação de tarefas, mas também da alocação dos dados.



(a) Estruturas de dados para representação do cromossomo.



(b) Cálculo da altura das tarefas.

Figura 5.1: Codificação do cromossomo.

A ordem de execução das tarefas é representada por uma lista encadeada chamada de *lista de execução*, presente também na Figura 5.1a. Para definir a ordem de execução das tarefas, um valor de altura é associado a cada uma delas. Uma tarefa só pode ser

executada quando todas as tarefas de altura menores que a dela estiverem finalizadas. Já as tarefas com altura semelhante executam concorrentemente. O valor de altura é dado pelas equações 5.1 e 5.2, propostas por Tsujimura e Gen [84].

$$altura_inicial(tf_i) = \begin{cases} 0, & \text{se } pred(tf_i) = \emptyset \\ 1 + \max_{tf_j \in pred(tf_i)} altura_inicial(tf_j), & \text{caso contrário} \end{cases} \quad (5.1)$$

$$altura(tf_i) = \begin{cases} altura_inicial(tf_i), & \text{se } suc(tf_i) = \emptyset \\ rand \in [altura_inicial(tf_i), \min_{\forall tf_k \in suc(tf_i)} \{altura(tf_k)\} - 1], & \text{caso contrário} \end{cases} \quad (5.2)$$

A equação 5.1 atribui para cada tarefa uma altura inicial correspondente ao nível no qual a tarefa aparece no grafo que representa o *workflow*; a altura é zero quando não há predecessor, ou o máximo entre todas alturas iniciais de seus predecessores mais um. Essa equação permite que seja realizada uma ordenação topológica do grafo, já que basta definir grupos de tarefas de mesma altura (ou seja, que estão no mesmo nível) e, com base nos valores atribuídos, ordenar esses grupos de forma crescente. Porém, caso as tarefas sejam ordenadas apenas com base na altura inicial dada pela equação 5.1, o paralelismo entre diferentes grupos de tarefas não seria explorado. Por exemplo, na Figura 5.1b, a tarefa tf_2 pode ser executada em paralelo tanto com tf_1 quanto com tf_3 , pois não há relação de dependência entre essas tarefas. No entanto, a ordenação com base na altura inicial sempre atribuirá a altura de valor 1 para tf_2 , permitindo apenas o paralelismo entre tf_2 e tf_1 .

Para permitir que o paralelismo entre as tarefas seja totalmente explorado, um componente randômico é introduzido na equação 5.2. Esse componente permite que a tarefa possa ser executada em qualquer ordem entre a altura previamente calculada (altura inicial) e o valor mínimo da altura inicial de seus sucessores menos um. Seguindo o exemplo anterior, com o cálculo da Equação 5.2 a tarefa tf_2 pode ser associada tanto ao valor 1, sua altura inicial previamente calculada, quanto ao valor 2, a altura de seu sucessor menos 1. Sendo assim, no caso em que o valor da altura de tf_2 for igual a 2, a tarefa executaria em paralelo com tf_3 . Dessa forma, diferentes sequencias de tarefas podem ser geradas para construir a *lista de execução*. Essa característica é explorada pela função de *crossover* e nos procedimentos de busca local, apresentados nas Subseções 5.1.3 e 5.1.5, respectivamente.

5.1.2 População Inicial

A população inicial é gerada por meio de duas abordagens distintas: (i) 80% das soluções são geradas por heurísticas; e (ii) 20% são geradas randomicamente. As heurísticas MinMin [7] e HEFT [82] foram utilizadas para gerar a primeira parte da população inicial. Como essas heurísticas são determinísticas, isto é, as mesmas soluções são produzidas para as mesmas entradas, cada solução gerada passa por um processo de mutação que seleciona aleatoriamente e altera uma porcentagem dos genes que compõe o cromossomo. Dessa forma, diferentes soluções são geradas a partir de uma única solução.

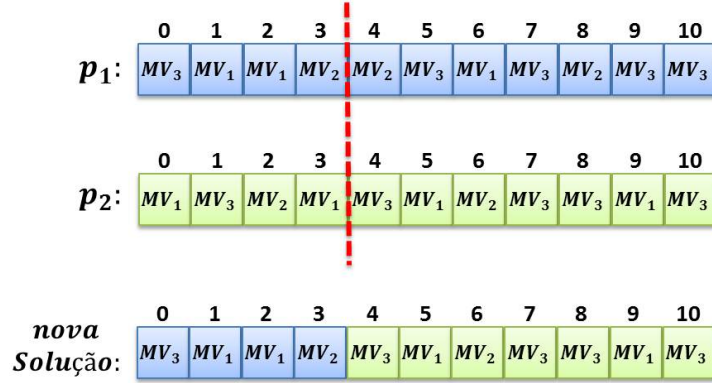
Em primeiro lugar, 40% das soluções são geradas a partir da solução produzida pelo MinMin, sendo que $\lambda\%$ dos genes no *vetor de alocação* são alterados randomicamente, onde λ varia de 5% até 90%. Em seguida, a mesma abordagem é utilizada para gerar os outros 40% de cromossomos, porém, aplicada a solução produzida pelo HEFT. A ideia por trás dessa abordagem é garantir que a população inicial tenha uma diversidade satisfatória e represente um bom ponto de partida para a busca.

Para gerar os cromossomos restantes (20% da população), cada tarefa e arquivo dinâmico é atribuído a uma MV randomicamente selecionada. A ordem de execução das tarefas é dada pela Equação 5.2, que é calculada para cada cromossomo gerado. Em ambas as abordagens, pode ocorrer de um arquivo dinâmico ser escalonado para uma MV que não tenha espaço de armazenamento suficiente. Quando isso ocorre, uma heurística proposta, chamada de *move-arquivos*, é executada para realocar os arquivos dinâmicos de modo que a restrição de armazenamento não seja violada. A heurística *move-arquivos* é apresentada na Subseção 5.1.7.

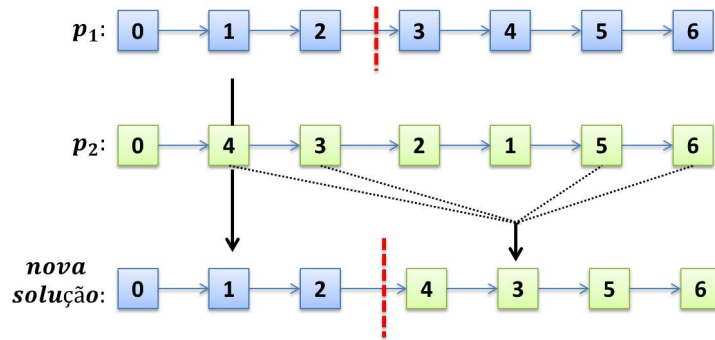
5.1.3 Operador de *Crossover* e Fase de Seleção

A operação de *crossover*, apresentada no Algoritmo 3, combina dois indivíduos da população para formar um novo *cromossomo*. No procedimento para gerar novas soluções (Algoritmo 2), dois cromossomos da população atual P são escolhidos utilizando o algoritmo de seleção por torneio [63]. Em seguida, um operador de *crossover* de recombinação em um ponto [46] é aplicado em ambas as estruturas que compõem os cromossomos. Inicialmente, dois pontos de corte são definidos nas linhas 1 e 2 do Algoritmo 3, um para o *vetor de alocação* (*corte1*) e outro para a *lista de execução* (*corte2*). Os genes à esquerda do *corte1* são copiados do *vetor de alocação* de $p1$ para as posições correspondentes da solução filho. Os genes restantes, posicionados à direita do corte, são copiados do *vetor*

de alocação de p_2 . Essa operação é também representada na Figura 5.2a. Como pode ser visto, a solução resultante contém uma parte das alocações originadas de p_1 (representadas pela cor azul) e outra parte de p_2 (cor verde).



(a) *Crossover realizado no vetor de alocação.*



(b) *Crossover realizado na lista de execução.*

Figura 5.2: Representação dos operadores de *crossover* utilizados pelo AEH-ETAA.

No caso do *crossover* na *lista de execução*, representado na Figura 5.2b e presente nas linhas 8 a 12 do Algoritmo 3, os genes do lado esquerdo do corte são copiados de p_1 para a solução filho (mantendo as mesmas posições). Já o restante dos genes da nova solução, são copiados da *lista de execução* de p_2 . Nesse último caso, as tarefas são verificadas uma a uma, e apenas as tarefas que ainda não estejam na solução filho são copiadas de p_2 . Essa verificação é necessária para garantir que não haja repetição de tarefas, e assegurar que as ordens de precedência entre as tarefas sejam respeitadas.

Algoritmo 3 Procedimento *Crossover***Entrada:** Cromossomos $p1$ e $p2$.**Saída:** Novo Cromossomo (*novo*).

```

1:  $corte1 \leftarrow rand(1, |vetorAlocação|)$ 
2:  $corte2 \leftarrow rand(1, |listaExecução|)$ 
3: para  $i \leftarrow 1$  a  $|vetorAlocação|$  faça
4:   se  $i < corte1$  então
5:      $novo.vetorAlocação[i] \leftarrow p1.vetorAlocação[i]$ 
6:   se não
7:      $novo.vetorAlocação[i] \leftarrow p2.vetorAlocação[i]$ 
8:   se  $i < corte2$  então
9:      $novo.listaExecução \leftarrow novo.listaExecução \cup p1.listaExecução(i)$ 
10: para  $i \leftarrow 1$  a  $|listaExecução|$  faça
11:   se  $p2.listaExecução(i) \notin novo.listaExecução$  então
12:      $novo.listaExecução \leftarrow novo.listaExecução \cup p2.listaExecução(i)$ 
13: retorna  $novo$ 

```

Na fase de seleção (apresentada no Algoritmo 2, linha 9 e 14), os cromossomos da próxima população são selecionados a partir do conjunto chamado *Soluções*, que é formado pelos novos cromossomos gerados e pela população anterior P . Primeiro, para garantir que as melhores soluções sempre estarão incluídas na população, uma seleção elitista é aplicada na linha 10, na qual 5% das melhores soluções são encontradas e adicionadas na próxima população P' . Em seguida, as demais soluções são escolhidas utilizando o algoritmo de seleção por torneio e também são incluídas em P' . Note que quando um cromossomo é selecionado ele é removido do conjunto *Soluções* para prevenir que um mesmo cromossomo seja incluído mais de uma vez na população P' .

5.1.4 Operador de Mutação

O operador de mutação é responsável pela diversificação da população, aumentando o espaço de busca e escapando dos chamados ótimos locais. Neste trabalho, o operador de mutação é executado apenas no *vetor de alocação*. Os testes empíricos mostraram que a mutação aplicada à *lista de execução* não apresenta melhorias significantivas em termos de qualidade da solução e, adicionalmente, aumenta o custo computacional do algoritmo.

A operação de mutação é chamada após o procedimento de *crossover*, para cada nova

solução gerada. Com base nas avaliações realizadas, fixou-se a probabilidade de mutação dos genes em 10%. Como pode ser visto na Figura 5.3, o operador altera a identificação da MV aleatoriamente e, com isso, gera uma nova solução.

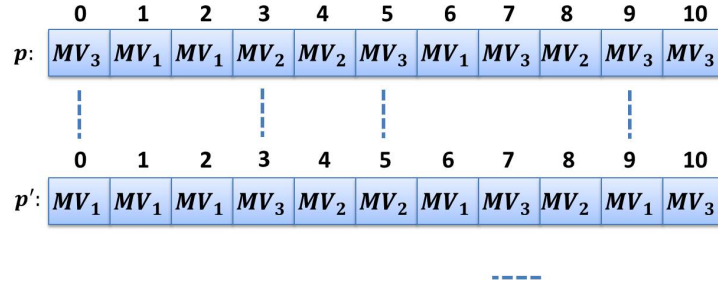


Figura 5.3: Operador de Mutação.

5.1.5 Procedimentos de buscas locais

Após a execução dos operadores de *crossover* e mutação para construir a nova população, buscas locais são realizadas a 15% das melhores soluções da população atual. As buscas locais ocorrem com uma probabilidade de 50%. A porcentagem de soluções nas quais as buscas são realizadas e a probabilidade de ocorrência foram definidas através de testes empíricos feitos para ajustar os parâmetros do algoritmo. Como pode ser visto no Algoritmo 1 linha 6, em cada iteração a probabilidade de busca local é verificada e, caso seja verdadeira, os procedimentos de buscas são executados.

Neste trabalho, três procedimentos de buscas locais foram definidos e são executados na seguinte ordem: (i) *troca-mv* (Algoritmo 4), que executa a troca de dois elementos no *vetor de alocação*; (ii) *troca-posição* (Algoritmo 5), que troca a posição de dois elementos de mesma altura na *lista de execução*; e (iii) *move-elemento* (Algoritmo 6), que move uma tarefa ou um arquivo para uma MV diferente. Cada busca local é executada até que ocorra uma melhora na solução (primeira-melhora) ou até que todas as combinações sejam testadas.

Algoritmo 4 Procedimento *troca-mv*

Entrada: Cromossomo p .**Saída:** Cromossomo p .

```

1:  $cópia \leftarrow p$ 
2: para  $i \leftarrow 1$  a  $|vetorAlocação|$  faça
3:   para  $j \leftarrow i + 1$  a  $|vetorAlocação|$  faça
4:     se  $p.vetorAlocação[i] \neq p.vetorAlocação[j]$  então
5:        $troca(p.vetorAlocação[i], p.vetorAlocação[j])$ 
6:       calculaFitness( $p$ ) ▷ Algoritmo 8.
7:       se  $fitness(p)$  melhor que  $fitness(cópia)$  então
8:         retorna  $p$ 
9:       se não
10:         $troca(p.vetorAlocação[i], p.vetorAlocação[j])$  ▷ Retorna ao estado
        anterior.
11: retorna  $cópia$ 

```

Algoritmo 5 Procedimento *troca-posição*

Entrada: Cromossomo p .**Saída:** Cromossomo p .

```

1:  $cópia \leftarrow p$ 
2: para  $i \leftarrow 1$  a  $|listaExecução|$  faça
3:    $tarefa_i \leftarrow p.listaExecução(i)$ 
4:   para  $j \leftarrow i + 1$  a  $|listaExecução|$  faça
5:      $tarefa_j \leftarrow p.listaExecução(j)$ 
6:     se  $altura(tarefa_i) = altura(tarefa_j)$  então
7:        $troca(p.listaExecução(i), p.listaExecução(j))$ 
8:       calculaFitness( $p$ ) ▷ Algoritmo 8.
9:       se  $fitness(p)$  melhor que  $fitness(cópia)$  então
10:        retorna  $p$ 
11:       se não
12:         $troca(p.listaExecução(i), p.listaExecução(j))$  ▷ Retorna ao estado
        anterior.
13:   se não
14:    interrompe ▷ Interrompe laço e retorna ao laço externo.
15: retorna  $cópia$ 

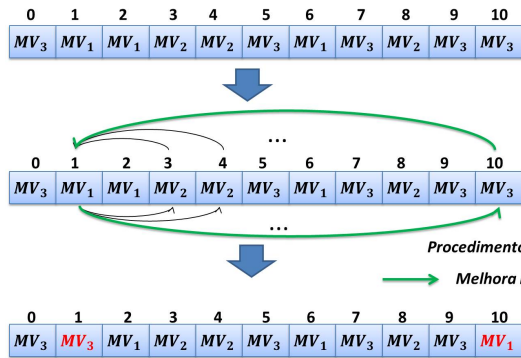
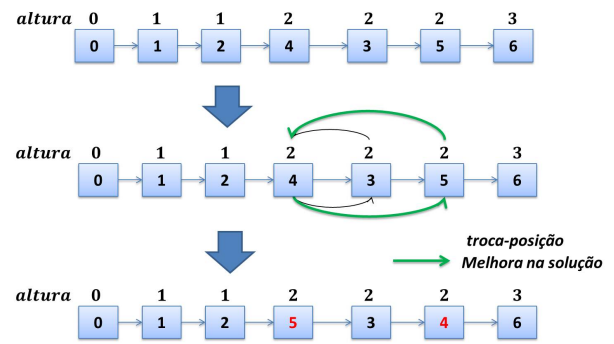
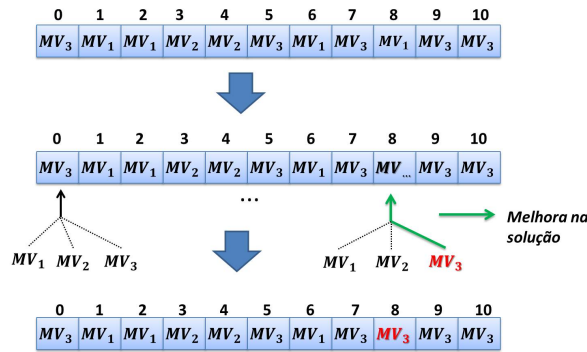
```

Algoritmo 6 Procedimento *move-elemento***Entrada:** Cromossomo p .**Saída:** Cromossomo p .

```

1:  $cópia \leftarrow p$ 
2: para  $i \leftarrow 1$  a  $|vetorAlocação|$  faça
3:    $MV\_atual \leftarrow p.vetorAlocação[i]$ 
4:   para  $MV\_prox \leftarrow 1$  a  $NÚMERO\_MV_s$  faça
5:     se  $MV\_atual \neq MV\_prox$  então
6:        $p.vetorAlocação[i] = MV\_prox$ 
7:       calculaFitness( $p$ ) ▷ Algoritmo 8.
8:       se  $fitness(p)$  melhor que  $fitness(cópia)$  então
9:         retorna  $p$ 
10:      se não
11:         $p.vetorAlocação[i] = MV\_atual$  ▷ Retorna ao estado anterior.
12: retorna  $cópia$ 

```

(a) Troca de máquinas no *vetor de alocação*.(b) Troca de tarefas na *lista de execução*.

(c) Move uma tarefa ou um arquivo de uma máquina para outra.

Figura 5.4: Procedimentos de Buscas Locais Implementados.

A Figura 5.4 apresenta os três tipos de buscas locais utilizadas no AEH-ETAA. A Figura 5.4a mostra a execução do procedimento *troca-mv* que, após testar vários movimentos, troca os elementos da posição 1 e 10 do *vetor de alocação*. A Figura 5.4b também executa várias trocas, até mover a ordem de execução da tarefa 4 e 5, que apresentam a mesma altura na *lista de execução*. Por fim, na Figura 5.4c, a tarefa ou o arquivo é movido da MV_2 para MV_3 .

5.1.6 Path Relinking

Path relinking é uma heurística capaz de gerar soluções intermediárias entre duas outras soluções. O operador começa com uma solução inicial e, passo a passo, insere elementos de uma solução guia. O objetivo é visitar novas soluções no caminho traçado entre uma solução e outra [37]. Portanto, durante a execução do *path relinking*, a distância entre as soluções diminui gradualmente.

Neste trabalho, a distância de um cromossomo p_i em relação a um cromossomo p_j é dada pela soma do número de elementos (tarefas ou arquivos) alocados a diferentes MVs, com o número de movimentos necessários para que a *lista de execução* de p_i seja semelhante a de p_j . A Figura 5.5 mostra a distância entre os cromossomos p_1 e p_2 . Como pode ser visto, o número de tarefas e arquivos alocados em diferentes MVs é 5 e apenas um movimento é necessário na *lista de execução*, portanto $dist(p_1, p_2) = 6$.

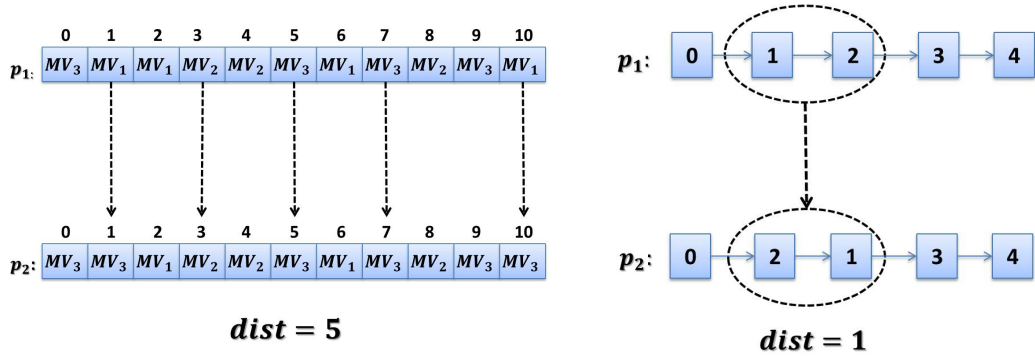


Figura 5.5: Cálculo da distância entre os cromossomos p_1 e p_2 , $dist(p_1, p_2) = 6$

O *path relinking* é aplicado quando uma melhor solução (*best*) é encontrada no procedimento principal (Algoritmo 1, linha 11). Como pode ser visto no Algoritmo 7, as soluções presentes no conjunto elite atuam como pontos iniciais da busca (chamados de *origem*). O conjunto elite é composto pelas melhores soluções cuja distância entre cada uma delas é maior que α , onde α é calculado como 25% do número de tarefas e arquivos do *workflow*. Esse parâmetro controla a diversidade do conjunto e, conseqüentemente, a

eficiência da busca. Além disso, para controlar o número de buscas efetuadas, um limite máximo de β cromossomos é definido para o conjunto elite, sendo que o valor de β é igual a metade do tamanho da população. Quando esse limite é alcançado, uma política de substituição *First-in First-out* (FIFO) é aplicada para substituir a solução mais antiga pela mais nova.

Algoritmo 7 Procedimento *PathRelinking*

Entrada: Cromossomo *destino* e Conjunto Elite *ConjElite*.

Saída: Melhor cromossomo encontrado (*best*).

```

1: best  $\leftarrow$  destino
2: para cont  $\leftarrow$  1 a |ConjElite| faça
3:   origem  $\leftarrow$  ConjElite(cont)
4:   para i  $\leftarrow$  1 a |origem.vetorAlocação| faça
5:     se origem.vetorAlocação[i]  $\neq$  destino.vetorAlocação[i] então
6:       origem.vetorAlocação[i]  $\leftarrow$  destino.vetorAlocação[i]
7:       calculaFitness(origem) ▷ Algoritmo 8.
8:       se fitness(origem) melhor que fitness(best) então
9:         best  $\leftarrow$  origem
10: retorna best

```

Como visto no Algoritmo 7, a busca se concentra no *vetor de alocação* das soluções. A cada iteração do algoritmo (linha 4), um elemento do *vetor de alocação* da solução *destino* é copiado para o vetor da solução origem. Na sequência, o valor de *fitness* da solução resultante é comparado com o melhor resultado obtido (armazenado em *best*) e, caso o valor seja melhor, o *best* é atualizado (linha 9). Essas etapas são repetidas para cada um dos cromossomos do conjunto elite e, por fim, a melhor solução encontrada é retornada para o procedimento principal.

5.1.7 Heurística *move-arquivos*

Quando um novo cromossomo é gerado sua viabilidade é avaliada. Se os arquivos alocados a uma MV excedem sua capacidade de armazenamento, a MV é considerada sobrecarregada e a heurística *move-arquivos* é executada.

A heurística contém as seguintes etapas:

1. Seleciona a MV mais sobrecarregada, MV_s .
2. Selecione a MV com maior quantidade de espaço livre, MV_d .
3. Move o arquivo de menor tamanho alocado em MV_s para MV_d .

Os passos acima são repetidos até que não haja MVs sobrecarregadas.

Baseado na hipótese de que as alocações definidas pela metaheurística correspondem as melhores alocações possíveis, parece razoável não modificar muito o cromossomo original, já que isso pode impactar o *makespan* resultante. Para minimizar esse provável impacto, os arquivos de menor tamanho são selecionados para serem movidos.

5.1.8 Função de *fitness*

A função de *fitness* pode ser definida como $f : s \rightarrow Z$, onde s é uma solução viável do espaço de busca e Z é um número inteiro que quantifica a qualidade da solução representada pelo cromossomo. Portanto, a função de *fitness* permite ordenar as soluções em termos de qualidade e direcionar a busca para as melhores soluções [80].

Neste trabalho, a qualidade de um indivíduo é dada pelo *makespan*, *i.e.*, seja p_i e p_j dois cromossomos, se $makespan(p_i) < makespan(p_j)$, então a solução dado por p_i é melhor que a dada por p_j . Ou seja, a função de *fitness* se resume ao cálculo do *makespan* associado a cada solução. O Algoritmo 8 efetua esse cálculo. Como pode ser visto, as informações da *lista de execução* e do *vetor de alocação* são utilizadas para estimar o tempo final de cada tarefa, sendo que antes do cálculo é verificado se a solução é viável, isto é, se os arquivos foram alocados em máquinas com espaço de armazenamento suficiente. Caso a solução não seja viável, a heurística *move_arquivos*, apresentada na subseção 5.1.7, é chamada e os arquivos são reorganizados. A heurística garante que a nova alocação sempre resultará em uma solução viável.

Algoritmo 8 Procedimento *CalculaFitness***Entrada:** Vetor de Alocação e Lista de Execução.**Saída:** Valor do *Makespan*.

```

1: se SolucaoNaoViavel(vetorAlocacao, listaExecucao) então
2:   |   move_arquivos(vetorAlocacao)   ▷ Procedimento apresentado na Subseção 5.1.7.
3:    $Q \leftarrow 0$ 
4:    $FT \leftarrow 0$ 
5:   para  $tf_i$  em listaExecução faça
6:     |    $MV_j \leftarrow \text{vetorAlocacao}[tf_i]$ 
7:     |    $t\_max\_pred \leftarrow \text{maxTempoFinalPred}(FT, tf_i)$ 
8:     |    $t\_inicial_i \leftarrow \max(t\_max\_pred, Q[MV_j])$ 
9:     |    $t\_final_i \leftarrow t\_inicial_i + \text{execucao}(tf_i, MV_j) + \text{leitura}(tf_i) + \text{escrita}(tf_i)$ 
10:    |    $Q[vm_j] \leftarrow t\_final_i$ 
11:    |    $FT[tf_i] \leftarrow t\_final_i$ 
12: retorna  $\max(FT)$ 

```

No Algoritmo 8, os vetores Q e FT são estruturas auxiliares usadas para manter o tempo calculado em cada etapa do algoritmo. O vetor Q é indexado pelo identificador da máquina e contém o tempo final da última tarefa executada na MV correspondente. Já o vetor FT armazena o tempo final de cada tarefa tf_i , utilizada como índice do vetor.

Os cálculos de tempo inicial e final de cada tarefa seguem o modelo apresentado na Subseção 4.1.1. Primeiramente, o tempo inicial da tarefa tf_i é computado como o máximo entre os tempos finais de seus predecessores imediatos (representado por t_max_pred) e o tempo final da última tarefa executada na MV_j (armazenado em $Q[MV_j]$), como apresentado na linha 8.

Em seguida, na linha 9, o tempo final da tarefa tf_i é calculado como a soma dos seguintes valores: (i) o tempo inicial de tf_i ($t_inicial_i$); (ii) o tempo de execução de tf_i quando executada na MV_j ; (iii) o tempo necessário para ler todos os dados de entrada de tf_i ; e (iv) o tempo necessário para escrever todos os arquivos gerados por tf_i . Finalmente, na linha 12, o algoritmo retorna o valor de *makespan*.

Capítulo 6

Resultados Experimentais

Esse capítulo apresenta os experimentos conduzidos com AEH-ETAA para avaliar a abordagem proposta. Foram conduzidos experimentos teóricos e práticos. As seções seguintes apresentam esses experimentos e discutem os resultados obtidos.

6.1 Experimentos Teóricos

O AEH-ETAA foi comparado em termos de qualidade de solução e tempo de execução com: (i) as soluções dadas pela formulação matemática IP-ETAA (resolvida com CPLEX 12.5.1); (ii) A heurística MinMin [7]; e (iii) a heurística HEFT [82]. As heurísticas MinMin e HEFT foram escolhidas pois executam em tempo polinomial, produzem escalonamentos eficientes e foram utilizados na comparação de diversos trabalhos relacionados [70, 59, 93].

Os testes foram realizados em um computador com processador Intel Core i7-3770 CPU 3,40 GHz com 12GB memória, executando o Ubuntu 14.04. O algoritmos AEH-ETAA, MinMin e HEFT foram implementados com C++, e compilados com o G++ versão 5.3.0.

Nesses experimentos, foram utilizados dois tipos de instâncias sintéticas. O primeiro tipo, usado na comparação entre o AEH-ETAA e IP-ETAA (Subseção 6.1.1), é composto por soluções geradas randomicamente, variando o número de tarefas, a quantidade de arquivos e o modelo de representação dos *workflows*. O segundo, utilizado na Subseção 6.1.2, foi gerado pelo *Workflow Generator* [20] (<https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator>), uma aplicação que gera *workflows* sintéticos utilizados na avaliação de diversos algoritmos de escalonamento. O *Workflow Generator* utiliza dados coletados de execuções reais de WfCs executados em *grids* e ambientes

de nuvem e, por conta disso, apresenta uma boa acurácia em relação às características reais dos *workflows*.

A metaheurística AEH-ETAA foi executada com uma população de 50 indivíduos e o critério de parada foi definido como 100 iterações sem melhorias.

6.1.1 Comparação do AEH-ETAA com a Abordagem Exata IP-ETAA

O AEH-ETAA e o IP-ETAA foram avaliados utilizando um conjunto de 12 instâncias divididas em 4 grupos de acordo com o seus tamanhos. Para cada grupo, três instâncias diferentes foram geradas variando o número de tarefas, arquivos, e a representação do *workflow*. Dois ambientes computacionais foram definidos para a simulação, com 3 e 5 MVs. O tamanho dos arquivos (MB) e tempo de execução das tarefas (segundos) foram definidos aleatoriamente entre 0,1 e 100. Uma capacidade de armazenamento de 10 GB foi definida e enlaces de 5, 10 e 15 MB/s foram utilizados. Por fim, o valor de *slowdown* das máquinas foi configurado entre 0,01 e 1. A Tabela 6.1 mostra a quantidade de tarefas e arquivos para cada instância gerada. Além disso, a estrutura básica do *workflow* de cada uma delas é apresentada de acordo com a classificação feita por Bharathi *et al.* [6]. A Figura 6.1 apresenta essa classificação e as estruturas dos *workflows*.

Tabela 6.1: Descrição das Instâncias Utilizadas na comparação com a formulação matemática.

| Referência | Estrut. Básica | Núm. tarefas | Núm. arquivos |
|------------|---|--------------|---------------|
| 5A | Processamento | 2 | 3 |
| 5B | Processamento e agregação de dados | 2 | 3 |
| 5C | Agregação de dados | 1 | 4 |
| 7A | Agregação e redistribuição de dados | 2 | 5 |
| 7B | Processamento | 3 | 4 |
| 7C | Agregação de dados e <i>pipeline</i> | 3 | 4 |
| 10A | Agregação de dados | 4 | 6 |
| 10B | Redistribuição de dados | 3 | 7 |
| 10C | Distribuição de dados e <i>pipeline</i> | 3 | 6 |
| 15A | Agregação e distribuição de dados | 5 | 10 |
| 15B | Distribuição de dados e <i>pipeline</i> | 4 | 11 |
| 15C | Agregação e distribuição de dados | 5 | 10 |

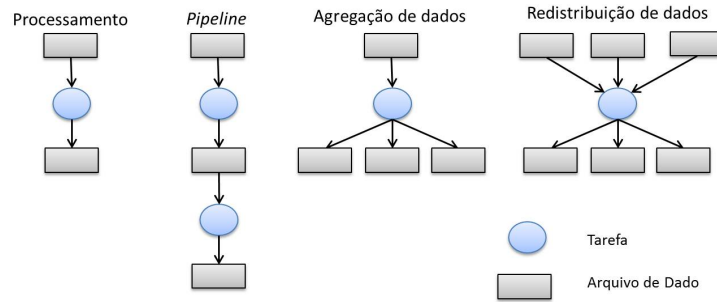


Figura 6.1: Estruturas básicas dos *workflows* (adaptado de [6]).

A Tabela 6.2 apresenta os resultados obtidos pela abordagem exata e pela metaheurística AEH-ETAA. A primeira coluna identifica as instâncias e nas duas colunas seguintes são apresentados os resultados obtidos pelo AEH-ETAA: *makespan* obtido e tempo de execução do algoritmo. Em seguida, nas duas próximas colunas, são apresentadas as informações referentes a abordagem exata. Por fim, a diferença percentual entre a solução dada pela metaheurística e pela abordagem exata é mostrada. O valor de *makespan* apresentados para AEH-ETAA representam a média de cinco execuções. Além disso, como os valores de desvio padrão foram iguais a zero para quase todas as instâncias, com exceção das instâncias 15B_m3, 15B_m5 e 15A_m5 que foram menos de 0,6%, essa informação não é apresentada.

Analisando a Tabela 6.2 pode-se observar que para todas as instâncias avaliadas o AEH-ETAA obteve soluções com uma diferença percentual baixa (média 1,1%), com tempos de execução significativamente menores em relação à formulação matemática resolvida com o CPLEX, em média 1,5s contra 12.640,3s, respectivamente. Além disso, o CPLEX não foi capaz de encontrar soluções viáveis para as instâncias 10A_m5 e 15C_m5 dentro de um limite de 24 horas. Ademais, na instância 15C_m3, o CPLEX não foi capaz de encontrar a solução ótima. Embora esses resultados sejam encorajadores, o AEH-ETAA também foi avaliado com outras heurísticas utilizando outras instâncias, como apresentado nas próximas seções.

Tabela 6.2: Resultados da metaheurística AEH-ETAA e da formulação matemática utilizando o CPLEX - tempos em segundos.

| Instâncias | AEH-ETAA | | Formulação Matemática | | Dif. Percentual |
|---------------------|-----------------|----------------|-----------------------|----------------|-----------------|
| | <i>Makespan</i> | Tempo de Exec. | <i>Makespan</i> | Tempo de Exec. | |
| 5A_m3 | 10,0 | 0,83 | 10,0 | 4,2 | 0,0 |
| 5B_m3 | 11,0 | 0,79 | 11,0 | 14,0 | 0,0 |
| 5C_m3 | 13,0 | 0,71 | 13,0 | 2,8 | 0,0 |
| 7A_m3 | 21,0 | 0,96 | 21,0 | 140,8 | 0,0 |
| 7B_m3 | 16,0 | 1,17 | 16,0 | 7,3 | 0,0 |
| 7C_m3 | 14,0 | 1,11 | 14,0 | 105,0 | 0,0 |
| 10A_m3 | 21,0 | 1,80 | 21,0 | 1644,6 | 0,0 |
| 10B_m3 | 12,0 | 1,74 | 12,0 | 21,4 | 0,0 |
| 10C_m3 | 21,0 | 1,22 | 21,0 | 316,7 | 0,0 |
| 15A_m3 | 16,0 | 2,47 | 16,0 | 2606,6 | 0,0 |
| 15B_m3 | 11,4 | 3,01 | 11,0 | 600,9 | 3,6 |
| 15C_m3 ^a | 19,2 | 2,42 | 94,0 | 86395,5 | — |
| 5A_m5 | 10,0 | 0,90 | 10,0 | 14,4 | 0,0 |
| 5B_m5 | 9,0 | 0,50 | 9,0 | 462,6 | 0,0 |
| 5C_m5 | 10,0 | 0,75 | 10,0 | 9,0 | 0,0 |
| 7A_m5 | 27,0 | 0,97 | 27,0 | 1836,4 | 0,0 |
| 7B_m5 | 26,0 | 1,26 | 26,0 | 21,0 | 0,0 |
| 7C_m5 | 16,0 | 1,09 | 16,0 | 2426,9 | 0,0 |
| 10A_m5 ^a | 25,0 | 1,58 | — | 86489,2 | — |
| 10B_m5 | 14,0 | 1,75 | 13,0 | 103,0 | 7,6 |
| 10C_m5 | 47,6 | 1,88 | 47,0 | 7353,5 | 1,2 |
| 15A_m5 | 16,0 | 2,64 | 16,0 | 15783,9 | 0,0 |
| 15B_m5 | 11,0 | 2,97 | 10,0 | 10764,3 | 10,0 |
| 15C_m5 ^a | 20,0 | 2,49 | — | 86243,0 | — |
| Média | 17,4 | 1,5 | 20,2 | 12640,3 | 1,1 |

^a O CPLEX não foi capaz de encontrar solução viável para a instância.

6.1.2 Comparação do AEH-ETAA com as Heurísticas HEFT e MinMin

Para comparar a performance e a qualidade dos resultados do AEH-ETAA com as heurísticas HEFT e MinMin, as instâncias sintéticas produzidas pelo *workflow generator* baseadas nos *workflows* Montage, Cybershake, Epigenomics e Inspiral [51] foram escolhidas. Cada instância tem as seguintes informações: (i) um DAG representando o *workflow*; (ii) o tempo de execução de cada tarefa em relação a uma máquina com capacidade de processamento conhecida; e (iii) o tamanho de cada arquivo de entrada e de saída. A Tabela 6.3 apresenta um resumo das principais características de cada um destes *workflows*. Para o ambiente de execução, foram consideradas as características (largura de banda, capacidade de armazenamento e processamento) das MVs oferecidas pela Amazon EC2. Foram utilizadas 4 MVs: 1 m3.medium (1 vCPU Intel Xeon E5-2670 v2); 1 m3.large (2 vCPU Intel Xeon E5-2670 v2); 1 m3.xlarge (4 vCPU Intel Xeon E5-2670 v2); 1 m3.2xlarge (2 vCPU Intel Xeon E5-2670 v2). Portanto, embora a avaliação seja teórica, os dados utilizados na descrição dos *workflows* e das MVs foram adquiridos com base em cenários reais.

Tabela 6.3: Atributos dos *workflows* utilizados na avaliação teórica entre o AEH-ETAA, HEFT e MinMin (Adaptado de [79]).

| Workflow | Tipo <i>I/O; Memória; CPU^a</i> | Núm. de tarefas por instâncias ^a | Núm. de arquivos dinâmicos e estáticos ^a |
|-------------|--|--|--|
| Cybershake | Baixo; Baixo; Médio | 30; 50; 100 | 49; 79; 154 |
| Epigenomics | Baixo; Médio; Alto | 24; 47; 79; 100; 127 | 38; 71; 119; 152; 192; |
| Montage | Alto; Baixo; Baixo | 25; 50; 100 | 38; 53; 93 |
| Inspiral | Baixo; Médio; Alto | 30; 50; 100 | 47; 77; 151 |

^a Dados referentes as diferentes instâncias utilizadas nos testes.

O tempo de execução das tarefas é definido como o produto do *tempo base* pelo valor de *slowdown* da MV que executará a tarefa. O valor de *slowdown* é definido como $\frac{P_B}{P_{mv_j}}$, onde P_B é a capacidade de processamento da máquina usada na estimativa do *tempo base*, e P_{mv_j} é a capacidade de processamento da máquina virtual mv_j usada no experimento. Portanto, o *slowdown* dá a variação da capacidade de processamento de uma MV em relação a máquina utilizada para calcular o *tempo base* da tarefa. Neste experimentos, o *tempo base* e a capacidade do processamento da máquina P_B foram obtidos a partir das instâncias dadas pelo *Workflow Generator*, enquanto P_{mv_j} é a capacidade de processamento da

máquina virtual oferecido pela Amazon EC2.

A Tabela 6.4 apresenta o *makespan* obtido pela média de 5 execuções, o desvio padrão relativo (RSD) e o tempo de execução do AEH-ETAA. Também são apresentados o *makespan* obtido com as heurísticas MinMin e HEFT. Como essas heurísticas são determinísticas, os resultados sempre serão os mesmos para entradas semelhantes, os valores de *makespan* não são médias de execuções e sim o valor obtido com uma única execução. Além disso, o tempo de execução do MinMin e do HEFT não são apresentados, pois eles são insignificantes (menos que 1 minuto).

Como pode ser visto na Tabela 6.4 a metaheurística AEH-ETAA supera ambos os algoritmos comparados. De fato, esses resultados são esperados, já que variações das soluções dadas pelo MinMin e pelo HEFT são utilizadas pela metaheurística na geração da população inicial. Porém, vale notar que houve uma melhora na qualidade dos valores de *makespan* em todos os casos de teste, com um tempo de execução aceitável (menos de 9 minutos no pior caso). O AEH-ETAA apresentou uma melhora média de 22,72% em relação ao MinMin e 11,15% em relação ao HEFT.

Tabela 6.4: Resultados da comparação entre AEH-ETAA, HEFT e MinMin.

| Inst. | MinMin | HEFT | AEH-ETAA | | |
|----------------|----------------------|----------------------|----------------------|------------|--------------------|
| | <i>Makesp. (min)</i> | <i>Makesp. (min)</i> | <i>Makesp. (min)</i> | <i>RSD</i> | <i>Exec. (min)</i> |
| Cybershake30 | 11,98 | 11,28 | 10,25 | 0,0019 | 0,39 |
| Cybershake50 | 14,88 | 16,65 | 12,46 | 0,0264 | 2,19 |
| Cybershake100 | 26,93 | 28,08 | 14,52 | 0,0261 | 8,11 |
| Genome.3510 | 530,88 | 469,38 | 444,91 | 0,0109 | 4,05 |
| Genome.7020 | 923,46 | 865,45 | 833,98 | 0,0004 | 6,79 |
| Epigenomics24 | 67,36 | 57,30 | 55,80 | 0,0000 | 0,03 |
| Epigenomics46 | 119,95 | 104,07 | 99,27 | 0,0140 | 0,48 |
| Epigenomics100 | 1,004,23 | 916,73 | 889,65 | 0,0001 | 4,02 |
| Montage25 | 1,81 | 1,16 | 0,95 | 0,0242 | 0,05 |
| Montage50 | 3,06 | 2,08 | 1,88 | 0,0036 | 0,10 |
| Montage100 | 5,31 | 4,66 | 3,93 | 0,0038 | 3,33 |
| Inspiral30 | 18,26 | 14,61 | 13,95 | 0,0014 | 0,17 |
| Inspiral50 | 29,96 | 23,36 | 22,41 | 0,0005 | 0,79 |
| Inspiral100 | 44,65 | 41,80 | 40,76 | 0,0036 | 1,55 |

Na Tabela 6.4 é possível verificar uma variação na melhora de *Makespan* entre os

diferentes *workflows*. Após uma análise da estrutura dos WfCs e do volume de dados transferidos com os diferentes planos de escalonamento (Tabela 6.5), é possível concluir que essas variações ocorrem principalmente devido ao volume de dados que é transferido entre as tarefas em cada *workflow*. De fato, *workflows* que produzem arquivos de grande tamanho apresentam um ganho de performance significativo quando o plano de escalonamento dado pelo AEH-ETAA é utilizado, já que este considera a alocação de dados em sua essência. Como pode ser visto na Tabela 6.5 o Cybershake_100, que apresenta a maior diminuição de *makespan*, teve a quantidade de transferências reduzidas em 42,5% e 36,6% quando comparado com o MinMin e o HEFT, respectivamente. Por outro lado, o *workflow* Inspiral_100, que apresenta a melhora de *makespan* menos significativa, teve a transferência reduzida em 23% e 6,4% quando comparado com MinMin e o HEFT, respectivamente. Esses dados corroboram para as análises apresentadas já que o Inspiral_100 tem vários arquivos de pequenos tamanhos, sendo que mesmo quando os planos de escalonamento do MinMin e do HEFT (que não consideram alocação de arquivos) são aplicados o volume de dados transferidos não apresenta um impacto considerável no *makespan* da aplicação.

Tabela 6.5: Volume de dados transferidos (em MBytes) com os planos de escalonamento dados pelos algoritmos MinMin, HEFT e AEH-ETAA.

| Instâncias | MinMin | HEFT | AEH-ETAA |
|-----------------|------------|------------|------------|
| Cybershake_30 | 3.228,59 | 3.460,30 | 2.270,97 |
| Cybershake_50 | 6.025,76 | 5.549,10 | 4.833,56 |
| Cybershake_100 | 14.335,00 | 12.931,10 | 8.234,62 |
| GENOME.d.3510 | 68.511,50 | 67.893,10 | 64.897,90 |
| GENOME.d.7020 | 111.237,00 | 106.490,00 | 106.480,00 |
| Epigenomics_24 | 8.344,20 | 7.958,83 | 7956,5 |
| Epigenomics_46 | 14.489,30 | 12.871,90 | 12.607,00 |
| Epigenomics_100 | 121.654,00 | 116.368,00 | 116.355,22 |
| Montage_25 | 247,10 | 117,44 | 80,86 |
| Montage_50 | 577,60 | 264,69 | 240,04 |
| Montage_100 | 764,21 | 729,59 | 593,47 |
| Inspiral_30 | 812,12 | 772,32 | 656,72 |
| Inspiral_50 | 1.383,53 | 1.104,66 | 1.104,58 |
| Inspiral_100 | 2.603,20 | 2.139,80 | 2.002,35 |

6.2 Avaliação do AEH-ETAA com *Workflow* Real

Esta subseção apresenta a avaliação do AEH-ETAA usando um *workflow* real de bioinformática. Para isso, o SGWfC SciCumulus foi escolhido para executar os escalonamentos. As seguintes seções apresentam o *workflow* utilizado como estudo de caso, uma breve descrição do SGWfC usado e as modificações necessárias para realizar os testes, a configuração do ambiente e os resultados obtidos.

6.2.1 SciPhy: Um *workflow* para análises filogenéticas

Vários tipos de experimentos de bioinformática são baseados nos resultados das análises filogenéticas, como os experimentos farmacológicos, desenvolvimento de novas drogas, *etc.* Uma análise filogenética tem o objetivo de produzir árvores filogenéticas, que são estruturas que mostram as relações evolucionárias inferidas entre genes homólogos representados no genoma da espécie divergente. Gerenciar experimentos filogenéticos não é uma tarefa trivial, pois eles são *compute-* e *data-intensive*. Esses experimentos são baseados em *pipelines* de programas científicos e, portanto, podem ser facilmente modelados como um WfC.

O SciPhy [67] é um dos *workflows* existentes para análises filogenéticas. O *workflow* executa análises com varredura de parâmetros, no qual o mesmo *workflow* é executado para cada um dos arquivos de entrada. O *workflow* é composto por quatro atividades principais: alinhamento de sequências múltiplas (MSA, do inglês *multiple sequence alignment*), conversor de sequências, busca pelo melhor modelo evolucionário, e construção das árvores filogenéticas. Essas atividades executam, respectivamente, as seguintes aplicações de bioinformática: programas MSA (MAFFT, Kalign, CLustalW, Muscle e ProbCons), ReadSeq, ModelGenerator, e RAxML.

Um grafo representando uma execução do SciPhy é apresentado na Figura 6.2. Cada círculo representa uma execução diferente da atividade em uma MV, e os retângulos representam os dados produzidos e consumidos. A primeira atividade do SciPhy (que é associada a tarefa tf_1, tf_5 e tf_n) constrói MSAs individuais utilizando um dos cinco programas de MSA disponíveis - ClustalW, Kalign, MAFFT, Muscle, e ProbCons - com os parâmetros padrão. Cada programa MSA recebe um arquivo multifasta como entrada (contendo uma sequência de DNA, RNA e aminoácidos - Mf1, Mf2 e Mfm), e produz um MSA como saída (Ms1, Ms2 e Msm). Cada MSA é então convertido para o formato phylip pela segunda atividade (que é associada as tarefas tf_2, tf_6 e tf_{n+1}) para serem

futuramente processadas. Após a conversão para o formato phylip (Rs1, Rs2 e Rsm), os arquivos são testados na terceira atividade para encontrar o melhor modelo evolutivo usando o ModelGenerator (associado as tarefas tf_3, tf_6, tf_{n+2}). Em seguida, o MSA individual, o MSA convertido e o modelo evolutivo são utilizados na quarta atividade (associada às tarefas tf_4, tf_8 e tf_{n+3}) para gerar as árvores filogenéticas (T1, T2 e Tm) usando o RAxML com 100 replicações [78]. Consequentemente, podem ser obtidas várias árvores diferentes para cada um dos diferentes programas MSA e para os vários arquivos multifasta de entrada. Nos experimentos apresentados neste trabalho o MAFFT foi empregado como método de MSA.

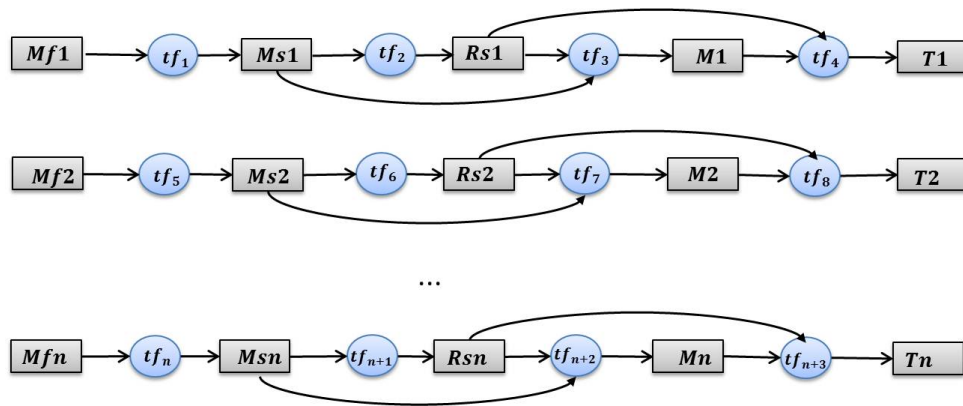


Figura 6.2: Um grafo representando uma única execução do Sciphy.

Como o objetivo é executar uma varredura de parâmetros no Sciphy, cada uma das atividades são executadas com diferentes arquivos de entrada contendo várias sequencias (arquivos multifasta) e, portanto, gerando diferentes tarefas. Cada uma dessas tarefas podem ser executadas em paralelo. Para mais informações sobre o Sciphy, consulte Ocanã *et al.* [67].

6.2.2 SGWfC SciCumulus

O SGWfC SciCumulus oferece apoio para dois tipos de paralelismo: varredura de parâmetros [88]; e paralelismo de dados [18]. O gerenciador atua na distribuição, controle e monitoramento de execuções paralelas de WfCs executados em ambientes de nuvem, e é composto por 4 módulos:

- I Camada Cliente: Os componentes desse módulo enviam os *workflows* para serem executados na nuvem.
- II Camada de Distribuição: Gera e gerencia atividades executáveis (tarefas) em uma ou mais MVs instanciadas em um ou mais ambientes de nuvem.

III Camada de Execução: É responsável pela execução dos programas invocados pelas atividade do *workflow*, pela geração dos dados e pelo armazenamento dos dados de proveniência.

IV Camada de Dado: Essa camada é base de dados de proveniência, que armazena todos os dados de proveniência consumidos e gerados pela execução paralela do *workflow*.

O SciCumulus oferece uma infraestrutura computacional mínima para suportar o paralelismo de *workflows* com gerenciamento de dados de proveniência. Para os experimentos apresentados neste trabalho o escalonados do SciCumulus foi modificado. Na versão atual do SciCumulus, o escalonador é baseado em uma abordagem gulosa e dinâmica, chamada aqui de Greedy-SC [22]. No Greedy-SC sempre que uma máquina fica ociosa o escalonador é chamado para decidir qual será a melhor tarefa a ser executada naquela máquina. Essa é uma abordagem simples que apresenta bons resultados, especialmente quando há variações de performance nas MVs.

Na versão do SciCumulus usada neste experimento, o AEH-ETAA é invocado antes da execução do *workflow*, e são fornecidas as informações das tarefas, dos arquivos e das MVs que formam o *cluster* virtual na nuvem. Todos esses dados são consultados e resgatados dos dados de proveniência [36]. O AEH-ETAA cria então um plano de escalonamento que é retornado para o SciCumulus. O plano de escalonamento é carregado para uma tabela na base de dados, que é consultada pelo escalonar sempre que uma MV fica ociosa. Para mais informações sobre o SciCumulus consulte Oliveira *et al.* [24].

6.2.3 Configuração do Ambiente

Os experimentos práticos deste trabalho foram realizados no serviço de nuvem Amazon EC2. O Amazon EC2 é um dos provedores de nuvem comercial mais popular da atualidade, e inúmeras aplicações científicas e comerciais são executadas pelo serviço diariamente. No experimento apresentado neste trabalho foram considerados 2 tipos de MVs oferecidas pelo serviço: m3.small (1 VCpu e 3,75 GB memória RAM) e m3.2xlarge (8 VCpu e 30 GB de memória). As MVs foram definidas conforme as recomendações do GraspCC [19], uma técnica baseada do GRASP que dimensiona o ambiente de nuvem para WfCs. Segundo o algoritmo de dimensionamento, um ambiente composto por uma m3.small e duas m3.2xlarge seria suficiente para a execução do *workflow* SciPhy.

As MVs utilizadas têm como sistema operacional a distribuição Gnu/Linux Cent OS 7 (64 bits), e foram configuradas com os softwares e as bibliotecas necessárias para a

execução das aplicações de bioinformática. As máquinas foram instanciadas na região de US East - N. Virginia e seguem as regras de preço desta localidade. Além disso, as execuções do SciPhy foram realizadas em um único site da Amazon EC2.

6.2.4 Configuração do Experimento

Para executar o SciPhy, foram utilizados como entrada um conjunto de arquivos multifasta contendo sequências de proteínas extraídas do RefSeq versão 75 - 14 de março, 2016. Esse conjunto de dados é formado por 25 arquivos multifasta de aminoácidos, e cada arquivo é constituído por uma média de 20 sequências. Uma vez baixados, cada arquivo multifasta é armazenado em uma das MVs m3.2xlarge. A fim de gerar arquivos de grande tamanho para o experimento, cada arquivo multifasta teve o seu tamanho artificialmente aumentado. Embora do ponto de vista biológico os resultados produzidos a partir desses arquivos não sejam úteis, pois as sequências foram replicadas inúmeras vezes, a variação no tamanho dos arquivos é fundamental para avaliar a performance dos algoritmos. As seguintes versões dos programas foram utilizadas: MAFFT 6.857, ReadSeq 2.1.22, ModelGenerator 0.85 e RAxML 7.2.8-ALPHA. Para o ModelGenerator, foram considerados os seguintes modelos evolutivos: BLOSUM62, CPREV, HTT, WAG, e RtREV. Cada execução do SciPhy com essa configuração gerou 100 tarefas executáveis e produziu 125 arquivos.

6.2.5 Resultados Experimentais do SciPhy

Nesta subseção são apresentados os tempos de execução do SciPhy utilizando os planos de execução gerados pelos algoritmos Greedy-SC, MinMin, HEFT e AEH-ETAA. Os resultados são apresentados na Figura 6.3. Para obter resultados estatísticos significantes, o SciPhy foi executado 5 vezes para cada uma das abordagens. Como apresentado na Figura 6.3, o tempo médio de execução do SciPhy foi de 273,1 minutos para o Greedy-SC, 225,6 para o MinMin, 214,5 para o HEFT e 198,2 minutos para AEH-ETAA. Isso representa aproximadamente 27,4% de melhora quando utilizado AEH-ETAA em comparação com o Greedy-SC, 11,7% em comparação com o MinMin e 8,1% em comparação com o HEFT. Essa diferença é devida a distribuição dos dados feita pelo AEH-ETAA, que alocou os arquivos utilizados por várias tarefas nas MVs com as maiores larguras de banda, diminuindo assim os tempos de transferências.

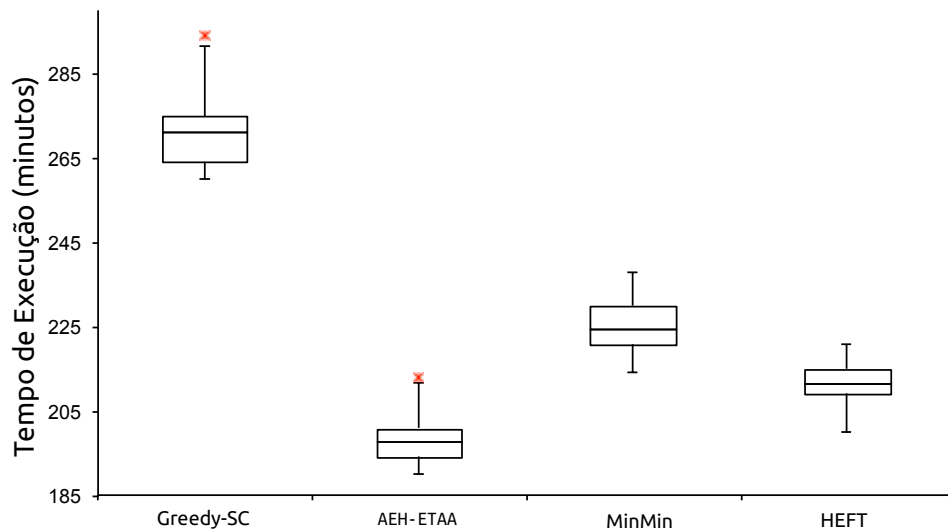


Figura 6.3: Diagrama de caixa comparando as execuções do SciPhy utilizando os algoritmos de escalonamento AEH-ETAA, Greedy-SC, HEFT e MinMin.

A Tabela 6.6, apresenta a média dos valores de *makespan* estimados pelos algoritmos estáticos (MinMin, HEFT e AEH-ETAA) em comparação com os valores reais obtidos na execução. Note que os algoritmos estimaram tempos menores em relação aos tempos reais obtidos, representando uma diferença média de mais ou menos 26%. Isso ocorre pois tanto os *overheads* oriundos do SciCumulus (como o tempo de consulta à base de dados, por exemplo), quanto as variações de performance das MVs, não são consideradas pelos algoritmos estáticos. Os valores de tempo estimados do algoritmo Greedy-SC não são apresentados, pois como se trata de um algoritmo dinâmico, não há uma estimativa prévia dos tempos de execução do *workflow* escalonado pelo Greedy-SC.

Tabela 6.6: Comparação dos valores estimados e obtidos pelos algoritmos de escalonamento.

| Algoritmo | Tempo Estimado | Tempo obtido |
|-----------|----------------|--------------|
| Greedy-SC | — | 273,1 |
| MinMin | 219,31 | 224,6 |
| HEFT | 168,10 | 214,5 |
| AEH-ETAA | 133,36 | 198,2 |

Capítulo 7

Conclusões e Trabalhos Futuros

Experimentos científicos computacionalmente intensivos geralmente produzem um grande volume de dados e apresentam uma longa duração, na qual várias execuções, utilizando diferentes parâmetros e dados de entrada, são necessárias para obter resultados. Esses experimentos são comumente modelados como *workflows* científicos, que são compostos por uma cadeia de atividades que representam diferentes conjuntos de tarefas. Cada tarefa é associada a uma execução de uma atividade para uma porção especificada dos dados ou para um conjunto de valores de parâmetros. Essas tarefas são executadas em paralelo em ambientes HPC para produzir resultados a um tempo aceitável.

No entanto, não é trivial gerenciar execuções paralelas de WfCs, particularmente na nuvem. Uma questão importante a ser considerada na execução em ambientes de nuvem é como escalonar as várias tarefas a um conjunto de MVs heterogêneas. O escalonamento de tarefas é um problema NP-Difícil bem conhecido, mesmo na sua forma simples. Além disso, para escalonar as tarefas para MVs várias características devem ser levadas em consideração, tais como a capacidade de processamento, capacidade de armazenamento e o impacto das transferências de dados no tempo total de execução. Este trabalho parte da ideia de que o problema de escalonamento de tarefas e o problema de alocação de dados não são independentes e, portanto, devem ser analisados de forma conjunta pela abordagem de escalonamento.

Neste trabalho, um novo modelo de representação de *workflow* foi proposto, junto a uma formulação matemática do problema de escalonamento de dados e distribuição de tarefas e um algoritmo de escalonamento evolutivo híbrido chamado AEH-ETAA que considera a heterogeneidade das MVs no ambiente de nuvem (diferentes larguras de banda, capacidade de processamento e armazenamento), a distribuição dos dados e as restrições, todos juntos na mesma solução, ou seja, as tarefas e os arquivos são escalonados juntos

pelo SGWfC.

O AEH-ETAA foi avaliado utilizando *workflows* sintéticos e reais utilizando o SGWfCs SciCumulus. A avaliação de performance do AEH-ETAA em comparação com a solução exata, mostrou que, para todas as instâncias, o AEH-ETAA obtém boas soluções com pequenas diferenças percentuais, na média 2,6%. Além disso, o algoritmo leva um tempo significativamente menor quando comparado à formulação matemática solucionada com o CPLEX, em média 1,5s contra 1942,7s, respectivamente. Na execução real, foi utilizado o SciPhy como estudo de caso para a comparação do AEH-ETAA com o algoritmo guloso do SciCumulus Greedy-SC, MinMin e HEFT. O tempo de execução do SciPhy quando executado com o plano de escalonamento dado pelo AEH-ETAA foi em torno de 27,4% menor que o dado pelo algoritmo guloso, 11,7% menor que o MinMin e 8,1% menor que o HEFT.

Como trabalhos futuros, a abordagem proposta será incrementado com técnicas de tolerância a falhas que realizarão *backups* das tarefas, e possibilitarão a recuperação parcial, ou total, dos processos caso haja interrupções devido a ocorrência de falhas no ambiente. Além disso, também serão adotadas técnicas de redimensionamento do ambiente que serão capazes de alocar e desalocar as MVs durante a execução do *workflow*. As técnicas de redimensionamento são importantes nos ambientes de nuvens, pois possibilitam diminuir o custo financeiro associado ao ambiente. Outro trabalho interessante é a inclusão de outros objetivos (abordagem multiobjetivo) no modelo proposto, como minimizar os custos financeiros e o *makespan*, respeitando o tempo limite de execução definido pelo usuário e um valor máximo de orçamento.

Referências

- [1] 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. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.
- [2] ALTINTAS, I.; LUDAESCHER, B.; KLASKY, S.; VOUK, M. A. Introduction to scientific workflow management and the kepler system. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 2006), SC '06, ACM.
- [3] AMAZON. Amazon elastic compute cloud (amazon ec2). <https://aws.amazon.com/pt/ec2/>.
- [4] ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. A view of cloud computing. *Commun. ACM* 53, 4 (Apr. 2010), 50–58.
- [5] BARKER, A.; VAN HEMERT, J. *Scientific Workflow: A Survey and Research Directions*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 746–753.
- [6] 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.
- [7] BLYTHE, J.; JAIN, S.; DEELMAN, E.; GIL, Y.; VAHI, K.; MANDAL, A.; KENNEDY, K. Task scheduling strategies for workflow-based applications in grids. In *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005*. (May 2005), vol. 2, pp. 759–767 Vol. 2.
- [8] BOUSSAÏD, I.; LEPAGNOT, J.; SIARRY, P. A survey on optimization metaheuristics. *Information Sciences* 237 (2013), 82 – 117. Prediction, Control and Diagnosis using Advanced Neural Computations.
- [9] BRODER, A.; GARCIA-PUEYO, L.; JOSIFOVSKI, V.; VASSILVITSKII, S.; VENKATESAN, S. Scalable k-means by ranked retrieval. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining* (New York, NY, USA, 2014), WSDM '14, ACM, pp. 233–242.
- [10] BROWN, D. A.; BRADY, P. R.; DIETZ, A.; CAO, J.; JOHNSON, B.; McNABB, J. *A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis*. Springer London, London, 2007, pp. 39–59.

- [11] BRYK, P.; MALAWSKI, M.; JUVE, G.; DEELMAN, E. Storage-aware algorithms for scheduling of workflow ensembles in clouds. *Journal of Grid Computing* (2015), 1–20.
- [12] BYUN, E.-K.; KEE, Y.-S.; KIM, J.-S.; MAENG, S. Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems* 27, 8 (2011), 1011 – 1026.
- [13] CALLAHAN, S. P.; FREIRE, J.; SANTOS, E.; SCHEIDEGGER, C. E.; SILVA, C. T.; VO, H. T. VisTrails: Visualization Meets Data Management. In *SIGMOD International Conference on Management of Data* (New York, NY, USA, 2006), SIGMOD '06, ACM, pp. 745–747.
- [14] ÇATALYÜREK, U. V.; KAYA, K.; UÇAR, B. Integrated data placement and task assignment for scientific workflows in clouds. In *Proceedings of the Fourth International Workshop on Data-intensive Distributed Computing* (New York, NY, USA, 2011), DIDC '11, ACM, pp. 45–54.
- [15] 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, 2 (2013), 157–164.
- [16] 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, 1 (Jan 2009), 29–43.
- [17] CHOPRA, N.; SINGH, S. Heft based workflow scheduling algorithm for cost optimization within deadline in hybrid clouds. In *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)* (July 2013), pp. 1–6.
- [18] COUTINHO, F.; OGASAWARA, E.; DE OLIVEIRA, D.; BRAGANHOLO, V.; LIMA, A. A. B.; DÁVILA, A. M. R.; MATTOSO, M. Data parallelism in bioinformatics workflows using hydra. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (New York, NY, USA, 2010), HPDC '10, ACM, pp. 507–515.
- [19] COUTINHO, R. D. C.; DRUMMOND, L. M.; FROTA, Y.; DE OLIVEIRA, D. Optimizing virtual machine allocation for parallel scientific workflows in federated clouds. *Future Generation Computer Systems* 46 (2015), 51 – 68.
- [20] D. SILVA, R. F.; CHEN, W.; JUVE, G.; VAHI, K.; DEELMAN, E. Community resources for enabling research in distributed scientific workflows. In *e-Science (e-Science), 2014 IEEE 10th International Conference on* (Oct 2014), vol. 1, pp. 177–184.
- [21] DE OLIVEIRA, D.; BAIÃO, F. A.; MATTOSO, M. Towards a Taxonomy for Cloud Computing from an e-Science Perspective. In *Cloud Computing*, N. Antonopoulos and L. Gillam, Eds., Computer Communications and Networks. Springer London, 2010, pp. 47–62.

- [22] 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, 3 (2012), 521–552.
- [23] DE OLIVEIRA, D.; OGASAWARA, E. Is cloud computing the solution for brazilian researchers? *International Journal of Computer Applications* 6, 8 (2010), 19–23.
- [24] DE OLIVEIRA, D.; OGASAWARA, E.; BAIÃO, F.; MATTOSO, M. Scicumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows. In *3rd International Conference on Cloud Computing* (2010), pp. 378–385.
- [25] DEELMAN, E.; CALLAGHAN, S.; FIELD, E.; FRANCOEUR, H.; GRAVES, R.; GUPTA, N.; GUPTA, V.; JORDAN, T. H.; KESSELMAN, C.; MAECHLING, P.; MEHRINGER, J.; MEHTA, G.; OKAYA, D.; VAHI, K.; ZHAO, L. Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example. In *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)* (Dec 2006), pp. 14–14.
- [26] DEELMAN, E.; GANNON, D.; SHIELDS, M.; TAYLOR, I. Workflows and e-science: An overview of workflow system features and capabilities. *Future Gener. Comput. Syst.* 25, 5 (May 2009), 528–540.
- [27] DEELMAN, E.; SINGH, G.; LIVNY, M.; BERRIMAN, B.; GOOD, J. The cost of doing science on the cloud: The montage example. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (Piscataway, NJ, USA, 2008), SC '08, IEEE Press, pp. 50:1–50:12.
- [28] DEELMAN, E.; SINGH, G.; SU, M.-H.; BLYTHE, J.; GIL, Y.; KESSELMAN, C.; MEHTA, G.; VAHI, K.; BERRIMAN, G. B.; GOOD, J.; LAITY, A. C.; JACOB, J. C.; KATZ, D. S. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming* 13, 3 (2005), 219–237.
- [29] DIKAIKOS, M. D.; KATSAROS, D.; MEHRA, P.; PALLIS, G.; VAKALI, A. Cloud computing: Distributed internet computing for it and scientific research. *IEEE Internet Computing* 13, 5 (Sept 2009), 10–13.
- [30] DONG, F.; AKL, S. G. Scheduling algorithms for grid computing: State of the art and open problems. Tech. rep., Technical report, 2006.
- [31] DORIGO, M.; DI CARO, G. Ant colony optimization: a new meta-heuristic. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on* (1999), vol. 2, p. 1477 Vol. 2.
- [32] DURILLO, J. J.; FARD, H. M.; PRODAN, R. Moheft: A multi-objective list-based method for workflow scheduling. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings* (Dec 2012), pp. 185–192.
- [33] DURILLO, J. J.; NAE, V.; PRODAN, R. Multi-objective energy-efficient workflow scheduling using list-based heuristics. *Future Generation Computer Systems* 36 (2014), 221 – 236. Special Section: Intelligent Big Data ProcessingSpecial Section: Behavior Data Security Issues in Network Information PropagationSpecial Section:

- Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications.
- [34] FAKHFAKH, F.; KACEM, H. H.; KACEM, A. H. Workflow scheduling in cloud computing: A survey. In *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations* (Sept 2014), pp. 372–378.
 - [35] FC, F.; A, C. Competitive science: is competition ruining science? *Infect Immun.* 83, 4 (2015), 1229–1233.
 - [36] FREIRE, J.; KOOP, D.; SANTOS, E.; SILVA, C. T. Provenance for Computational Tasks: A Survey. *Computing in Science and Engg.* 10, 3 (May 2008), 11–21.
 - [37] GLOVER, F.; LAGUNA, M.; MARTI, R. Fundamentals of scatter search and path relinking. *Control and Cybernetics Vol. 29, no 3* (2000), 653–684.
 - [38] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
 - [39] GOOGLE. Gmail. <https://www.google.com/gmail/about/>.
 - [40] GOOGLE. Google app engine. <https://cloud.google.com/appengine/>.
 - [41] GOOGLE. Google cloud platform. <https://cloud.google.com/>.
 - [42] GOOGLE. Google docs. <https://www.google.com/docs/about/>.
 - [43] GUERRA, G. M.; ZIO, S.; CAMATA, J. J.; DIAS, J.; ELIAS, R. N.; MATTOSO, M.; B. PARAIZO, P. L.; G. A. COUTINHO, A. L.; ROCHINHA, F. A. Uncertainty quantification in numerical simulation of particle-laden flows. *Computational Geosciences* 20, 1 (2016), 265–281.
 - [44] HASHEM, I. A. T.; YAQOOB, I.; ANUAR, N. B.; MOKHTAR, S.; GANI, A.; KHAN, S. U. The rise of “big data” on cloud computing: Review and open research issues. *Information Systems* 47 (2015), 98 – 115.
 - [45] HOFFA, C.; MEHTA, G.; FREEMAN, T.; DEELMAN, E.; KEAHEY, K.; BERRIMAN, B.; GOOD, J. On the use of cloud computing for scientific workflows. In *2008 IEEE Fourth International Conference on eScience* (Dec 2008), pp. 640–645.
 - [46] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.
 - [47] HOLLINGSWORTH, D. The Workflow Reference Model. Tech. Rep. TC00-1003, Workflow Management Coalition, Jan. 1995.
 - [48] HU, Y.; XING, L.; ZHANG, W.; XIAO, W.; TANG, D. A knowledge-based ant colony optimization for a grid workflow scheduling problem. In *Advances in Swarm Intelligence*, Y. Tan, Y. Shi, and K. Tan, Eds., vol. 6145 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 241–248.

- [49] ISARD, M.; PRABHAKARAN, V.; CURREY, J.; WIEDER, U.; TALWAR, K.; GOLDBERG, A. Quincy: Fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 261–276.
- [50] JUVE, G.; CHERVENAK, A.; DEELMAN, E.; BHARATHI, S.; MEHTA, G.; VAHI, K. Characterizing and profiling scientific workflows. *Future Generation Computer Systems* 29, 3 (2013), 682 – 692. Special Section: Recent Developments in High Performance Computing and Security.
- [51] JUVE, G.; CHERVENAK, A.; DEELMAN, E.; BHARATHI, S.; MEHTA, G.; VAHI, K. Characterizing and profiling scientific workflows. *Future Gener. Comput. Syst.* 29, 3 (Mar. 2013), 682–692.
- [52] JUVE, G.; DEELMAN, E.; VAHI, K.; MEHTA, G.; BERRIMAN, B.; BERMAN, B. P.; MAECHLING, P. Scientific workflow applications on amazon ec2. In *2009 5th IEEE International Conference on E-Science Workshops* (Dec 2009), pp. 59–66.
- [53] JUVE, G.; DEELMAN, E.; VAHI, K.; MEHTA, G.; BERRIMAN, B.; BERMAN, B. P.; MAECHLING, P. Data sharing options for scientific workflows on amazon ec2. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (Washington, DC, USA, 2010), SC '10, IEEE Computer Society, pp. 1–9.
- [54] KALRA, M.; SINGH, S. A review of metaheuristic scheduling techniques in cloud computing. *Egyptian Informatics Journal* 16, 3 (2015), 275 – 295.
- [55] KENNEDY, J.; EBERHART, R. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on* (Nov 1995), vol. 4, pp. 1942–1948 vol.4.
- [56] LIU, J.; PACITTI, E.; VALDURIEZ, P.; MATTOSO, M. A survey of data-intensive scientific workflow management. *Journal of Grid Computing* 13, 4 (2015), 457–493.
- [57] 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, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part I* (New York, NY, USA, 2014), CLADE '07, ACM, pp. 105–116.
- [58] LIVNY, J.; TEONADI, H.; LIVNY, M.; WALDOR, M. K. High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas. *PloS one* 3, 9 (2008), e3197.
- [59] LOPEZ, M. M.; HEYMANN, E.; SENAR, M. A. Analysis of dynamic heuristics for workflow scheduling on grid systems. In *2006 Fifth International Symposium on Parallel and Distributed Computing* (July 2006), pp. 199–207.
- [60] 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.

- [61] MATTOSO, M.; WERNER, C.; TRAVASSOS, G. H.; BRAGANHOLO, V.; OGASAWARA, E.; OLIVEIRA, D. D.; CRUZ, S. M.; MARTINHO, W.; MURTA, L. Towards supporting the life cycle of large scale scientific experiments. *International Journal of Business Process Integration and Management* 5, 1 (2010), 79+.
- [62] MICROSOFT. Microsoft azure. <https://azure.microsoft.com/pt-br/>.
- [63] MILLER, B. L.; GOLDBERG, D. E. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems* 9, 3 (1995), 193–212.
- [64] MORAGLIO, A. Geometry of evolutionary algorithms. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation* (New York, NY, USA, 2012), GECCO '12, ACM, pp. 1317–1344.
- [65] MOSCATO, P.; COTTA, C. *Handbook of Metaheuristics*. Springer US, Boston, MA, 2010, ch. A Modern Introduction to Memetic Algorithms, pp. 141–183.
- [66] NAVJOT KAUR TARANJIT SINGH AULAKH, R. S. C. Comparison of Workflow Scheduling Algorithms in Cloud Computing. *International Journal of Advanced Computer Science and Applications(IJACSA)* 2, 10 (2011).
- [67] OCAÑA, K.; DE OLIVEIRA, D.; OGASAWARA, E. S.; DÁVILA, A. M. R.; LIMA, A. A. B.; MATTOSO, M. SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes. In *BSB* (2011), O. N. de Souza, G. P. Telles, and M. J. Palakal, Eds., vol. 6832 of *Lecture Notes in Computer Science*, Springer, pp. 66–70.
- [68] OLIVEIRA, D. E. D.; BOERES, C.; FAUSTI, A.; PORTO, F. Avaliação da localidade de dados intermediários na execução paralela de workflows big data. In *in Proc. of the XXX Brazilian Symposium on Databases* (2015), pp. 29–40.
- [69] PANDEY, S.; WU, L.; GURU, S. M.; BUYYA, R. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *24th International Conference on Advanced Information Networking and Applications (AINA)* (2010), IEEE, pp. 400–407.
- [70] RAHMAN, M.; VENUGOPAL, S.; BUYYA, R. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. In *e-Science and Grid Computing, IEEE International Conference on* (Dec 2007), pp. 35–42.
- [71] RANGANATHAN, K.; FOSTER, I. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proceedings 11th IEEE International Symposium on High Performance Distributed Computing* (2002), pp. 352–358.
- [72] REEVES, C. R.; YAMADA, T. Genetic algorithms, path relinking, and the flowshop sequencing problem. *Evol. Comput.* 6, 1 (Mar. 1998), 45–60.
- [73] RODRIGUEZ, M. A.; BUYYA, R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Transactions on Cloud Computing* 2, 2 (April 2014), 222–235.
- [74] SCHUTTEN, J. List scheduling revisited. *Operations Research Letters* 18, 4 (1996), 167 – 170.

- [75] SHARELATEX. <https://www.sharelatex.com/>.
- [76] SILVA, J. M.; BOERES, C.; DRUMMOND, L. M.; PESSOA, A. A. Memory aware load balance strategy on a parallel branch-and-bound application. *Concurr. Comput. : Pract. Exper.* 27, 5 (Apr. 2015), 1122–1144.
- [77] SMANCHAT, S.; VIRIYAPANT, K. Taxonomies of workflow scheduling problem and techniques in the cloud. *Future Generation Computer Systems* 52 (2015), 1 – 12. Special Section: Cloud Computing: Security, Privacy and Practice.
- [78] STAMATAKIS, A. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics* 22, 21 (Oct. 2006), 2688–2690.
- [79] 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.
- [80] TALBI, E.-G. *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.
- [81] TAYLOR, I. J.; DEELMAN, E.; GANNON, D. B.; SHIELDS, M. *Workflows for e-Science: Scientific Workflows for Grids*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [82] 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, 3 (Mar 2002), 260–274.
- [83] TSAI, C. W.; RODRIGUES, J. J. P. C. Metaheuristic scheduling for cloud: A survey. *IEEE Systems Journal* 8, 1 (March 2014), 279–291.
- [84] TSUJIMURA, Y.; GEN, M. *Simulated Evolution and Learning: First Asia-Pacific Conference, SEAL'96 Taejon, Korea, November 9–12, 1996 Selected Papers*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, ch. Genetic algorithms for solving multiprocessor scheduling problems, pp. 106–115.
- [85] ULLMAN, J. D. Polynomial complete scheduling problems. *SIGOPS Oper. Syst. Rev.* 7, 4 (Jan. 1973), 96–101.
- [86] VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM Computer Communication Review* 39, 1 (Dec. 2008), 50–55.
- [87] VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.* 39, 1 (Dec. 2008), 50–55.
- [88] WALKER, E.; GUIANG, C. Challenges in executing large parameter sweep studies across widely distributed computing environments. In *Proceedings of the 5th IEEE Workshop on Challenges of Large Applications in Distributed Environments* (New York, NY, USA, 2007), CLADE '07, ACM, pp. 11–18.

- [89] WANG, M.; ZHANG, J.; DONG, F.; LUO, J. Data placement and task scheduling optimization for data intensive scientific workflow in multiple data centers environment. In *Advanced Cloud and Big Data (CBD), 2014 Second International Conference on* (Nov 2014), pp. 77–84.
- [90] WOLSTENCROFT, K.; HAINES, R.; FELLOWS, D.; WILLIAMS, A.; WITHERS, D.; OWEN, S.; SOILAND-REYES, S.; DUNLOP, I.; NENADIC, A.; FISHER, P.; BHAGAT, J.; BELHAJJAME, K.; BACALL, F.; HARDISTY, A.; NIEVA DE LA HIDALGA, A.; BALCAZAR VARGAS, M. P.; SUFI, S.; GOBLE, C. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research* 41, W1 (2013), W557–W561.
- [91] WOZNIAK, J. M.; ARMSTRONG, T. G.; WILDE, M.; KATZ, D. S.; LUSK, E.; FOSTER, I. T. Swift/t: Scalable data flow programming for many-task applications. *SIGPLAN Not.* 48, 8 (Feb. 2013), 309–310.
- [92] YOUSEFF, L.; BUTRICO, M.; SILVA, D. D. Toward a unified ontology of cloud computing. In *2008 Grid Computing Environments Workshop* (Nov 2008), pp. 1–10.
- [93] YU, J.; BUYYA, R. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Sci. Program.* 14, 3,4 (Dec. 2006), 217–230.
- [94] YU, J.; BUYYA, R.; RAMAMOHANARAO, K. *Workflow Scheduling Algorithms for Grid Computing*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 173–214.
- [95] YU, Z.; SHI, W. An adaptive rescheduling strategy for grid workflow applications. In *2007 IEEE International Parallel and Distributed Processing Symposium* (March 2007), pp. 1–8.
- [96] YUAN, D.; YANG, Y.; LIU, X.; CHEN, J. A data placement strategy in scientific cloud workflows. *Future Generation Computer Systems* 26, 8 (2010), 1200 – 1214.