UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Luis Gustavo Macedo Lousada

Avaliação de algoritmos de escalonamento de tarefas em ambientes heterogêneos e distribuídos baseado em *workflows* científicos

Uberlândia, Brasil

2026

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Luis Gustavo Macedo Lousada

Avaliação de algoritmos de escalonamento de tarefas em ambientes heterogêneos e distribuídos baseado em *workflows* científicos

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Paulo Henrique Ribeiro Gabriel

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2026

Agradecimentos

Primeiramente, agradeço a Deus por ter me proporcionado saúde, inteligência, vigor e discernimento, forças que me acompanharam durante a graduação me permitindo percorrer essa jornada com êxito.

Aos meus pais **Lucas Macedo Dias de Oliveira** e **Ana Paula Lousada Macedo** que são a base de tudo o que sou. Agradeço pelos ensinamentos ao longo da vida, pelo amor imensurável, e por sempre acreditarem em mim.

Aos meus familiares e meus amigos, que entenderam minhas ausências, comemoraram minhas conquistas e tornaram o peso dessa árdua caminhada mais leve. O incentivo e a companhia de vocês foram essenciais.

Ao meu orientador, Prof. **Paulo Henrique Ribeiro Gabriel**, agradeço imensamente pela paciência e por todo o conhecimento compartilhado. Suas orientações foram essenciais não apenas para esta monografia, mas também para a minha jornada acadêmica.

Resumo

O escalonamento de tarefas em ambientes heterogêneos e distribuídos é um desafio clássico (NP-completo) e fundamental para a otimização de determinados sistemas. Na literatura, grande parte das análises de algoritmos de escalonamento baseiam-se em grafos sintéticos, o que pode mascarar gargalos, padrões de comunicação e padrões de dependências que estão presentes em ambientes produtivos reais. A partir disso, esta monografia tem como objetivo avaliar o desempenho de cinco heurísticas (HEFT, IHEFT, PEFT, IPEFT e DLS) utilizando instâncias de *workflows* científicos reais (*1000genome*, BWA, *Epigenomics* e *Montage*). A metodologia baseou-se no desenvolvimento de um simulador que analisou as heurísticas utilizando grafos com diferentes escalas e ambientes com diferentes quantidades de processadores (4, 8, 16 e 32). Após a simulação, avaliamos os resultados sob a ótica das seguintes métricas: *makespan*, *scheduling length ratio*, *load balancing*, *waiting time* e *communication cost*. Os resultados demonstram que a eficiência do escalonamento é muito condicionada pela topologia do grafo, pela escala da infraestrutura e pelas heurísticas usadas pelos algoritmos. O IPEFT destacou-se pela minimização do *makespan* ao proteger o caminho crítico, enquanto o PEFT, utilizando sua estratégia de *look-ahead*, provou-se altamente eficiente no gerenciamento do *waiting time* e *communication cost*. Por outro lado, algoritmos que se baseiam em políticas do tipo *non-insertion*, como o DLS, ou em mecanismos de troca de processador baseada em um *threshold*, como o IHEFT, revelaram os piores resultados dependendo da densidade do grafo. Conclui-se que, embora não exista uma solução universal para todos os cenários, abordagens que antecipam gargalos (*look-ahead*) ou protegem tarefas críticas demonstram maior adaptabilidade, reforçando que o desenvolvimento de sistemas distribuídos exige um profundo entendimento da natureza de cada problema e da aplicação.

Palavras-chave: Computação Paralela, Escalonamento de Tarefas, Ambientes Heterogêneos, Aplicações Paralelas, Sistemas distribuídos, DAG.

Abstract

Task scheduling in heterogeneous and distributed environments is a classic (NP-complete) challenge and is fundamental to the optimization of certain systems. In the literature, most analyses of scheduling algorithms are based on synthetic graphs, which can mask bottlenecks, communication patterns, and dependency patterns that are present in real production environments. Based on this, this thesis aims to evaluate the performance of five heuristics (HEFT, IHEFT, PEFT, IPEFT, and DLS) using instances of real scientific workflows (1000genome, BWA, Epigenomics, and Montage). The methodology was based on the development of a simulator that analyzed the heuristics using graphs of different scales and environments with varying numbers of processors (4, 8, 16, and 32). Following the simulation, the results were evaluated using the following metrics: makespan, scheduling length ratio, load balancing, waiting time, and communication cost. The results demonstrate that scheduling efficiency is heavily influenced by the graph topology, the scale of the infrastructure, and the heuristics used by the algorithms. IPEFT stood out for minimizing makespan by protecting the critical path, while PEFT, using its look-ahead strategy, proved highly efficient in managing waiting time and communication cost. On the other hand, algorithms based on non-insertion policies, such as DLS, or on threshold-based processor-switching mechanisms, such as IHEFT, yielded the worst results depending on the graph density. It can be concluded that, although there is no universal solution for all scenarios, approaches that anticipate bottlenecks (look-ahead) or protect critical tasks demonstrate greater adaptability, reinforcing the fact that the development of distributed systems requires a deep understanding of the nature of each problem and application.

Palavras-chave: Parallel Computing, Task Scheduling, Heterogeneous Environments, Parallel Applications, Distributed Systems, DAG.

Lista de ilustrações

Lista de abreviaturas e siglas

AFT *Actual Finish Time*

AWS *Amazon Web Services*

BLAST *Basic Local Alignment Search Tool*

BWA *Burrows-Wheeler Aligner*

CC *Communication Cost*

CP *Critical Path*

CPOP *Critical Path on a Processor*

DAG *Directed Acyclic Graph*

DLS *Dynamic Level Scheduling*

EFT *Earliest Finish Time*

EST *Earliest Start Time*

HEFT *Heterogeneous Earliest Finish Time*

IHEFT *Improved Heterogeneous Earliest Finish Time*

IPEFT *Improved Predict Earliest Finish Time*

LB *Load Balancing*

OCT *Optimistic Cost Table*

PCT *Pessimistic Cost Table*

PEFT *Predict Earliest Finish Time*

SLR *Scheduling Length Ratio*

WT *Waiting Time*

Sumário

1 Introdução

A utilização de sistemas distribuídos e heterogêneos no processamento de aplicações paralelas e na resolução de problemas computacionais complexos se mostrou um pilar da computação moderna. Devido à grande relevância da computação distribuída nas atividades que exigem alta escalabilidade e eficiência operacional, busca-se a melhoria contínua do processamento desses *workloads*, a fim de torná-los mais eficientes e/ou acelerados. Um dos meios para atingir tal melhoria é o aperfeiçoamento dos algoritmos de escalonamento de tarefas (ALEBRAHIM; AHMAD, 2017a), as quais serão processadas nos diferentes *nodes* do sistema.

O uso mais eficiente dos recursos de um sistema distribuído tem grande aplicação prática, seja na área da saúde e ciências biológicas, no setor de engenharia, serviços financeiros ou até na energia e meio ambiente (AWS, 2025). A computação paralela pode ser definida por:

> Um sistema de multiprogramação no qual todas as atividades são divididas entre vários processos sequenciais. Esses processos sequenciais estão situados em vários níveis hierárquicos, em cada um dos quais uma ou mais abstrações independentes foram implementadas. A estrutura hierárquica revelou-se essencial para a verificação da solidez lógica (DIJKSTRA, 1967)

Dessa forma, a distribuição, ordem e local onde esses elementos (tarefas) serão executados são de suma importância para tornar o processamento mais rápido e eficiente nos ambientes distribuídos.

O escalonamento de tarefas é um problema já bem conhecido e muito abordado pela academia, além de ser recorrente no dia a dia dos profissionais que lidam com computação distribuída. Dessa forma, o problema já conta com diversas pesquisas, estudos e soluções práticas (MOHAMMADJAFARI; KHAJOUIE, 2024) com a finalidade de solucionar, ou minimizar, as dificuldades. Estudiosos da área já pensaram sobre o problema e desenvolveram alguns algoritmos que são influentes até hoje, como, por exemplo, os algoritmos *Heterogeneous Earliest Finish Time* (HEFT), *Critical Path on a Processor* (CPOP) e *Min-You Wu* (TOPCUOGLU; HARIRI; WU, 1999), ou as heurísticas *Min-min* e *Max-min* (BRAUN et al., 2001). Além disso, diversas iterações foram feitas utilizando algoritmos já existentes como base, a fim de obter melhores resultados, como é o caso dos algoritmos *Improved Heterogeneous Earliest Finish Time* (IHEFT) (ALEBRAHIM; AHMAD, 2017a), e *Improved Predict Earliest Finish Time* (IPEFT) proposto por Zhou et al. (2017). Com a tecnologia se tornando cada vez mais complexa, também foi necessário desenvolver novos algoritmos que tentassem acompanhar a complexidade intrínseca dos problemas da computação, dessa forma, alguns algoritmos de escalonamento usam a com-

putação bioinspirada (L.D.; Venkata Krishna, 2013) para propor diferentes abordagens, e até soluções guiadas por *Deep Learning* (MAO et al., 2016). Dados esses sofisticados trabalhos, levanta-se o questionamento de qual é o algoritmo mais adequado para cada tipo, natureza ou classe de problemas, com isso, novos trabalhos surgem a fim de fazer uma comparação empírica de diversos algoritmos, exemplo o trabalho de Monteiro (2025) que analisou o resultado de determinados algoritmos e obteve comportamentos não previstos em seus artigos de origem.

1.1 Modelagem do Problema

No contexto do escalonamento de tarefas em sistemas distribuídos, uma aplicação é tradicionalmente modelada como um *Directed Acyclic Graph* (DAG). Dessa forma, o objetivo do algoritmo de escalonamento é, em sua síntese, determinar qual é o melhor processador para executar determinada tarefa.

Achar uma solução ótima para o problema de escalonamento em ambientes heterogêneos é um problema NP-completo (ARABNEJAD; BARBOSA, 2014). Calcular todas as combinações possíveis de escalonamento para determinado DAG e conjunto de processadores se torna computacionalmente inviável à medida que o conjunto de tarefas aumenta. Com isso, foi necessário o desenvolvimento de heurísticas que permitam encontrar soluções subótimas viáveis em tempo polinomial para o problema.

Boa parte dos algoritmos são do tipo *list-scheduling*, tal metodologia consiste de duas fases principais que guiam a modelagem do algoritmo, a fase de **priorização da tarefa**, onde cada tarefa tem um peso (prioridade) atribuída, e a fase de **escolha do processador**, onde a tarefa priorizada na fase anterior é atribuída ao processador que minimiza uma função de custo determinada.

Além da metodologia *list-scheduling* citada anteriormente, a heurística pode ser **estática** ou **dinâmica**. **Estática** são aquelas onde a fase de escolha do processador começa após a fase de priorização da tarefa acabar por completo, já **dinâmicas** são heuríticas onde as 2 fases são intercaladas.

O trabalho de Hagras e Janeček (2003) realizou um estudo comparativo da performance desses dois tipos de heurísticas em ambientes homogêneos, e constatou que na maioria dos casos, abordagens do tipo ***dynamic list-scheduling*** apresentam melhor desempenho em métricas como *average makespan* e *average SLR*. Já quando a complexidade assintótica tem maior relevância, é recomendado o uso de heurísticas do tipo ***static list-scheduling***. Diferenciando-se desse trabalho, esta monografia expande o escopo do estudo ao focar em arquiteturas heterogêneas e utilizar DAGs reais, buscando investigar se as premissas e vantagens observadas se sustentam.

1.2 Motivação e Objetivo

Muitas das avaliações de algoritmos de escalonamento presentes na literatura se baseiam em DAGs sintéticos ou aleatórios, o que nem sempre reflete os gargalos, as complexidades e as características de ambientes de produção. Diante disso, a principal motivação desta monografia é ir além da teoria e trazer cenários reais para as análises. Submeter algoritmos clássicos a ambientes diversos e problemas reais distintos é fundamental para investigar comportamentos ainda não mapeados e para validar a viabilidade de tais esforços serem integrados no mundo real.

Nesse contexto, o objetivo principal deste trabalho é avaliar e comparar o desempenho de cinco heurísticas clássicas de escalonamento de tarefas em ambientes heterogêneos, utilizando exclusivamente instâncias de *workflows* científicos reais extraídas do repositório WfCommons (2024a). Ao empregar topologias de aplicações reais, busca-se expor os algoritmos a gargalos estruturais e a diferentes graus de paralelismo, aproximando a análise dos desafios enfrentados no mundo real.

Para alcançar os objetivos propostos e garantir o caráter científico dos resultados, este trabalho adota uma metodologia experimental e comparativa. Uma vez escolhidos alguns algoritmos notáveis pela academia, foram executados utilizando diversos DAGs como entrada e em seguida, ao fim do escalonamento, foi feita uma análise comparativa dos resultados de cada algoritmo levando em consideração diversas métricas fundamentais para o problema do escalonamento de tarefas em ambientes distribuídos e heterogêneos.

Toda a avaliação foi feita por meio de um simulador determinístico desenvolvido em Python, que é mais detalhado nas próximas seções. A escolha pela utilização de um simulador determinístico garante um ambiente de execução estritamente controlado, onde é possível a modificação dos parâmetros, como o DAG utilizado, heterogeneidade dos processadores, custos de comunicação, etc. Sem a variabilidade de *hardware* ou de rede, que podem representar variáveis complexas de controlar com precisão durante os experimentos.

Foram avaliados os seguintes algoritmos de escalonamento:

1. *Dynamic Level Scheduling* (DLS).
2. *Heterogeneous Earliest Finish Time* (HEFT).
3. *Improved Heterogeneous Earliest Finish Time* (IHEFT).
4. *Predict Earliest Finish Time* (PEFT).
5. *Improved Predict Earliest Finish Time* (IPEFT).

Após a simulação independente de cada algoritmo, algumas métricas foram extraídas e utilizadas para avaliar o desempenho de cada heurística. São elas:

1. *Makespan*: Tempo total para concluir todas as tarefas.
2. *Scheduling Length Ratio* (SLR): *Makespan* normalizado, permite a comparação justa entre grafos distintos.
3. *Load Balancing* (LB): Grau de eficiência e uniformidade da distribuição de carga nos processadores.
4. *Communication Cost* (CC): Custo de comunicação total durante a execução do DAG.
5. *Waiting Time* (WT): Tempo em que as tarefas permanecem aguardando liberação do processador para iniciarem sua execução.

1.2.1 Estrutura do Trabalho

Esta monografia está estruturada da seguinte forma:

- **Capítulo 2 – Fundamentação Teórica:** aborda os conceitos essenciais da computação distribuída, define o problema do escalonamento de tarefas baseado em grafos, as métricas de avaliação, formalismo matemático, e por fim, os trabalhos relacionados.
- **Capítulo 3 – Método e Desenvolvimento:** descreve a metodologia adotada, explanando a arquitetura do simulador construído, detalhando o funcionamento das heurísticas escolhidas e as características dos *workflows* utilizados para a análise.
- **Capítulo 4 – Resultados e Análise:** apresenta os resultados obtidos, discutindo o comportamento dos algoritmos frente a diferentes topologias e variações na escala da infraestrutura, além de fazer uma comparação com simulações que utilizam DAGs sintéticos.
- **Capítulo 5 – Conclusão:** sintetiza as análises feitas e evidencia o impacto de cada estratégia de escalonamento nos cenários analisados.

2 Fundamentação Teórica

Este capítulo apresenta alguns conceitos fundamentais para embasar esta monografia. Inicialmente, é feita uma revisão da literatura e sua evolução, para que sustente o desenvolvimento dessa pesquisa. Logo após, define-se o problema do escalonamento de tarefas, junto do seu modelo de tarefas baseado em grafos e as características de um ambiente heterogêneo. Posteriormente, são definidas algumas fórmulas matemáticas e métricas de avaliação utilizadas para comparar o desempenho dos algoritmos. Por fim, é feita uma revisão bibliográfica, e encerra-se com a análise de trabalhos relacionados com esta monografia, que são relevantes para o meio da computação distribuída.

2.1 Revisão Bibliográfica

A computação paralela refere-se à técnica de usar vários processadores, computadores ou núcleos para resolver problemas de forma simultânea, acelerando o processamento e com um aproveitamento mais inteligente dos recursos. Segundo Dijkstra (1967), sistemas de multiprogramação são compostos por atividades divididas em processos sequenciais, que são distribuídos em níveis hierárquicos. Esses processos independentes são essenciais para garantir a eficiência da execução. Além disso, a computação distribuída, envolve a distribuição de tarefas entre computadores localizados em diferentes localidades e com poder de processamento distintos (AWS, 2025).

Desde as primeiras arquiteturas de multiprocessamento, a computação paralela e distribuída passou a desempenhar um papel importantíssimo na resolução de problemas complexos que exigem um alto grau de processamento. A evolução dos computadores e, nos dias de hoje a rápida adoção da computação em nuvem, permitiu que esses conceitos fossem ampliados para ambientes heterogêneos, onde diferentes tipos de processadores e arquiteturas interagem de forma coordenada.

Nos últimos anos, houve um grande avanço na utilização de *clusters* de alta performance e redes de computadores para integrar esses diferentes *nodes* para criar soluções distribuídas. Além disso, o uso de tecnologias como *Hadoop* (FOUNDATION, 2006), *Spark* (FOUNDATION, 2009) ou orquestradores de *containers* como Kubernetes (2014) permitiram uma evolução na computação paralela e distribuída, com a democratização do acesso a tais tecnologias.

Um exemplo clássico de computação paralela são os problemas de otimização, como a programação linear ou a solução de problemas combinatórios, que possam ser divididos em tarefas menores e processados simultaneamente. Na computação distribuída,

um exemplo seria o uso de ambientes de nuvem como a *Amazon Web Services* (AWS) ou o *Google Cloud* para processamento de grandes volumes de dados, onde as tarefas são distribuídas entre diferentes servidores localizados geograficamente em pontos distintos e com capacidades de processamento distintas.

Recentemente, alguns dos algoritmos já conhecidos foram revisitados e melhorados, como é o caso do *Improved Heterogeneous Earliest Finish Time* (ALEBRAHIM; AHMAD, 2017a) e o *Improved Predict Earliest Finish Time* proposto por Zhou et al. (2017). Além disso, com os problemas se tornando cada vez mais complexos, também foi necessário desenvolver novos algoritmos que tentassem acompanhar tal complexidade, como, algoritmos que usam a computação bioinspirada (CUNHA, 2024) para propor diferentes abordagens, e até soluções guiadas por *Deep Learning* (MAO et al., 2016).

2.2 Escalonamento de Tarefas

O escalonamento de tarefas é o processo de distribuir e atribuir as tarefas (*tasks*) - é considerado tarefas como sendo as unidades de trabalho que devem ser executadas - nos diferentes *nodes* do sistema, de forma a otimizar o desempenho da aplicação. Em sistemas heterogêneos e distribuídos, o escalonamento se torna um desafio, tanto por cada máquina apresentar capacidades distintas, como capacidade de processamento e memória, tanto pelo custo de comunicação (rede) entre os *nodes*. O escalonamento tem como objetivo distribuir as tarefas de forma eficiente a fim de reduzir o tempo total de execução e maximizar a utilização dos recursos disponíveis (ALEBRAHIM; AHMAD, 2017a).

Devido à capacidade de processamento dos computadores da época, os primeiros estudos sobre escalonamento de tarefas focavam em sistemas homogêneos, onde todos os nós eram iguais em termos de capacidade de processamento. Contudo, com o crescimento e o avanço da tecnologia, surgiram os sistemas heterogêneos, onde os *nodes* podem ser diferentes, exigindo algoritmos mais sofisticados que levem em consideração a particularidade de cada máquina, como o IHEFT ou o IPEFT.

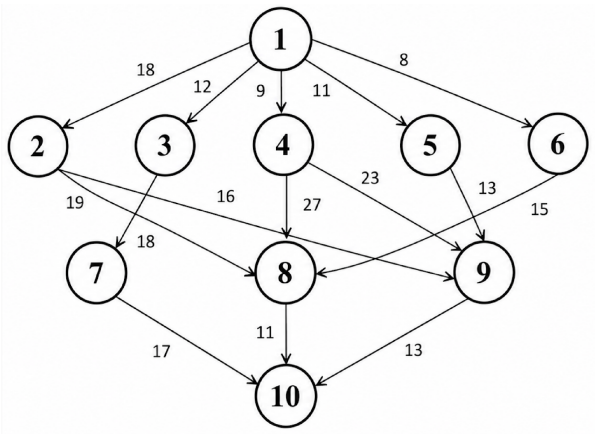
Em um ambiente de *cloud computing*, o escalonamento de tarefas tem grande importância para garantir que os recursos sejam utilizados de maneira eficiente. Um exemplo é o *Basic Local Alignment Search Tool* (BLAST) (1990) que encontra regiões de similaridade entre sequências biológicas. O programa compara sequências de nucleotídeos ou proteínas com bancos de dados de sequências e calcula a significância estatística, o BLAST utiliza um DAG para representar a hierarquia das tarefas e conseguir realizar o escalonamento, a fim de tornar o sistema mais eficiente e a execução no menor tempo.

2.2.1 Modelo de Tarefas (DAG)

O escalonamento de tarefas em sistemas distribuídos é tradicionalmente modelado por um DAG ponderado $G = (V, E)$, e as matrizes C e W, onde:

- V: É o conjunto de vértices (ou nós), em que cada vértice $v \in V$ representa uma unidade de trabalho (tarefa) a ser executada.
- E: É o conjunto de arestas (direcionadas), onde cada aresta $e(i, j) \in E$ indica uma dependência de precedência, ou seja, a tarefa j só pode iniciar após a conclusão completa da tarefa i.
- C: É o peso de cada aresta, que representa os custos de comunicação - Denota o tempo ou custo de transferir os dados entre os *nodes*.
- W: São os custos computacionais, representando o tempo de execução estimado para cada tarefa em cada processador.

A Figura 1 ilustra um exemplo dessa modelagem, evidenciando as dependências de execução, seus pesos 1a e os custos de computação 1b.



(a)

Tarefa	P_1	P_2	P_3	$\overline{W}$
1	4	3	2	3
2	5	4	3	4
3	2	3	3	3
4	5	5	5	5
5	4	3	2	3
6	6	3	3	4
7	6	4	2	4
8	7	5	3	5
9	4	6	5	5
10	5	3	1	5

(b)

Figura 1 – (a) DAG de escalonamento / (b) Tabela de custos

Fonte: Hesar, Alireza em Hesar (2023).

Além disso, um conjunto de recursos computacionais $P = \{p_1, p_2, ..., p_q\}$ representa cada máquina ou processo onde serão executadas as tarefas. Em um ambiente homogêneo cada máquina será idêntica, terá a mesma quantidade de *cores*, CPU, velocidade de

clock, geração, etc, ou seja, com poder de processamento estritamente igual. Já em um ambiente heterogêneo, cada máquina pode ter seu poder de processamento diferente, ou seja, uma tarefa pode ter tempo de execução diferente se for alocado em um processador diferente, sendo assim, utiliza-se de uma matriz de custo para representar a relação tarefa x processador e seu tempo de execução - Figura 1b.

No problema do escalonamento de tarefas assume-se que o DAG terá somente um *node* de entrada (n_{start}) e um *node* de saída (n_{exit}). Devido a isso, caso o DAG possua mais de um *node* de entrada ou de saída eles serão conectados a um *node* artificial (ALEBRAHIM; AHMAD, 2017b), fazendo com que o grafo fique com apenas uma entrada e uma saída.

2.2.2 Premissas do modelo

O modelo adotado assume algumas premissas importantes que balizam a execução das tarefas, premissas essas:

1. **Paralelismo:** O paralelismo da aplicação é extraído da topologia do DAG. Tarefas que não são dependentes podem ser executadas simultaneamente, desde que haja processadores disponíveis e cores disponíveis na máquina.

2. **Atomicidade:** Cada tarefa n_i é considerada uma unidade indivisível de processamento.

3. **Não preempção:** Uma vez que uma tarefa inicia sua execução em um processador, ela não pode ser interrompida ou transferida para outro recurso.

2.2.3 Custo de comunicação

O ambiente de execução é composto por um conjunto $P = \{p_1, p_2, ..., p_q\}$ de q processadores heterogêneos. Devido à heterogeneidade do ambiente, uma mesma tarefa pode ter tempos de execução diferentes dependendo de qual processador for alocada.

Para modelar tal característica, utiliza-se a matriz de custo computacional W de dimensão $v \times q$, onde cada elemento $w_{i,j}$ representa o tempo estimado de execução da tarefa n_i no processador p_j.

Além do processamento, há o custo de comunicação $c_{i,j}$, que refere-se ao custo para transferir os dados da tarefa n_i para a tarefa n_j em máquinas distintas. É representado pelos pesos nas arestas, e definido por

$$c_{i,j} = \overline{L} + \frac{data_{i,j}}{B}, \tag{2.1}$$

Onde $\overline{L}$ é a latência média dos processadores, B é a largura de banda e $data_{i,j}$ é a quantidade de dados em transferência. Se ambas as tarefas forem executadas no mesmo processador, $c_{i,j}$ é considerado zero.

2.3 Definições

A fim de implementar os algoritmos escolhidos, e as métricas utilizadas para a comparação dos mesmos, definimos alguns conceitos essenciais para o entendimento do modelo e a construção das heurísticas.

2.3.1 *Earliest Start Time* (EST)

Para decidir a melhor alocação, a grande maioria das heurísticas de escalonamento calcula o tempo de início da tarefa n_i no processador p_j. O EST é calculado para cada tarefa recursivamente a partir do nó de entrada.

O *Earliest Start Time* indica o momento mais cedo em que a tarefa n_i pode iniciar no processador p_j, aguardando a finalização dos seus predecessores e a chegada de todos os dados. É dado por:

$$EST(n_i, p_j) = \max\left\{avail(p_j), \max_{n_m \in pred(n_i)}(AFT(n_m) + c_{m,i})\right\}$$

Onde $avail(p_j)$ é o tempo em que o processador p_j estará livre, $pred(n_i)$ é o conjunto de tarefas predecessoras de n_i, e $AFT(n_m)$ é o *Actual Finish Time* da tarefa predecessora.

Algoritmos que seguem uma política baseada em inserção (*insertion based policy*) buscam o primeiro *slot* de ociosidade entre duas tarefas já alocadas que seja grande o suficiente para acomodar a nova tarefa. Já algoritmos que optam por seguir o mecanismo de *non-insertion based policy* consideram alocar novas tarefas somente quando a última tarefa daquele recurso terminou sua execução (TOPCUOGLU; HARIRI; WU, 1999).

2.3.2 *Earliest Finish Time* (EFT)

O *Earliest Finish Time* indica o momento mais cedo em que a tarefa será concluída, ou seja, a soma do momento que ela se inicia mais o seu custo computacional no processador selecionado:

$$EFT(n_i, p_j) = EST(n_i, p_j) + w_{i,j}$$

Onde $w_{i,j}$ é a matriz de custo computacional da tarefa n_i no processador p_j.

2.3.3 *Actual Finish Time* (AFT)

O *Actual Finish Time* representa o tempo exato e definitivo em que a tarefa n_i conclui sua execução. Após o escalonamento, o AFT da tarefa assume o valor do EFT previamente calculado para aquele recurso. Definido por:

$$AFT(n_i) = EFT(n_i, p_k)$$

Onde p_k é o processador selecionado para executar a tarefa n_i.

2.3.4 *Critical Path* (CP)

O CP (caminho crítico) representa o caminho mais longo entre a tarefa de entrada (n_{start}) até a tarefa de saída (n_{exit}). Além disso, é derivado o $CPmin$ que é o tamanho mínimo do *critical path* (limite inferior), calculado a partir da soma do menor custo computacional de cada tarefa do caminho crítico.

2.3.5 Predecessores e Sucessores

$pred(n_i)$ engloba todas as tarefas que são predecessoras diretas da tarefa n_i, se $pred(n_i) = 0$, diz que n_i é tarefa de entrada (*start task*). A tarefa n_i só será considerada "pronta" (*ready task*) para iniciar sua execução assim que todas as tarefas $n_m \in pred(n_i)$ tiverem sido concluídas e todos os dados necessários para a execução tiverem chegado em n_i. Matematicamente, pode ser expressa como:

$$pred(n_i) = \{n_m \in V \mid e(m, i) \in E\}$$

De forma correlata, o conjunto de sucessores é definido por, $succ(n_i)$, que engloba todas as tarefas que dependem diretamente da conclusão de n_i, e se $succ(n_i) = 0$. diz que n_i é a tarefa de saída (*exit task*).

2.4 Métricas utilizadas

A fim de comparar quantitativamente a eficiência das heurísticas propostas na literatura, foram escolhidas 5 métricas para realizar essa análise comparativa, as próximas subseções definem formalmente cada métrica.

2.4.1 *Makespan*

O *makespan* representa o tempo absoluto em que a última *task* do *workflow* (DAG) é finalizada. Formalmente, é o (AFT) da tarefa de saída (n_{exit}):

$$Makespan = AFT(n_{exit})$$

O principal objetivo de boa parte dos algoritmos de escalonamento é minimizar o *makespan*.

2.4.2 *Scheduling Length Ratio* (SLR)

Como o *makespan* de diferentes DAGs pode variar consideravelmente dependendo do tamanho e topologia do grafo escolhido, a fim de permitir comparações justas entre cenários distintos utiliza-se o *scheduling length ratio* para normalizar os resultados. O SLR é definido por:

$$SLR = \frac{Makespan}{\sum_{n_i \in CP_{min}} \min_{p_j \in P} w_{i,j}}$$

É a divisão do *makespan* pelo custo computacional do *critical path* alocado nos processadores mais rápidos (CP_{min}).

2.4.3 *Load Balancing* (LB)

O balanceamento de carga (*load balancing*) tem como objetivo avaliar o quão eficiente e uniforme o algoritmo distribui as tarefas entre os processadores. Valores de LB que se aproximam de 1 indicam um balanceamento ideal, pois o tempo de conclusão *makespan* se aproxima da média de execução.

Matematicamente, o LB é a razão entre o *makespan* e a média do tempo de execução dos recursos. Sendo $ftp[p_j]$ o tempo total de execução (ocupação) do processador p_j, e m o número total de processadores, a média de execução (avg) é:

$$avg = \frac{\sum_{j=1}^{m} ftp[p_j]}{m}$$

A partir disso, a equação do *load balancing* é dada por:

$$LB = \frac{Makespan}{avg}$$

2.4.4 *Waiting Time* (WT)

O *waiting time* mede a diferença de tempo entre o momento que uma tarefa tem todas as suas dependências resolvidas, ou seja, está pronta para ser executada e o momento em que ela efetivamente começa a ser processada. Algoritmos menos eficientes podem

gerar longas filas de espera devido à má alocação, pois a *task* pode ter que esperar a desocupação do processador escolhido. O WT é definido por:

$$WT = \frac{1}{v} \sum_{i=1}^{v} AST(n_i)$$

2.4.5 *Communication Cost* (CC)

O *communication cost* representa o valor total da transferência de dados durante a execução do *workflow*. Essa é uma métrica importante, onde algoritmos que agrupam tarefas com alto volume de dados dependentes no mesmo processador tendem a apresentar menores custos de comunicação, mitigando o *overhead* e protegendo das variações da rede.

Para calcular o CC, soma-se o peso de todas as arestas nas quais os nós foram alocados em recursos distintos. Sendo $p(n_i)$ o processador alocado para a tarefa n_i, o custo de comunicação é definido por:

$$CC = \sum_{e(i,j) \in E} c_{i,j}, \quad \text{onde } c_{i,j} = 0 \text{ se } p(n_i) = p(n_j)$$

2.5 Trabalhos relacionados

No artigo de Braun et al. (2001), os autores tiveram como objetivo analisar e comparar 11 diferentes heurísticas de escalonamento de tarefas em ambientes de computação heterogênea. Para isso, foram adaptadas, implementadas e comparadas algumas heurísticas populares, dentro de um mesmo ambiente e com os mesmos casos de teste, utilizando simulações com *meta-tasks* - tarefas independentes e sem dependência de dados - e mapeamento estático. Os resultados indicaram que a heurística *Min-min* teve um desempenho consistentemente bom em comparação com as outras técnicas mais complexas. O artigo se relaciona com este trabalho pois ambos oferecem uma fundação sólida e voltada à investigação, a fim de escolher as melhores heurísticas em ambientes heterogêneos para determinados problemas, principalmente quando o objetivo é minimizar o *makespan*, um cenário comum nos dias atuais.

Na monografia de COSTA (2022), o autor seleciona 4 dos principais algoritmos de escalonamento para analisar seu comportamento em diferentes ambientes de execução. Para isso, foi desenvolvido o código que produz o comportamento exato dos algoritmos. Como casos de teste, foram avaliados alguns DAGs gerados a partir de um programa desenvolvido exclusivamente para este trabalho, a fim de tornar os grafos customizados e de fácil geração, permitindo assim a execução dos algoritmos em cenários distintos. Como método de avaliação foi feita uma comparação entre as principais métricas, *makespan* e *load balancing*. Por fim, após as análises o autor concluiu que o IPEFT obteve melhores

resultados em ambientes heterogêneos quando comparado com os outros algoritmos, já em ambientes homogêneos os algoritmos IPEFT e HEFT obtiveram resultados similares. Concluindo, o trabalho de COSTA (2022) se assemelha com este trabalho uma vez que ambos se propõem a investigar o desempenho de algoritmos de escalonamento em diferentes cenários dentro de ambientes distribuídos e heterogêneos.

Já no trabalho de Monteiro (2025), o autor propôs uma análise comparativa de algoritmos de escalonamento em processadores heterogêneos. Para realizar a análise, Monteiro explorou o comportamento das execuções em diferentes cenários utilizando DAGs sintéticos e comparou seis heurísticas clássicas baseadas em listas: DLS, HEFT, HEFT-LA, PEFT, IHEFT e IPEFT. Para comparar o desempenho e investigar as características dos algoritmos, foram usadas métricas como *makespan*, *load balancing*, *waiting time*, entre outras. Após as análises, foi possível concluir que as heurísticas com mecanismo de *look-ahead* (como PEFT, IPEFT e HEFT-LA) apresentaram, em média, reduções consistentes no *makespan* e no tempo de espera em cenários dominados por comunicação. Assim como o trabalho de COSTA (2022), o trabalho de Monteiro (2025) se assemelha com esta monografia devido à natureza experimental e comparativa dos algoritmos para solucionar o problema do escalonamento de tarefas.

2.6 Considerações Finais

Este capítulo estabeleceu os alicerces teóricos e matemáticos necessários para a compreensão do problema de escalonamento em ambientes heterogêneos. A formalização do modelo via DAG e a definição de métricas consolidadas fornecem a base analítica para o estudo. Através da revisão de trabalhos relacionados, evidenciou-se que muitas avaliações na literatura se limitam a simulações com grafos sintéticos ou aleatórios. Para preencher essa lacuna e garantir maior rigor prático, este trabalho foca na validação dessas heurísticas utilizando *workflows* científicos reais. Essa abordagem garante que os algoritmos sejam testados em topologias, gargalos e padrões de dependência encontrados em ambientes produtivos reais.

3 Método e Desenvolvimento

3.1 Simulador de Escalonamento de Tarefas

3.1.1 Visão Geral e Propósito

O simulador desenvolvido para esta monografia foi projetado para avaliar algoritmos de escalonamento de tarefas em ambientes distribuídos e heterogêneos. Como o foco deste trabalho é a análise em cenários de ambientes produtivos reais, foi abandonada a ideia da geração de DAGs sintéticos e decidiu-se por utilizar o padrão *WfFormat* (2024b), permitindo assim desfrutar do repositório de instâncias *WfInstances* (2024a) e utilizar *workflows* extraídos do mundo real.

A ferramenta simula o escalonamento das tarefas utilizando algoritmos de escalonamento, como DLS, HEFT, PEFT, etc. Para, no fim, expor algumas métricas que vão ser utilizadas para a análise de desempenho de cada algoritmo.

Para garantir a transparência, e fomentar possíveis contribuições para a área, o código-fonte do simulador foi disponibilizado de forma *open source*. O projeto pode ser acessado através do repositório oficial[1].

3.1.2 Arquitetura de Alto Nível

Para garantir a separação de responsabilidades e a facilidade de extensão, a aplicação foi estruturada em quatro camadas:

- **Camada de Apresentação e *command line interface*** (`main.py`): Atua como ponto de entrada do sistema. Responsável por instanciar o motor de simulação e orquestrar a execução do algoritmo selecionado.
- **Motor de Execução** (`simulator.py`): É o motor do simulador. Ele mantém o estado global da simulação e manipula, através da interface, o algoritmo escolhido.
- **Módulo de Escalonadores** (`schedulers/`): Contém as implementações das heurísticas (ex: HEFT, PEFT, DLS, etc.) seguindo uma interface pré-estipulada.
- **Análise e Apresentação de Resultados** (`metrics.py` e `charts.py`): Responsável por consumir os dados pós-execução e derivar as métricas.

[1] Repositório do Simulador de Escalonamento: <https://github.com/Luisgustavom1/task-scheduling>

3.1.3 Dinâmica de Funcionamento do Motor

O processo de simulação ocorre em um *loop* fechado de interação entre o motor e o algoritmo. Antes do início do escalonamento, o simulador realiza a normalização das tarefas de entrada e saída do *workflow*. Garantindo assim que todo o grafo possua um único ponto de entrada e um único ponto de saída, um dos requisitos estruturais do problema de escalonamento de tarefas.

Além da normalização, o motor pré-calcula alguns dados essenciais para posteriormente começar a execução. Durante o escalonamento, o algoritmo escolhido é chamado repetidamente e para cada decisão tomada pelo algoritmo, o simulador calcula o tempo de chegada dos dados (*data ready time*), e registra a alocação no histórico.

3.1.4 Uso Prático e Operação

O simulador foi desenhado para ser usado via *command line interface* (CLI). A aplicação suporta os seguintes parâmetros para a configuração e controle das simulações:

- `-scheduler`: Seleciona o algoritmo de escalonamento a ser utilizado na simulação (`HEFT`, `PEFT`, `IPEFT`, `IHEFT` ou `DLS`).
- `-compare`: Executa sequencialmente todos os algoritmos disponíveis e gera gráficos comparativos para as métricas avaliadas.
- `-dag-path`: Define o caminho para o arquivo JSON do DAG (no padrão *WfFormat* (2024b)) a ser carregado e utilizado.
- `-log-level`: Configura o nível de verbosidade dos *logs* (INFO, DEBUG, ERROR).
- `-silence`: Desativa a emissão de *logs* do simulador, útil para manter o *output* mais limpo.

Dessa forma, pode-se optar por analisar o comportamento de um algoritmo isolado (fornecendo o argumento `-scheduler`) ou executar uma comparação abrangente entre todos os algoritmos implementados (utilizando a *flag* `-compare`).

3.1.5 Ambiente de Simulação

Para garantir a transparência e o determinismo dos experimentos, foi utilizado o seguinte ambiente computacional e ferramentas de *software* para o desenvolvimento:

- **Sistema Operacional:** macOS 15.7.7;

- ***Software* e Bibliotecas:** Python 3.10+; bibliotecas *matplotlib*, *wfcommons* e *matplotlib*;
- ***Hardware*:** *Apple MacBook Pro, Chip Apple M1 Pro* (10 núcleos: 8 de performance e 2 de eficiência); RAM: 16 GB.

Ressalta-se que, devido ao caráter determinístico do simulador, as variações de capacidade do sistema operacional não interferem nos resultados das métricas, deixando o *hardware* exclusivamente como motor de processamento para a execução dos algoritmos e consolidação dos dados.

3.2 Algoritmos Escolhidos

Para compor a base de avaliação deste trabalho, foram selecionados cinco algoritmos de escalonamento para ambientes heterogêneos. O critério de seleção buscou abranger desde as abordagens clássicas, que servem como linha de base (*baseline*) consolidada como estado da arte, até variações conceituais mais recentes que propõem otimizações matemáticas sobre os modelos originais.

Foram escolhidas as seguintes heurísticas:

- *Dynamic Level Scheduling* (DLS): Heurística clássica que calcula a prioridade com base no ***dynamic level*** (DL) e serve-se de uma política de alocação estrita que não realiza inserção em *gaps* de ociosidade do processador (*non-insertion*).
- *Heterogeneous Earliest Finish Time* (HEFT): Um dos algoritmos que são usados como *baseline* para a construção de novos, ele ordena as tarefas com base no ***upward rank*** e as aloca no processador que minimiza o *Earliest Finish Time*.
- *Improved Heterogeneous Earliest Finish Time* (IHEFT): Uma extensão do HEFT que combina o ranqueamento das tarefas com uma escolha randômica, baseada em um ***threshold***, entre o processador que oferece o melhor EFT e aquele que possui o menor tempo de execução absoluto.
- *Predict Earliest Finish Time* (PEFT): Abordagem que introduz o uso de ***Optimistic Cost Table*** (OCT) para refinar tanto o cálculo da prioridade das tarefas quanto a seleção dos recursos.
- *Improved Predict Earliest Finish Time* (IPEFT): Uma evolução do PEFT, porém caracterizada pela utilização de ***Pessimistic Cost Table*** (PCT) e pela aplicação de lógicas baseadas na identificação de nós críticos (*critical nodes*) no grafo.

Nas seções seguintes, são detalhados o funcionamento interno, as funções de avaliação e as diferenças arquiteturais que governam as decisões de escalonamento de cada uma dessas heurísticas.

3.2.1 *Dynamic Level Scheduling* (DLS)

O *Dynamic Level Scheduling* é uma heurística de escalonamento projetada para lidar com os altos custos de *interprocessor communication* (IPC) em arquiteturas heterogêneas (SIH; LEE, 1993). O DLS é um algoritmo que utiliza heurísticas baseadas em lista (*list-scheduling*) e avalia os pares de *tarefa-processador* a cada iteração (*dynamic*). Além de adotar uma política de não inserção (*non-insertion*).

A decisão de escalonamento no DLS é ditada pelo cálculo do *dynamic level* (DL) (SIH; LEE, 1993). Para ambientes heterogêneos, a prioridade de alocar o nó N_i no processador P_j é definida pela diferença entre o *static level* (SL) da tarefa e o seu EST no processador avaliado, somado ao delta (Δ) da tarefa no processador, formalmente representado por:

$$DL(N_i, P_j, \Sigma) = SL^*(N_i) - \max[DA(N_i, P_j, \Sigma), TF(P_j, \Sigma)] + \Delta(N_i, P_j) \tag{3.1}$$

Os componentes desta fórmula refletem um balanço entre três fatores (SIH; LEE, 1993),

- $SL^*(N_i)$: O *static level* é o maior caminho de N_i até o nó de saída. Ele garante que tarefas no início do caminho crítico tenham maior prioridade, ou seja, tarefas que estão mais distantes do nó terminal (*exit*).
- $\max[DA, TF]$: É a penalidade de tempo. $DA(N_i, P_j, \Sigma)$ representa o tempo mais cedo em que todos os dados necessários por N_i estarão disponíveis em P_j. $TF(P_j, \Sigma)$ é o tempo em que o processador P_j terminará sua última tarefa atribuída. O máximo entre esses dois valores dita o momento em que a tarefa pode, de fato, começar.
- $\Delta(N_i, P_j)$: Representa a diferença entre a mediana do tempo de execução da tarefa e o tempo real no processador P_j. Valores positivos altos indicam que P_j executa a tarefa N_i mais rápido que outros processadores. Esse componente adiciona um maior peso a processadores que executam a tarefa mais rápido e penaliza os que executam lentamente.

A cada iteração, o DLS calcula essa equação para todas as *ready tasks* e todos os processadores, selecionando o par (N_i, P_j) que maximiza o valor de DL.

A topologia do DAG afeta as decisões e a eficácia do DLS. Quando a topologia apresenta alto grau de paralelismo (muitas ramificações independentes), algoritmos que tendem a alocar as tarefas por todos os processadores disponíveis, podem gerar um gargalo de comunicação na rede. Já o DLS, por incorporar o custo de comunicação no fator DA, penaliza transferências desnecessárias. Como resultado, em DAGs com ramificações altamente dependentes, o DLS exibe um comportamento de "clusterização natural", agrupando nós de uma mesma ramificação em um único processador, reduzindo o IPC.

Além disso, a assimetria na transferência de dados no DAG impacta o algoritmo. Se um nó passa um alto volume de dados para um de seus descendentes, o DLS pode sacrificar a execução rápida do nó pai num processador específico, caso esse processador seja péssimo para o nó filho. Essa dependência topológica força o DLS a buscar uma solução que equilibra a comunicação entre processadores e o processamento das tarefas (SIH; LEE, 1993).

3.2.2 *Heterogeneous Earliest Finish Time* (HEFT)

O algoritmo *Heterogeneous Earliest Finish Time*, proposto por Topcuoglu, Hariri e Wu (1999), é uma heurística estática de escalonamento de DAGs projetada para otimizar o tempo de execução do *workflow* (minimizar o *makespan*). O HEFT é do tipo *list-scheduling*, ou seja, é estruturado em duas fases principais: priorização de tarefas e seleção de processador.

Uma característica importante do HEFT é a sua política de alocação baseada em inserção. Diferente de métodos que apenas alocam tarefas no final da fila de execução de um recurso, essa abordagem é capaz de vasculhar e achar *gaps* de ociosidade (*idle time slots*) entre duas tarefas já agendadas em um processador, desde que esse espaço seja grande o suficiente para acomodar a nova tarefa sem ferir as restrições de precedência.

Na fase de priorização, o HEFT classifica cada *task* do *workflow* calculando o seu *upward rank* ($rank_u$), que representa o comprimento do caminho crítico do n_1 até o nó de saída do grafo (n_{exit}), levando em consideração os custos de comunicação (TOPCUOGLU; HARIRI; WU, 1999). Formalmente definido por:

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} (\overline{c_{i,j}} + rank_u(n_j))$$

Onde $\overline{w_i}$ é o custo médio de execução do *node* n_i em todos os processadores, $succ(n_i)$ são as tarefas sucessoras, e $\overline{c_{i,j}}$ é o custo de comunicação entre n_i e n_j. Após o cálculo do $rank_u$ de todas as tarefas é escolhida aquela com maior valor.

Na fase de seleção, o *node* com a maior prioridade é alocada para o processador p_j que minimize o seu *Earliest Finish Time* (TOPCUOGLU; HARIRI; WU, 1999).

Calculado por:

$$EFT(n_i, p_j) = w_{i,j} + EST(n_i, p_j)$$

Onde $w_{i,j}$ é o custo de executar o *node* n_i no processador p_j, e o EST (2.3.1) determina o momento mais cedo em que a tarefa pode iniciar.

O desempenho do algoritmo HEFT é diretamente influenciado pela topologia do DAG, em grafos com baixa taxa de paralelismo (topologias profundas e de muitos *levels*), o HEFT demonstra superioridade em relação a outras abordagens clássicas, dado a sua priorização a partir de *nodes* que estão mais longe do *node* de saída. Para topologias mais densas (curtas, porém grandes em largura), o HEFT também mantém um desempenho de ponta, superando algoritmos mais custosos (TOPCUOGLU; HARIRI; WU, 1999).

A proporção de dados transferidos também molda a eficácia da heurística. Em cenários de alto *communication to computation ratio* (CCR) - onde um CCR alto indica que o DAG é intensivo em comunicação e baixo indicando que é intensivo em computação - o HEFT excede o desempenho de alternativas como o DLS. Porém, em cenários intensivos em computação - onde o tempo de comunicação é quase irrisório comparado com o custo de computação (baixo CCR, com CCR $<= 0.2$) - o DLS obtém uma melhor performance.

3.2.3 *Improved Heterogeneous Earliest Finish Time* (IHEFT)

O algoritmo *Improved Heterogeneous Earliest Finish Time* tem como proposta ser um aperfeiçoamento das heurísticas de escalonamento do tipo *static list-based*, tanto na fase de priorização de tarefas quanto na fase de seleção de processadores, e mantendo a mesma complexidade comparado ao HEFT e PEFT, a fim de minimizar o *makespan* (ALEBRAHIM; AHMAD, 2017b). O diferencial central do IHEFT é a melhoria da fase de seleção do processador empregando uma técnica de troca de processadores (*crossover*), baseado em um *threshold*, que é um valor randômico, e será comparado com o valor do *Cross Threshold*.

Em vez de sempre alocar a tarefa no processador com o menor EFT, como é de praxe no HEFT (detalhado em 3.2.2), o IHEFT avalia se é mais vantajoso optar pela otimização local (o processador com o menor tempo de execução da tarefa) em vez da otimização global (o menor EFT geral). Tal flexibilidade permite que o algoritmo explore o espaço de busca e as possibilidades de maneira mais eficiente.

Na fase de priorização, o IHEFT calcula um peso específico ($Weight_{n_i}$) para cada tarefa, que é definido pelo valor absoluto da razão entre a diferença de tempo do o maior e menor tempo de execução e o aumento de velocidade dos processadores considerados:

$$Weight_{n_i} = \left| \frac{w(n_i, p_j) - w(n_i, p_k)}{w(n_i, p_j)/w(n_i, p_k)} \right|$$

Esse peso é então utilizado na equação de *upward rank* (3.2.2) proposta:

$$rank_{proposed}(n_i) = Weight_{n_i} + \max_{n_j \in succ(n_i)} \{c_{i,j} + rank_{proposed}(n_j)\}$$

Na fase de seleção de processador, o algoritmo calcula um novo peso, o ($Weight_{abstract}$) baseado na diferença de EFTs (2.3.2) entre os processadores:

$$Weight_{abstract} = \left| \frac{EFT(n_i, p_j) - EFT(n_i, p_k)}{EFT(n_i, p_j)/EFT(n_i, p_k)} \right|$$

A decisão de realizar a troca do processador de menor tempo de execução (ótimo local) ao invés de executar no processador de menor EFT (ótimo global), é ditada pelo *Cross_Threshold*:

$$Cross\_Threshold = \frac{Weight_{n_i}}{Weight_{abstract}}$$

Se o valor de $Cross\_Threshold$ for menor ou igual a um número aleatório r (no intervalo de 0.1 a 0.3), a mudança de processador é aplicada, caso contrário, a tarefa permanece no processador original (ALEBRAHIM; AHMAD, 2017b).

A topologia e tamanho do grafo afetam os resultados do IHEFT de maneira distinta quando comparado aos algoritmos predecessores. O algoritmo apresenta um desempenho superior em DAGs com um número massivo de nós e com topologias menos paralelas (baixo fator *FAT*), porém, em DAGs com baixa quantidade de nós algoritmos como HEFT ou PEFT apresentam melhor performance (ALEBRAHIM; AHMAD, 2017b).

Além disso, a heurística produz os seus melhores resultados em grafos com altos valores de *communication to computation ratio* (CCR). Essa característica acontece porque, em cenários onde a transferência de dados é custosa, o mecanismo de limiar do IHEFT limita a transferência entre os processadores. Ao limitar as trocas, o algoritmo evita as pesadas penalidades de comunicação em rede, agrupando tarefas dependentes e reduzindo o *makespan* global (ALEBRAHIM; AHMAD, 2017b).

3.2.4 *Predict Earliest Finish Time* (PEFT)

O algoritmo PEFT foi desenvolvido com o propósito de superar algoritmos até então considerados como *state-of-the-art* no que diz respeito a *makespan* e eficiência (ARABNEJAD; BARBOSA, 2014). O PEFT introduz um mecanismo de antecipação (*look-ahead*) otimista sem aumentar a complexidade assintótica, que se mantém em $O(v^2 \cdot p)$, diferente

de algoritmos como LDCP e *Lookahead* que também apresentam o mesmo mecanismo, porém com maior complexidade assintótica (ARABNEJAD; BARBOSA, 2014). A técnica do *look-ahead* se destaca pois tem a capacidade de prever o impacto de um escalonamento para as tarefas filhas da tarefa escalonada, eliminando assim a miopia na seleção dos recursos e permitindo a heurística exercer melhores decisões ao escolher os processadores.

Para viabilizar essa visão de futuro, o algoritmo introduz a *Optimistic Cost Table* (OCT) mantendo a complexidade quadrática. Os valores da OCT representam o tempo máximo do caminho mais curto de uma tarefa t_i até a tarefa de saída t_{exit}. É classificado como "otimista" pois o cálculo assume que os processadores estarão sempre disponíveis, ignorando o tempo em que estarão ocupados por outras tarefas. A construção da OCT é feita recursivamente, da tarefa de saída t_{exit} até a tarefa de entrada t_{start}, e é definida pela seguinte equação:

$$OCT(t_i, p_k) = \max_{t_j \in succ(t_i)} \left[\min_{p_w \in P} \{ OCT(t_j, p_w) + w(t_j, p_w) + \bar{c}_{i,j} \} \right]$$

Onde $\bar{c}_{i,j}$ é o custo de comunicação entre t_i e t_j, sendo zero caso ambas as tarefas sejam alocadas no mesmo processador, ou seja, se $p_i = p_j$.

Na fase de priorização, o PEFT calcula a prioridade a partir do $rank_{oct}$, que é definido pela média dos valores da OCT da tarefa em todos os processadores e escolhe a tarefa que maximiza essa função para ser escalonada. Podemos definir formalmente o $rank_{oct}$ por:

$$rank_{oct}(t_i) = \frac{\sum_{k=1}^{p} OCT(t_i, p_k)}{p}$$

Já na seleção de processador o PEFT não busca minimizar apenas o EFT (2.3.2). Ele calcula o *Optimistic EFT* (O_{EFT}), que é a soma do custo de término da tarefa atual com o maior caminho até a tarefa de saída t_{exit}. Dessa forma, o PEFT também leva em consideração o impacto futuro da seleção desse processador, talvez não escolhendo o processador com o menor EFT para a tarefa t_i, mas preferindo atingir um menor tempo de execução total das tarefas nas próximas etapas ao levar em consideração o maior caminho até t_{exit}. O_{EFT} pode ser definido por:

$$O_{EFT}(t_i, p_j) = EFT(t_i, p_j) + OCT(t_i, p_j)$$

O algoritmo então aloca a tarefa t_i no processador p_j que minimize o valor da equação do O_{EFT}.

A topologia do *workflow* evidencia os pontos fortes da estratégia de *look-ahead* adotada pelo PEFT. Em DAGs complexos, algoritmos com técnicas gulosas de programação tendem a degradar sua performance pois, ao alocar uma tarefa em um processador

muito rápido agora, forçam os filhos dessa tarefa a lidarem com latências altíssimas de rede depois. Como o termo *OCT* penaliza antecipadamente escolhas de comunicação ruins, o PEFT se destaca em grafos com alta CCR, onde transferir dados é custoso (ARABNEJAD; BARBOSA, 2014).

Além disso, em DAGs com alta heterogeneidade - heterogeneidade é o conceito que define o quão diferente é o custo de computação de uma tarefa nos diferentes processadores - a visão otimista do futuro evita gargalos no caminho crítico, resultando em um *makespan* global frequentemente inferior ao de heurísticas clássicas de mesma complexidade (ARABNEJAD; BARBOSA, 2014).

3.2.5 *Improved Predict Earliest Finish Time* (IPEFT)

O algoritmo IPEFT é uma heurística de escalonamento que tem como objetivo performar melhor que outras heurísticas em lista que são semelhantes, como o HEFT e PEFT, mas mantendo a mesma complexidade de tempo de $O(v^2 \cdot p)$ (ZHOU et al., 2017). Diferentemente do PEFT o IPEFT adota uma postura mais conservadora na priorização e introduz mecanismos que visam proteger o caminho crítico do grafo. Esses mecanismos consistem em antecipar o início das tarefas críticas (que definem o limite do *makespan*) e priorizar a execução das tarefas que possuem o maior risco de degradar o tempo total.

Para alcançar tais melhorias, o algoritmo introduz dois novos conceitos principais: a criação de uma *Pessimistic Cost Table* (PCT) para a fase de priorização, e o uso de uma Tabela da *critical node cost table* (CNCT) na fase de seleção do processador.

Na fase de priorização, o IPEFT computa o PCT de um par (tarefa, processador). Cada elemento da PCT representa o tempo máximo do caminho mais longo dos sucessores de v_i até tarefa de saída v_{exit}, assumindo sempre o pior cenário, ou seja, que as tarefas sucessoras sejam alocadas nos processadores com o pior desempenho. A equação é dada por:

$$PCT(v_i, p_k) = \max_{v_j \in succ(v_i)} \left[\max_{p_m \in P} \{ PCT(v_j, p_m) + w_{j,m} + \bar{c}_{i,j} \} \right]$$

A prioridade da tarefa, o $rank_{PCT}$, é definida pela média dos valores da PCT em todos os processadores somada ao custo médio de execução ($\bar{w_i}$). O algoritmo assim escolhe a tarefa que maximiza o valor de $rank_{PCT}$. Dessa forma, tarefas que gerariam resultados piores caso fossem atrasadas recebem maior prioridade e são escalonadas primeiro:

$$rank_{PCT}(v_i) = \frac{\sum_{k=1}^{p} PCT(v_i, p_k)}{p} + \overline{w_i}$$

Na fase de seleção de processador, o IPEFT introduz uma nova condicional, verificando se a tarefa v_i é um *critical node parent* (CNP). A tarefa v_i é considerada um

CNP se ela não for crítica (não faz parte do caminho crítico), e tiver pelo menos um filho imediato que faz parte do caminho crítico. Sendo assim, é calculado o valor do EFT_{CNCT} para ser utilizado como parâmetro para a alocação da tarefa v_i no processador p_j:

$$EFT_{CNCT}(v_i, p_j) = \begin{cases} EFT(v_i, p_j), & \text{se } CNP(v_i) = \text{true} \\ EFT(v_i, p_j) + CNCT(v_i, p_j), & \text{caso contrário} \end{cases}$$

Se a tarefa v_i for um CNP, o algoritmo a aloca no processador que minimiza o seu término (somente EFT), garantindo assim que a tarefa pai termine mais rápido, e por conseguinte adiantando o EST do nó crítico sucessor, o que impacta diretamente na redução do *makespan* global.

Por outro lado, se v_i não for um CNP, a heurística leva em consideração o impacto da escolha do processador usando a CNCT. A CNCT representa o maior valor do caminho mais curto das tarefas críticas sucessoras de v_i até a tarefa de saída t_{exit}, se v_i tem pelo menos um *critical node*, senão, a heurística estende sua busca para o caminho mais curto de todas as tarefas filhas de v_i. É definido pela seguinte equação:

$$CNCT(v_i, p_k) = \begin{cases} \max\limits_{\substack{v_j \in succ(v_i) \\ v_j \in CNs}} \left[\min\limits_{p_m \in P} \{CNCT(v_j, p_m) + w_{j,m} + \overline{c_{i,j}}\} \right], & \text{se } \exists v_j \in succ(v_i) \wedge v_j \in CNs \\ \max\limits_{v_j \in succ(v_i)} \left[\min\limits_{p_m \in P} \{CNCT(v_j, p_m) + w_{j,m} + \overline{c_{i,j}}\} \right], & \text{caso contrário} \end{cases}$$

Por fim, a tarefa é alocada ao processador que minimizar o EFT_{CNCT}.

A topologia do *workflow* é explorada de maneira eficiente pelo IPEFT por meio da identificação dos nós críticos. Em DAGs que possuem caminhos críticos muito definidos, o IPEFT garante que as tarefas "pais" do caminho crítico sejam alocadas focando estritamente em terminar o mais rápido possível (minimizar apenas EFT), destravando o caminho crítico antecipadamente.

Além disso, a abordagem pessimista atribui ao IPEFT um nível de robustez significativamente maior em topologias não previsíveis. O trabalho do IPEFT (ZHOU et al., 2017), demonstrou que em grafos com alta heterogeneidade computacional e alta CCR (CCR ≥ 0.8), o IPEFT supera consistentemente tanto o HEFT quanto o PEFT. No que diz respeito ao tamanho do DAG (quantidade de *tasks*) o IPEFT apresenta um maior decremento do SLR em DAGs menores, com a taxa de melhoria diminuindo à medida que o DAG aumenta, porém sempre performando melhor que algoritmos como HEFT e PEFT, esses resultados mostram que o efeito das *tasks* filhas críticas no processo de escalonamento das *tasks* pai diminui à medida que o número de *tasks* aumenta.

3.3 Formato dos DAGs

Para descrever e modelar as instâncias de execução de *workflows* científicos de forma padronizada e reprodutível, foi utilizado o formato *WfFormat* (2024b). O *WfFormat* é o esquema JSON oficial do projeto *WfCommons* (2022), atualmente em sua versão 1.5, desenvolvido para representar descrições e instâncias de execução de *workflows* científicos.

A utilização desse formato permite que as instâncias (representadas em arquivos `.json`) forneçam não apenas a topologia do DAG - definindo as tarefas e as arestas - mas também uma gama de metadados fundamentais para garantir a fidelidade do simulador (3.1) de escalonamento. Dentre as propriedades representadas pelo esquema, destacam-se para o propósito desta monografia:

- **Propriedades da Infraestrutura** (CPU): Detalhamento da arquitetura utilizada para capturar as informações de execução da tarefa, incluindo o número de núcleos alocados (`coreCount`), a velocidade de processamento (`speedInMHz`).
- **Métricas de Execução de Tarefas**: Informações de desempenho coletadas para cada *node* individual do grafo. Isso inclui métricas essenciais para alimentar o Simulador (3.1), como o tempo que a *task* levou para ser concluída (`runtimeInSeconds`), a taxa média de utilização do processador durante esse processamento (`avgCPU`) e a quantidade de cores consumidos pela tarefa *coreCount*.

3.3.1 Repositório de DAGs (WfInstances)

Para que as heurísticas de escalonamento sejam avaliadas de forma condizente com os desafios computacionais do mundo real, este trabalho extrai do repositório de instâncias *WfInstances* (2024a) os DAGs utilizados nas análises. Esse repositório atua como um catálogo de execuções de *workflows*, capturados durante a execução de aplicações científicas reais.

Os arquivos de instância contêm o comportamento histórico de diversas aplicações científicas, armazenando as características de entrada/saída (E/S), os volumes de transferência de dados reais e os tempos de processamento de cada tarefa. Ao importar *workflows* do *WfInstances*, garante-se que os cenários avaliados e os resultados calculados neste trabalho reflitam diretamente as complexidades, as assimetrias e os gargalos de comunicação inerentes às topologias de problemas científicos reais.

3.3.1.1 *1000genome*

O projeto *1000genome* (2015) fornece uma referência em variação genética humana que reconstruiu o genoma de milhares de indivíduos em diferentes populações. Já o

1000genome Workflow (A et al., 2015), que foi utilizado, tem como propósito identificar sobreposições mutacionais utilizando dados do projeto *1000genome*.

A estrutura desse *workflow* apresenta um grau de paralelismo bem segmentado: cada cromossoma processado define uma ramificação independente (um *branch*) no DAG. Exibe um padrão claro de *scatter-gather*, como mostrado na figura 2.

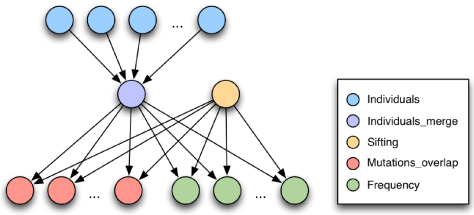


Figura 2 – Topologia do grafo *1000genome*

Fonte: https://github.com/wfcommons/WfInstances/tree/main/pegasus/1000genome

A tarefa `individuals` avalia exatamente 1000 sequências. Todas essas tarefas convergem obrigatoriamente para uma única tarefa `individuals_merge` por ramificação. A fase final da ramificação espalha-se novamente, com o número de tarefas `mutation_overlap` e de cálculo de frequências.

3.3.1.2 *Burrows-Wheeler Aligner* (BWA)

O BWA é um pacote de software usado para mapear sequências de baixa divergência contra um grande genoma de referência, como o genoma humano. Já o objetivo principal do BWA *Workflow* é executar as tarefas do BWA em um tempo computacional razoável.

O DAG do BWA exibe um padrão clássico de *scatter-gather*, representado na figura 3.

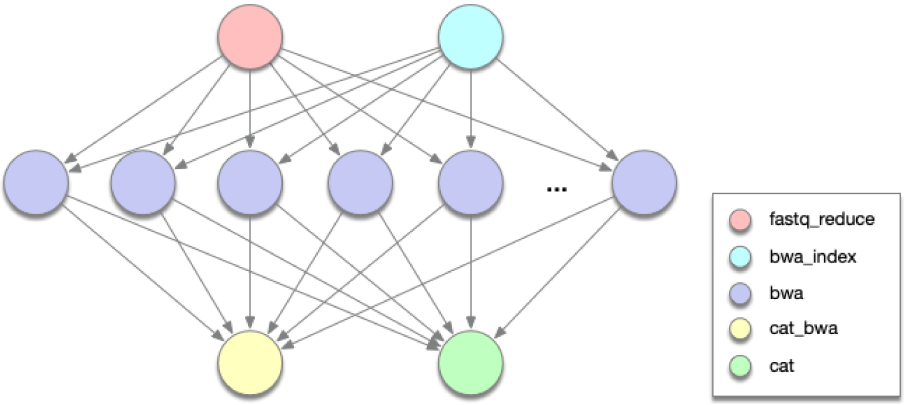


Figura 3 – Topologia do grafo BWA

Fonte: https://github.com/wfcommons/WfInstances/tree/main/makeflow/bwa

Começa com tarefas de preparação, `fastq_reduce` e a `bwa_index`. Em seguida, o processamento dispersa-se em múltiplas tarefas `bwa` que executam em paralelo. Por fim, as execuções paralelas convergem para duas tarefas: a tarefa `cat_bwa`, responsável por fazer o *merge* dos resultados de cada execução, e a tarefa `cat`, que agrupa todas as mensagens de erro geradas.

3.3.1.3 *Epigenomics*

O *workflow Epigenomics* é um *pipeline* de processamento de dados utilizado para executar operações de sequenciamento de genomas (JUVE et al., 2013). O seu objetivo é processar dados de sequências de DNA (gerados pelo sistema *Illumina-Solexa*), convertendo formatos, filtrando dados e mapeando as sequências de um genoma, permitindo analisar o DNA.

O fluxo de execução de cada ramificação começa com uma única tarefa `fastqSplit`. Em seguida, o processamento dispersa-se em múltiplas *pipelines* de execução de diferentes tipos de tarefas. As *pipelines* de uma mesma *branch* convergem para uma tarefa local `mapMerge`. Por fim, as execuções de todas as *branches* do DAG convergem para uma tarefa `mapMerge` global, que agrupa todos os dados num único arquivo de saída. Essa topologia é ilustrada na figura 4.

3.3.1.4 *Montage*

O *Montage* é um conjunto de ferramentas utilizadas para montar imagens astronómicas *flexible image transport system* (FITS) em mosaicos personalizados. Já o objetivo principal do *Montage Workflow* é processar as imagens de entrada, re-projetando-as, corrigindo o plano de fundo e combinando-as para criar um mosaico final (ADHIKARI; AMGOTH; SRIRAMA, 2019).

Começa com uma grande dispersão em múltiplas tarefas `mProject` e `mBackground` em cada *branch*. Em seguida, geram-se múltiplas tarefas `mDiffFit` que executam em paralelo. Por fim, as execuções paralelas convergem para uma linha sequencial de tarefas únicas: `mConcatFit`, `mBgModel`, múltiplas `mBackground` em paralelo, `mImgtbl`, `mAdd` e `mViewer`, que agrega e consolida os resultados. A topologia é ilustrada na figura 5.

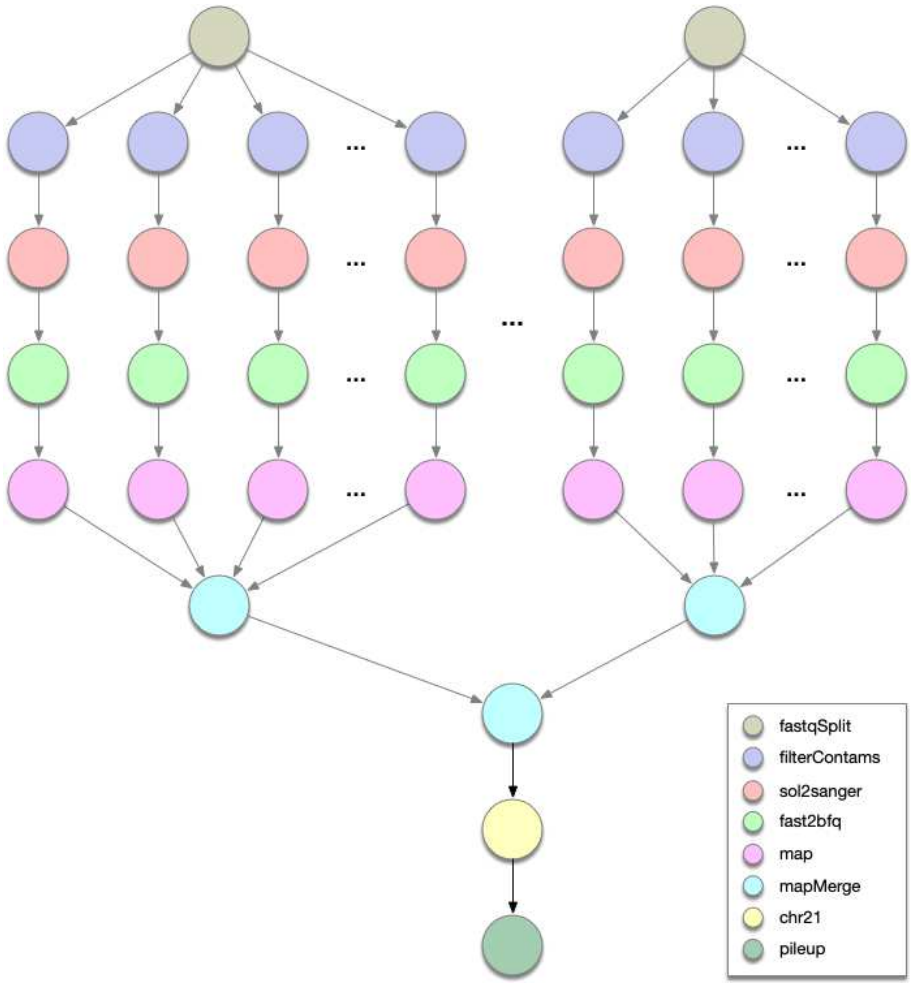


Figura 4 – Topologia do grafo *Epigenomics*

Fonte: https://github.com/wfcommons/WfInstances/tree/main/pegasus/epigenomics

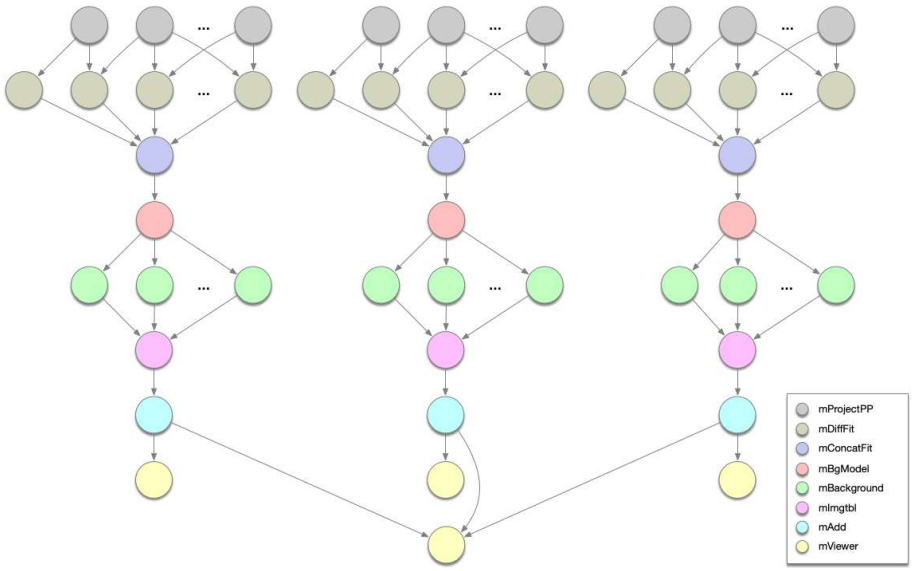


Figura 5 – Topologia do grafo *Montage*

Fonte: https://github.com/wfcommons/WfInstances/tree/main/pegasus/montage

4 Resultados e Análise

Este capítulo tem como objetivo apresentar, analisar e discutir os resultados obtidos a partir das simulações de escalonamento. Como mencionado na metodologia deste trabalho, a avaliação das heurísticas selecionadas (HEFT, IHEFT, PEFT, IPEFT e DLS) foi realizada utilizando instâncias reais de *workflows* científicos (*1000genome*, BWA, *Epigenomics* e *Montage*). O uso de DAGs extraídos de aplicações do mundo real garante que o comportamento dos algoritmos seja observado a partir de topologias, gargalos de comunicação e padrões de dependência de ambientes reais.

Para garantir uma análise completa, o experimento foi desenhado variando a escala da infraestrutura simulada. Especificamente, cada DAG foi submetido sob quatro cenários distintos de paralelismo, compostos por conjuntos de $P = \{4, 8, 16, 32\}$ processadores heterogêneos. Além disso, o experimento utilizou alguns DAGs disponibilizados pelo repositório *WfInstances* referentes aos *workflows* selecionados. O repositório expõe uma vasta gama de instâncias para cada *workflow*, abrangendo grafos com poucos nós, estruturas massivas contendo milhares de nós e até grafos com diferentes topologias e dependências. Essa diversidade permite investigar não apenas a natureza das aplicações, mas também mensurar como o simulador e os algoritmos reagem à escalabilidade do problema.

A seguir, as métricas extraídas dessas simulações são detalhadas qualitativa e quantitativamente. Avaliou-se o impacto da escala sobre as métricas pré-estabelecidas — *makespan*, *scheduling length ratio*, *load balancing* e *waiting time* —, traçando um comparativo entre os algoritmos para cada uma das aplicações científicas.

4.1 Análise do *Workflow*: *1000genome*

O *workflow 1000genome* caracteriza-se por um forte padrão de *scatter-gather*, no qual um alto volume de processamento paralelo é afunilado em poucas tarefas (*merge*). Esse tipo de topologia impõe desafios ao escalonamento, especialmente no que diz respeito a ociosidade dos recursos enquanto aguardam a conclusão de tarefas dependentes.

Para avaliar o comportamento das heurísticas frente a essa topologia, foram analisadas duas instâncias: uma de menor escala (Instância A, com 246 tarefas) e uma de larga escala (Instância B, com 902 tarefas).

4.1.1 Avaliação em Cenário de Menor Escala (246 Tarefas)

A Figura 6 ilustra a topologia da instância menor. Nota-se a presença de ramificações paralelas que rapidamente convergem para poucas tarefas (*merge*).

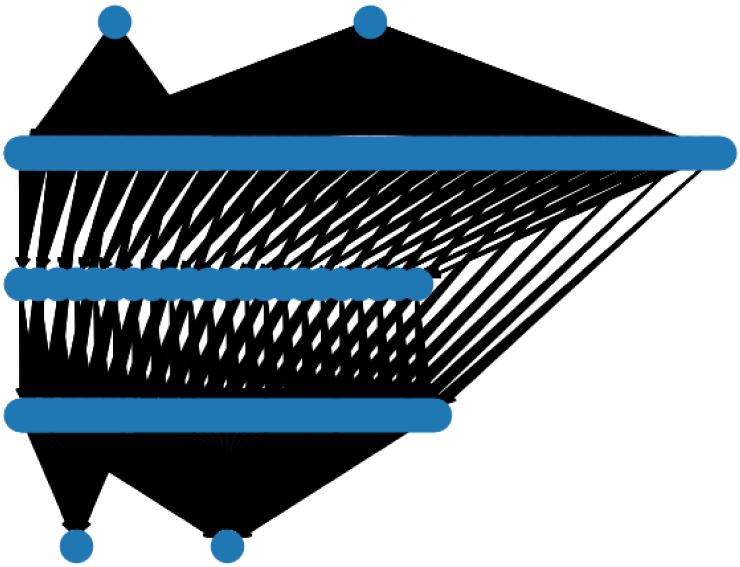


Figura 6 – Ilustração em alto nível da topologia da instância de 246 tarefas do grafo *1000genome*

Os resultados das simulações para esta instância, variando o conjunto de processadores $P \in \{4, 8, 16, 32\}$, estão consolidados nos gráficos da figura 7.

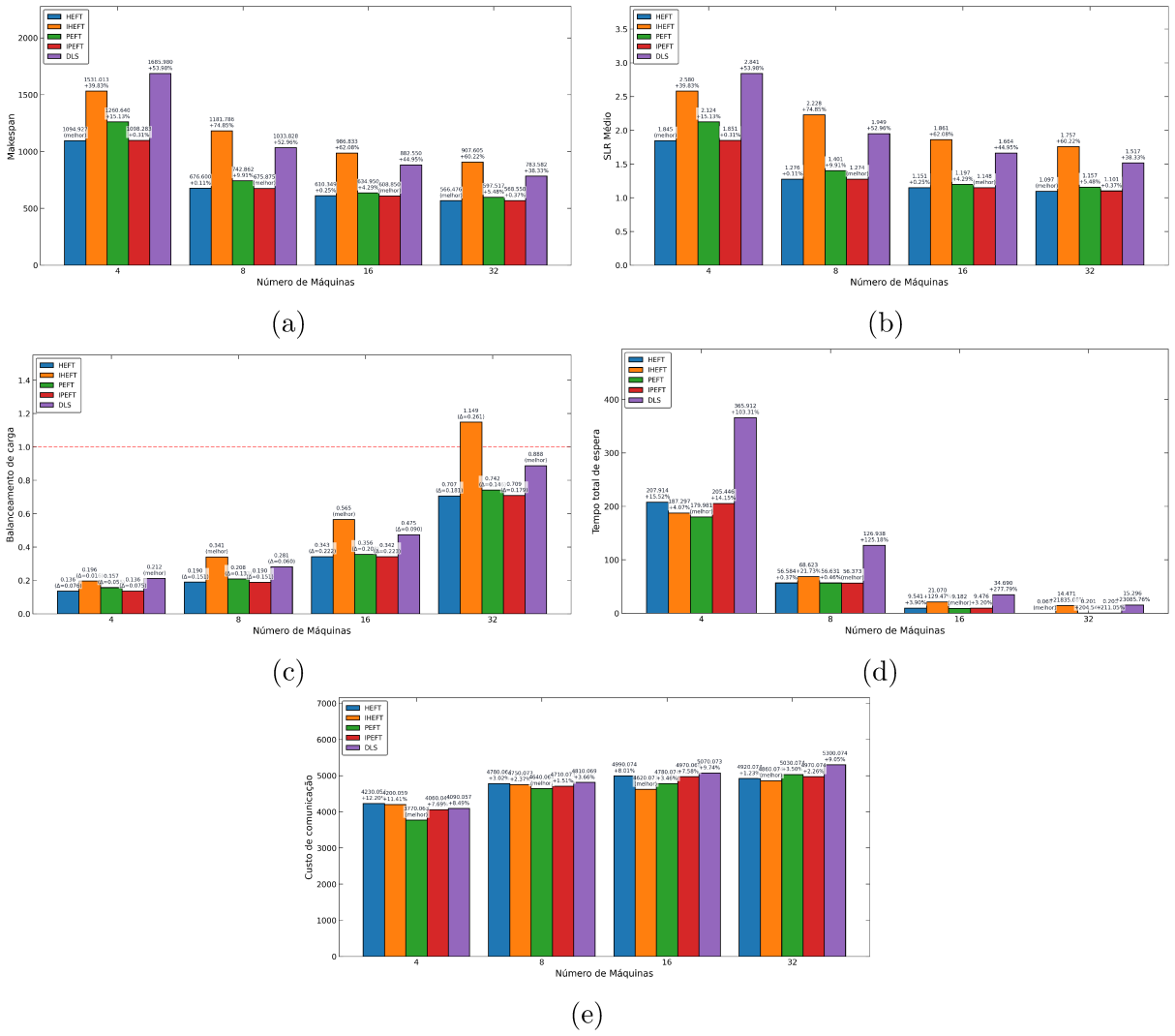


Figura 7 – Resultados - (a) *makespan* / (b) SLR / (c) LB / (d) WT / (e) CC - para o grafo *1000genome* (246 tarefas).

4.1.1.1 Análise de Desempenho (*Makespan* e SLR)

Ao escalar a infraestrutura de $P = 4$ para $P = 16$ processadores, observa-se uma redução acentuada no *makespan* e no SLR para algoritmos com política de inserção, destacando-se o HEFT e o IPEFT, que obtiveram resultados quase que idênticos (≈ 610 e ≈ 608, respectivamente). No entanto, ao atingir 32 máquinas, a curva de ganho de performance tende a um platô, com o *makespan* reduzindo apenas marginalmente (≈ 566). Isso ocorre porque o padrão *scatter-gather* atinge o limite do paralelismo intrínseco do grafo; adicionar mais capacidade computacional não reduz o tempo total, visto que as tarefas sequenciais de *merge* impõem um limite inferior de execução, um reflexo prático da Lei de Amdahl (GUSTAFSON, 2011).

4.1.1.2 Utilização de Recursos e Filas (LB e WT)

O WT reflete diretamente a eficiência das políticas de alocação. O algoritmo DLS obteve os piores resultados de espera (atingindo picos de $\approx 365s$ no cenário de 4 máquinas). Por utilizar uma política estrita sem inserção (*non-insertion*), o DLS é incapaz de aproveitar os *gaps* de ociosidade gerados pelas tarefas de *merge*, resultando em grandes filas. À medida que o número de máquinas aumenta para 32, o *load balancing* de todos os algoritmos sofre uma redução natural, evidenciando que grande parte dos recursos permanece ociosa aguardando a finalização de tarefas dependentes.

4.1.1.3 Penalidade de Comunicação (CC)

Ao analisarmos o custo de comunicação, nota-se que o aumento de recursos estimula heurísticas como o HEFT a dispersarem as tarefas entre os processadores (aumentando assim o custo de comunicação). Curiosamente, o PEFT obteve o menor custo de comunicação no cenário de $P = 4$, mostrando que sua técnica de *look-ahead* (com OCT) conseguiu agrupar tarefas dependentes de forma mais eficiente, mitigando o *overhead* de rede sem sacrificar tanto o *makespan*.

4.1.2 Avaliação em Cenário de Larga Escala (902 Tarefas)

Para a instância massiva, a complexidade estrutural e a quantidade de dados transitados destacam as falhas e virtudes de cada heurística. A Figura 8 ilustra a topologia da instância maior.

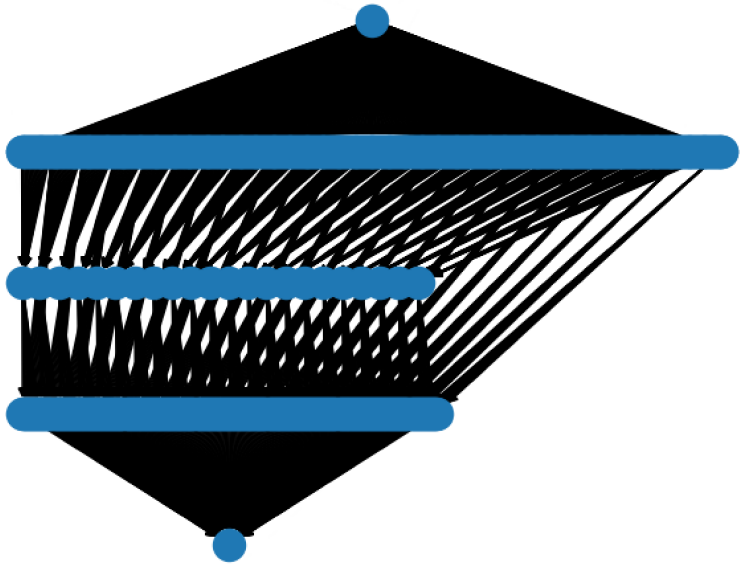


Figura 8 – Ilustração em alto nível da topologia da instância de 902 tarefas do grafo *1000genome*

Os resultados das simulações para esta instância, variando o conjunto de processadores $P \in \{4, 8, 16, 32\}$, estão consolidados nos gráficos da figura 9.

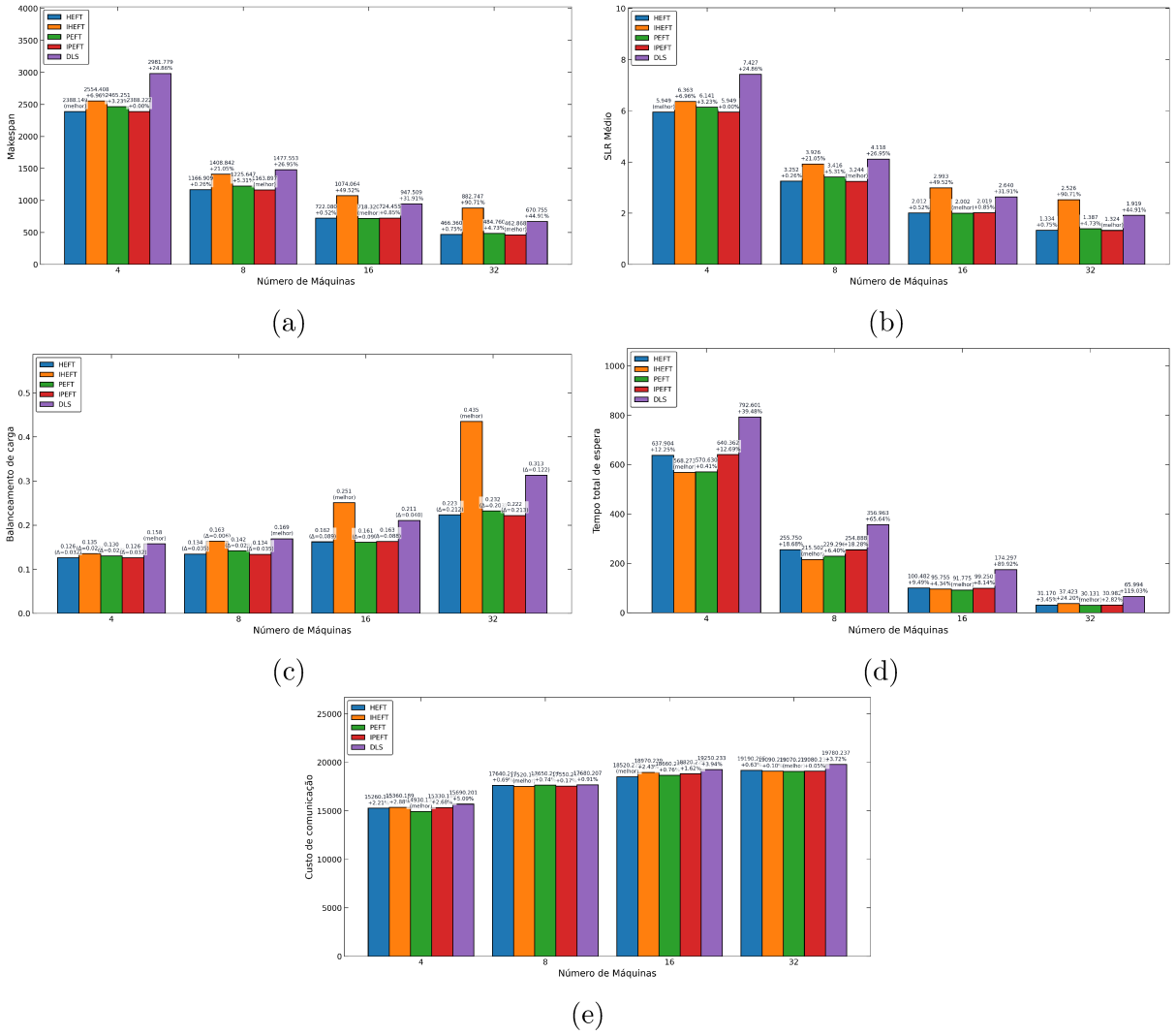


Figura 9 – Resultados - (a) *makespan* / (b) SLR / (c) LB / (d) WT / (e) CC - para o grafo *1000genome* (902 tarefas).

Nessa escala, a proteção do caminho crítico proporcionada pelo IPEFT mostra o seu valor. Enquanto no grafo menor o HEFT e o IPEFT apresentaram desempenhos similares, na escala de 902 tarefas e com $P = 32$, o IPEFT assumiu a liderança com o menor *makespan* (≈ 462) e o menor SLR (≈ 1.32). A introdução da Tabela PCT aliada à *critical node parent* (CNP), garantiu que tarefas essenciais fossem executadas com maior prioridade.

Por outro lado, o IHEFT, que introduz trocas randômicas para evitar ótimos locais, sofreu uma severa degradação de desempenho em relação à sua base (HEFT). Com $P = 32$, o IHEFT obteve um *makespan* de ≈ 882 (quase o dobro do melhor), indicando que o mecanismo de limiar para a mudança de processador, embora útil em alguns cenários, falhou em distribuir adequadamente as 902 tarefas.

4.2 Análise do *Workflow*: BWA

O *workflow burrows-wheeler aligner* (BWA) exibe um padrão clássico de *scatter-gather*, iniciando com tarefas sequenciais que se dispersam em múltiplas tarefas independentes e paralelas, antes de convergirem novamente para uma tarefa consolidação dos resultados. Esse afunilamento de tarefas testa a capacidade dos algoritmos de mitigar a ociosidade dos recursos.

Para avaliar o comportamento das heurísticas frente a essa topologia, analisamos duas instâncias: uma de menor escala (Instância A, com 104 tarefas) e uma de larga escala (Instância B, com 1004 tarefas).

4.2.1 Avaliação em Cenário de Menor Escala (104 Tarefas)

A Figura 10 ilustra a topologia da instância menor. Veja a dependência das tarefas paralelas em relação aos poucos nós iniciais e finais.

Figura 10 – Ilustração em alto nível da topologia da instância de 104 tarefas do grafo BWA

Os resultados das simulações para esta instância, variando o conjunto de processadores $P \in \{4, 8, 16, 32\}$, estão consolidados nos gráficos da figura 11.

4.2.1.1 Análise de Desempenho (*Makespan* e SLR)

No grafo de 104 tarefas, a saturação do paralelismo ocorre de forma muito prematura. Ao observar o *makespan*, verifica-se que o ganho ao escalar de 4 para 8 máquinas é razoável (de ≈ 890 para ≈ 761 no IPEFT), porém, saltar de 16 para 32 máquinas resulta em uma redução extremamente baixa, de ≈ 748 para ≈ 727. O IPEFT e o HEFT lideram

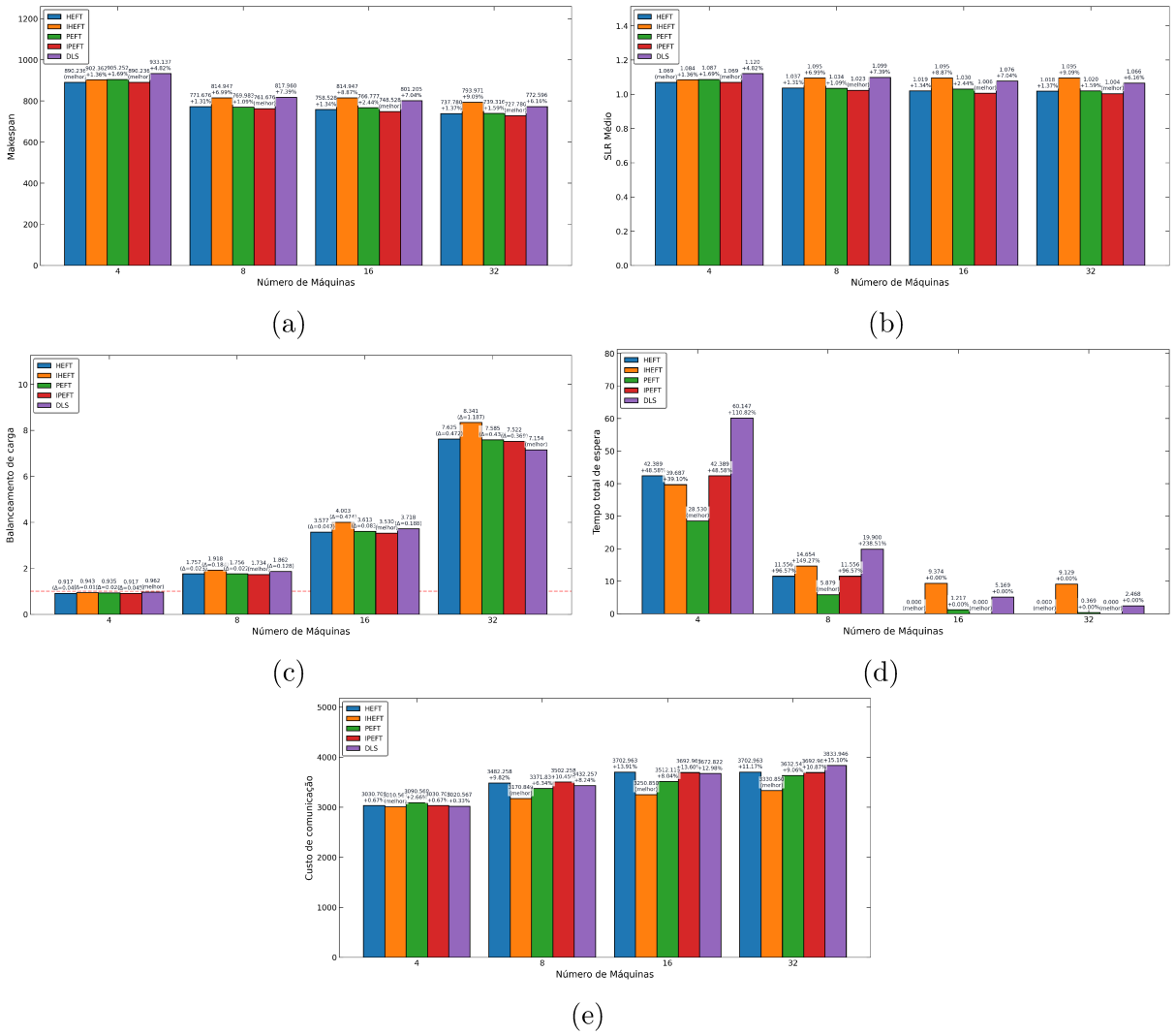


Figura 11 – Resultados - (a) *makespan* / (b) SLR / (c) LB / (d) WT / (e) CC - para o grafo BWA (104 tarefas).

com facilidade, empurrando o SLR para valores muito próximos do limite ótimo teórico (1.004 para o IPEFT em 32 máquinas).

4.2.1.2 Utilização de Recursos e Filas (LB e WT)

A métrica de *waiting time* destaca uma característica muita importante do PEFT. No cenário com baixa quantidade de processadores ($P = 4$), o PEFT obteve o menor tempo de espera absoluto ≈ 28, superando com folga o DLS (≈ 60) e o próprio HEFT (≈ 42). A técnica de *look-ahead* introduzida pela matriz OCT permite ao PEFT evitar que tarefas fiquem bloqueadas desnecessariamente em processadores gargalados. Conforme o número de recursos cresce para 32 máquinas, o *load balancing* infla severamente (ultrapassando 7.0), refletindo a extrema ociosidade gerada pela ausência de tarefas para alimentar todos os processadores simultaneamente.

4.2.1.3 Penalidade de Comunicação (CC)

Diferentemente do *makespan*, onde o IHEFT apresenta péssimo desempenho, no custo de comunicação (CC) ele se destaca sendo o melhor resultado dentre os algoritmos. Devido ao *threshold* aleatório para executar o *crossover* de tarefas entre máquinas diferentes. Em 32 máquinas, ele obteve o menor custo de rede ($\approx$ 3330), enquanto outras abordagens que tentam minimizar o *makespan* por dispersar mais as tarefas, como o DLS e o IPEFT, alcançando valores superiores a $\approx$ 3690.

4.2.2 Avaliação em Cenário de Larga Escala (1004 Tarefas)

Aumentar a escala da aplicação para 1004 tarefas acentua os desafios de otimização interprocessos. A Figura 12 ilustra a topologia.

Figura 12 – Ilustração em alto nível da topologia da instância de 1004 tarefas do grafo BWA

Os resultados das simulações para esta instância, variando o conjunto de processadores $P \in \{4, 8, 16, 32\}$, estão consolidados nos gráficos da figura 13.

4.2.2.1 Análise de Desempenho (*Makespan* e SLR)

A drástica mudança de escala fortalece a robustez do IPEFT. Em todas as infraestruturas, o IPEFT assumiu o protagonismo, entregando o menor *makespan* do experimento e um SLR extremamente próximo do ótimo, superando o HEFT e deixando o IHEFT como o pior algoritmo de escalonamento para o cenário de $P = 32$. O IPEFT mostra sua capacidade de manter o desempenho à medida que o DAG cresce, onde a tabela PCT guia as tarefas filhas críticas sem saturar o sistema.

4.2.2.2 Utilização de Recursos e Filas (LB e WT)

Novamente, o PEFT demonstra excelência na gerência do *waiting time*. Enquanto heurísticas como o DLS deixaram tarefas aguardando por grandes quantidades de tempo

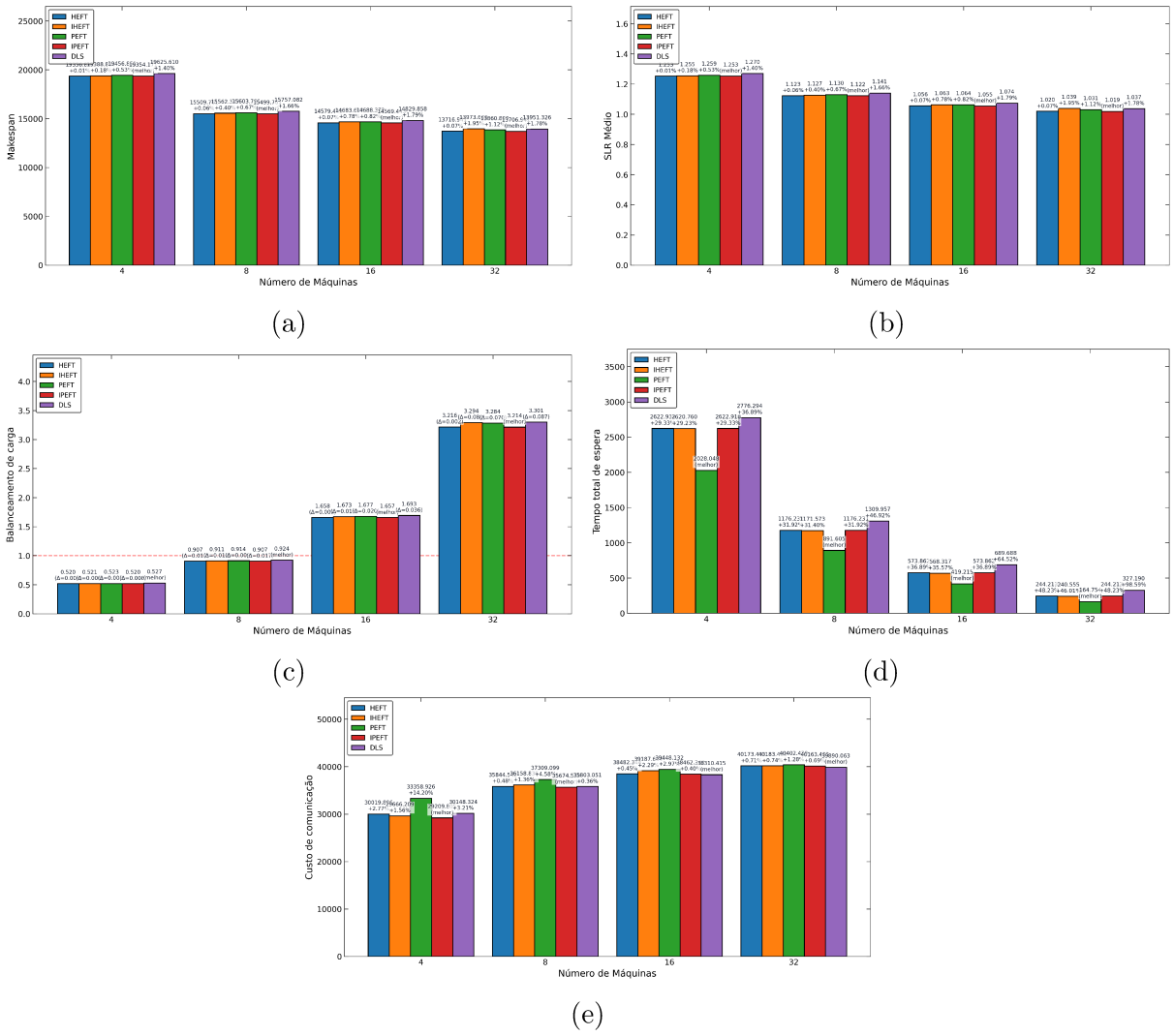


Figura 13 – Resultados - (a) *makespan* / (b) SLR / (c) LB / (d) WT / (e) CC - para o grafo BWA (1004 tarefas).

em $P = 4$, e o HEFT, IPEFT e IHEFT tiveram resultados parecidos nesse quesito, o PEFT reduziu a espera absoluta em todos os cenários. Essa margem expressiva consolida a teoria de que o *look-ahead* do PEFT toma decisões mais saudáveis para a otimização global do sistema, mesmo que, na etapa final de consolidação, o IPEFT ainda retome a liderança no tempo de *makespan*.

4.2.2.3 Penalidade de Comunicação (CC e LB)

Devido à base de 1004 tarefas, a transferência de dados pesados atinge seus maiores picos. O DLS apresenta comportamentos opostos: por um lado entrega péssimo gerenciamento de *waiting time* no ambiente de $P = 4$ e $P = 8$, por outro ele consegue um dos melhores valores de *communication cost* no ambiente de $P = 16$ e $P = 32$ máquinas, reflexo da sua função DA que penaliza dependência distantes e *"clusteriza"* tarefas dependentes na mesma máquina. Além disso, o LB sofre uma escalada contínua assim como

a instância menor, o que reflete um paralelismo melhor aproveitado.

4.3 Análise do *Workflow*: *Epigenomics*

O *workflow Epigenomics* é caracterizado por múltiplas *branches* independentes que executam diversas subtarefas em paralelo, convergindo para consolidações locais e, por fim, para uma consolidação global.

Para avaliar o comportamento das heurísticas frente a essa topologia, analisamos duas instâncias: uma de menor escala (Instância A, com 223 tarefas) e uma de larga escala (Instância B, com 1695 tarefas).

4.3.1 Avaliação em Cenário de Menor Escala (223 Tarefas)

A Figura 14 ilustra a topologia da instância menor. Nota-se a clara divisão em *branches* independentes de tarefas que processam faixas distintas de dados antes do *merge*.

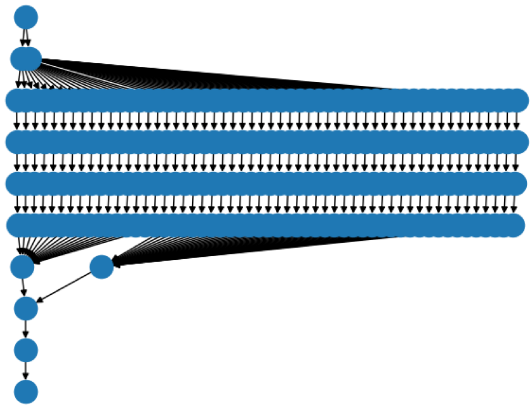


Figura 14 – Ilustração em alto nível da topologia da instância de 223 tarefas do grafo *Epigenomics*

Os resultados das simulações para esta instância, variando o conjunto de processadores $P \in \{4, 8, 16, 32\}$, estão consolidados nos gráficos da figura 15.

4.3.1.1 Análise de Desempenho (*Makespan* e SLR)

A topologia do *Epigenomics* permite um escalonamento eficiente quando há disponibilidade de recursos. No cenário mais restrito ($P = 4$), o HEFT e o IPEFT lideram com os melhores *makespan*s. Ao escalar para 32 máquinas, o IPEFT se sobressai marginalmente (≈ 194) contra o PEFT (≈ 196) e o HEFT (≈ 201). Já o SLR aproxima de 1.22 no IPEFT, demonstrando que, as heurísticas baseadas em caminhos críticos conseguem otimizar quase perfeitamente a distribuição das *branches* em processadores distintos.

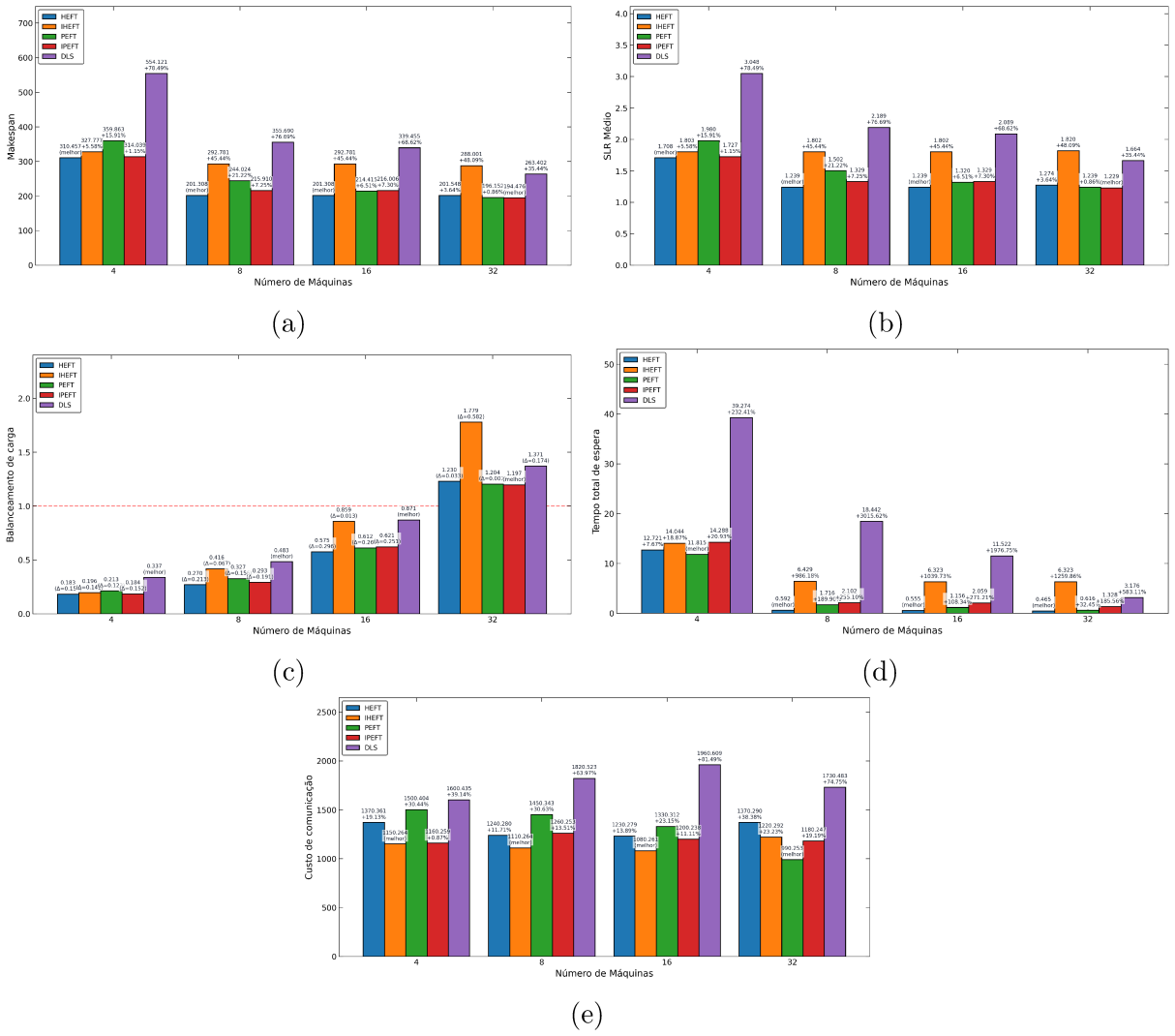


Figura 15 – Resultados - (a) *makespan* / (b) SLR / (c) LB / (d) WT / (e) CC - para o grafo *Epigenomics* (223 tarefas).

4.3.1.2 Utilização de Recursos e Filas (LB e WT)

Com $P = 4$, a política de *non-insertion* penaliza severamente o DLS. Nesse tamanho de grafo o tempo total de espera do DLS é quase 4 vezes maior do que algoritmos como PEFT (melhor resultado) e HEFT. Em $P = 32$, o *load balancing* atinge valores de ≈ 1.2, um valor substancialmente melhor (mais bem distribuído) se comparado às topologias de *scatter-gather* puras (como 7c e 11c), demonstrando que as *branches* do *Epigenomics* mantêm os processadores ocupados por mais tempo.

4.3.1.3 Penalidade de Comunicação (CC)

Em cenários de limitação de recursos ($P = 4$), o IHEFT apresenta o menor custo de comunicação, evitando penalidades desnecessárias de rede ao não escalonar tarefas aleatoriamente. Contudo, o PEFT e o DLS exibiram os maiores custos de rede nesse cenário, sugerindo uma tendência a espalhar tarefas das *branches* de sequenciamento em

busca do menor custo computacional (otimização local), sacrificando a localidade dos dados.

4.3.2 Avaliação em Cenário de Larga Escala (1695 Tarefas)

Ao lidar com 1695 tarefas, o *workflow* atinge grandes profundidades das *branches* geradas. A Figura 16 ilustra a expansão dessa topologia.

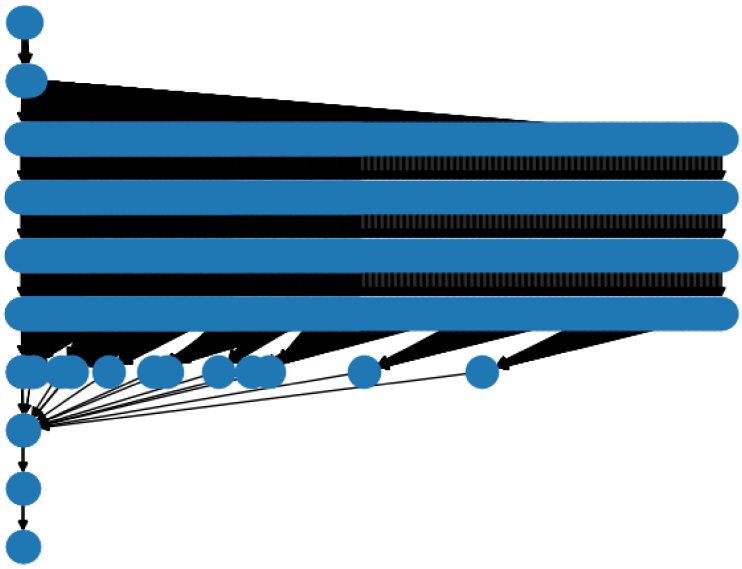


Figura 16 – Ilustração em alto nível da topologia da instância de 1695 tarefas do grafo *Epigenomics*

Os resultados das simulações para esta instância, variando o conjunto de processadores $P \in \{4, 8, 16, 32\}$, estão consolidados nos gráficos da figura 17.

4.3.2.1 Análise de Desempenho (*Makespan* e SLR)

A performance com escala massiva destaca um empate entre o HEFT e o IPEFT. Com $P = 32$, ambos atingiram *makespan*s quase idênticos e entregando um SLR quase ideal de ≈ 1.01. Essa eficiência indica que os cálculos de *upward rank* (do HEFT) e PCT (do IPEFT) funcionam bem com a profundidade do *Epigenomics*, permitindo alocar as subtarefas no melhor momento. Já o DLS, conseguiu recuperar competitividade, em ambientes com abundância de processadores, com um SLR de ≈ 1.07 (superando o IHEFT), constatando que grafos com forte paralelismo entre as *branches* mascaram parcialmente sua política baseada em não inserção (*non-insertion*).

4.3.2.2 Utilização de Recursos e Filas (LB e WT)

A diferença no controle de ociosidade é enorme em cenários com poucos recursos ($P = 4$). Enquanto HEFT, IHEFT, PEFT e IPEFT mantêm o *waiting time* estabilizado

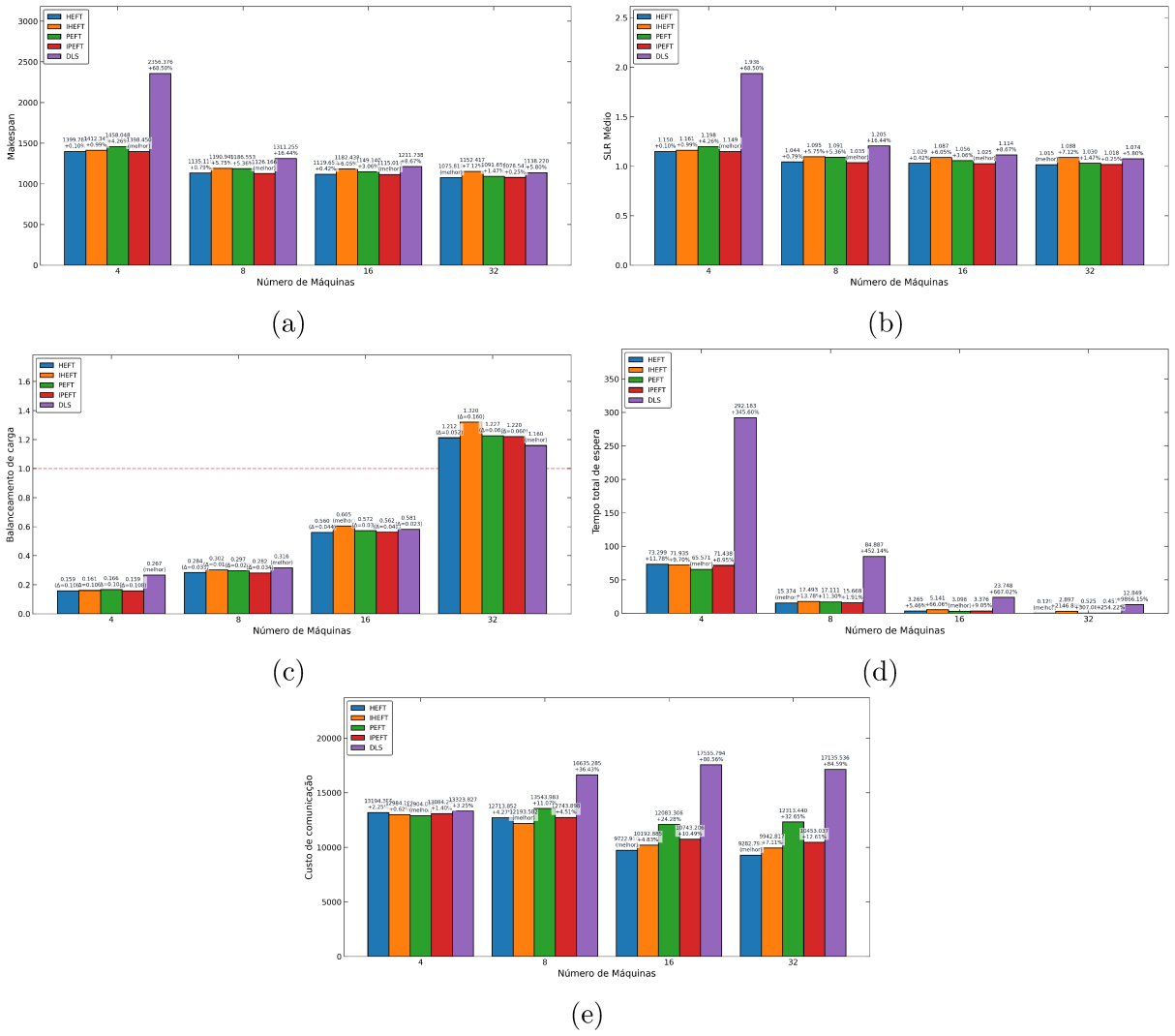


Figura 17 – Resultados - (a) *makespan* / (b) SLR / (c) LB / (d) WT / (e) CC - para o grafo *Epigenomics* (1695 tarefas).

entre ≈ 65 e ≈ 73, o DLS pune o *workflow* com um *waiting time* mais de 4 vezes maior. No entanto, quando há abundância de recursos ($P = 32$), algoritmos como o HEFT conseguem praticamente zerar o WT, demonstrando uma alocação imediata nos processadores.

4.3.2.3 Penalidade de Comunicação (CC e LB)

No que diz respeito ao *communication cost*, o HEFT e o IHEFT mostraram-se as melhores soluções para ambientes com $P = 32$, gerando os menores CC (com o HEFT se sobressaindo), porém, em ambientes com $P = 4$, o PEFT se sobressai dentre os algoritmos. Em contrapartida, o DLS na busca de redução do *makespan* falha estruturalmente ao dispersar os *branches* excessivamente, alcançando os piores valores de CC. Essa métrica prova que as heurísticas clássicas de escalonamento sem usar *look-ahead* ou sem priorizar a otimização global pode saturar o sistema, principalmente em sistemas reais baseadas em nuvem onde o custo de comunicação é uma métrica financeira e de performance sensível.

4.4 Análise do *Workflow*: *Montage*

O *workflow Montage* é uma aplicação astronômica utilizada para gerar mosaicos de imagens. Topologicamente, esse DAG divide-se em três grandes ramificações (*branches*) principais. Cada ramificação possui dezenas de tarefas em paralelo inicialmente, mas que obrigatoriamente convergem para uma sequência de tarefas de consolidação.

Para avaliar o comportamento das heurísticas frente a essa topologia, analisamos duas instâncias: uma de menor escala (Instância A, com 178 tarefas) e uma de larga escala (Instância B, com 4846 tarefas).

4.4.1 Avaliação em Cenário de Menor Escala (178 Tarefas)

A Figura 18 ilustra a topologia da instância menor. Note-se a alta densidade no topo do grafo, seguida pelo estreitamento severo na base.

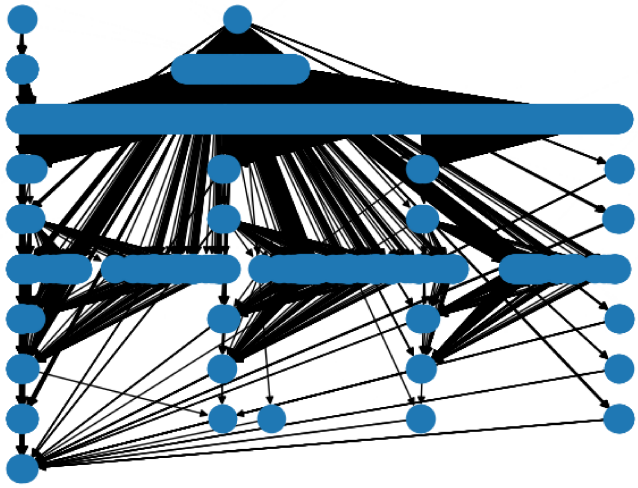


Figura 18 – Ilustração em alto nível da topologia da instância de 178 tarefas do grafo *Montage*

Os resultados das simulações para esta instância, variando o conjunto de processadores $P \in \{4, 8, 16, 32\}$, estão consolidados nos gráficos da figura 19.

4.4.1.1 Análise de Desempenho (*Makespan* e SLR)

O comportamento das heurísticas na instância menor revelou um domínio absoluto do HEFT e do IPEFT. Ambos os algoritmos apresentaram resultados quase idênticos em cenários de $P = 4$, $P = 8$ e $P = 16$ e o IPEFT terminou com o melhor *makespan* no cenário mais folgado de $P = 32$. Esse desempenho refletiu em um SLR quase ótimo de ≈ 1.01 e ≈ 1.02, indicando que suas funções de priorização conseguem lidar bem com as *branches* paralelas antes do afunilamento. Em contrapartida, o DLS e o IHEFT mostraram-se inadequados para esse tipo de DAG, apresentando *makespans* substancialmente maiores.

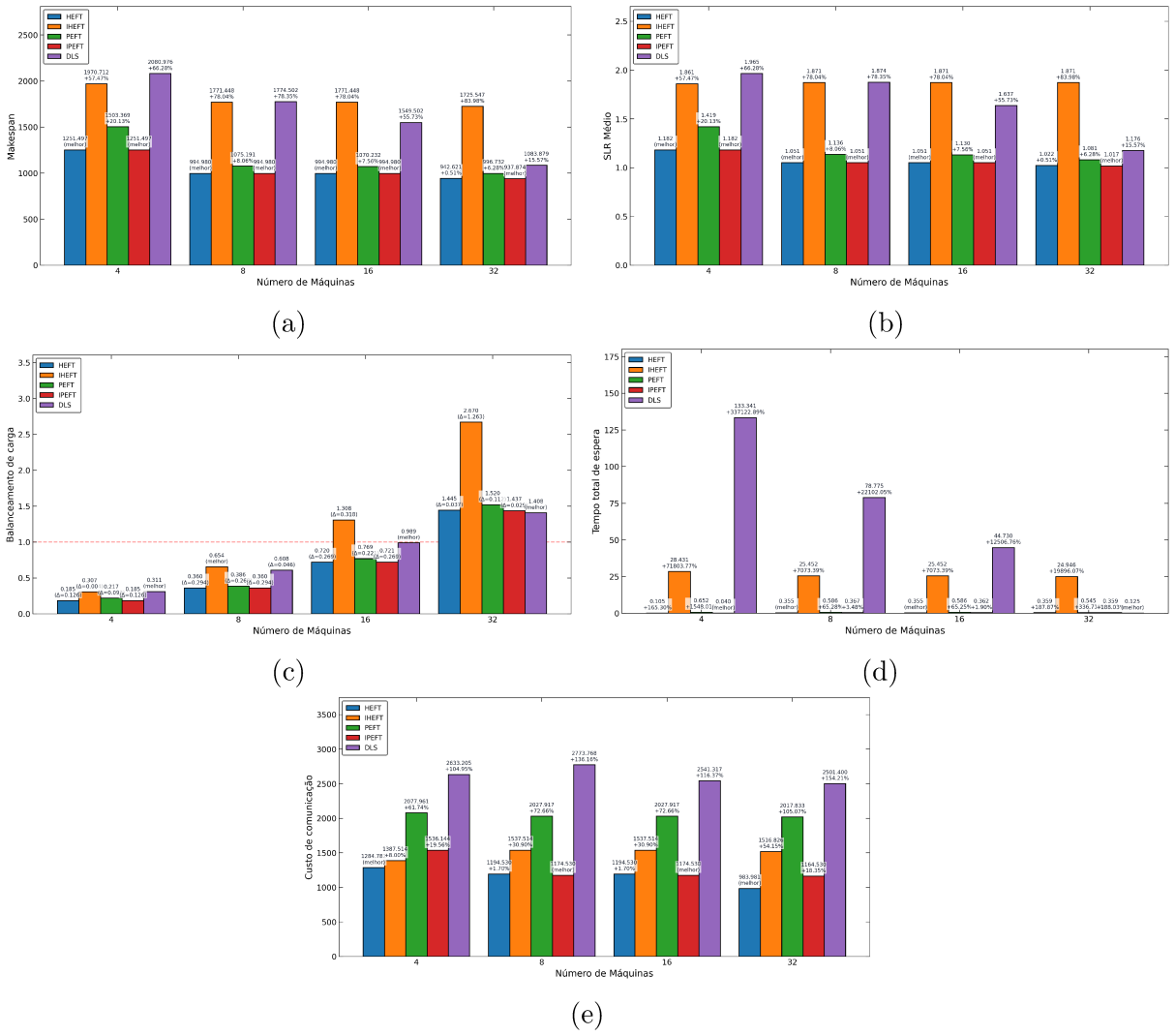


Figura 19 – Resultados - (a) *makespan* / (b) SLR / (c) LB / (d) WT / (e) CC - para o grafo *Montage* (178 tarefas).

4.4.1.2 Utilização de Recursos e Filas (LB e WT)

A topologia do *Montage* pune algoritmos do tipo *non-insertion*, o DLS registrou valores extremamente discrepantes de WT com $P = 4$, $P = 8$ e $P = 16$. Em contrapartida, o IPEFT conseguiu manter o WT em valores ínfimos de ≈ 0.04 no cenário $P = 4$ e o HEFT se sobressaiu nos demais cenários. Isso prova que a capacidade de preencher *gaps* de ociosidade praticamente elimina gargalos de fila na fase paralela do grafo.

4.4.1.3 Penalidade de Comunicação (CC)

O HEFT e o IPEFT se destacam no controle de comunicação de dados nessa escala, atingindo o menor CC em $P = 32$ e se sobressaindo diante dos demais nos outros cenários. No entanto, o PEFT e o DLS apresentaram os maiores valores de CC, indicando uma falha em agrupar as tarefas que compartilham grande dependência de dados antes do grafo se afunilar.

4.4.2 Avaliação em Cenário de Larga Escala (4846 Tarefas)

Ao expandirmos para o cenário de 4846 tarefas, o grau de paralelismo inicial atinge seu limite. A Figura 20 ilustra a topologia do *workflow*.

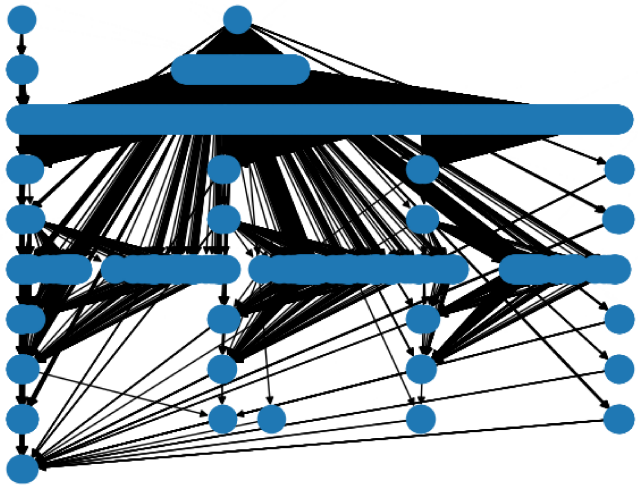


Figura 20 – Ilustração em alto nível da topologia da instância de 4846 tarefas do grafo *Montage*

Os resultados das simulações para esta instância, variando o conjunto de processadores $P \in \{4, 8, 16, 32\}$, estão consolidados na Figura 21.

4.4.2.1 Análise de Desempenho (*Makespan* e SLR)

Diferentemente do comportamento observado no DAG de menor escala, a escala massiva de quase 5 mil tarefas provocou uma maior disputa para essas métricas. Com $P = 32$, o HEFT e o IPEFT apresentaram os melhores valores de *makespan* e SLR. Já o DLS, que foi o pior algoritmo na instância menor, apresentou uma recuperação notável, superando o IHEFT para o cenário de $P = 32$, isso prova que a quantidade de tarefas soltas permite manter os recursos ocupados sem depender primariamente do preenchimento de *gaps*. O IHEFT que também teve os piores valores na instância menor, apresentou valores bem disputados no restante dos cenários, com menos recursos disponíveis, superando o PEFT para $P = 4$ e $P = 8$.

4.4.2.2 Utilização de Recursos e Filas (LB e WT)

A abundância de máquinas destaca o poder do *look-ahead* no PEFT. Apesar de não ter atingido o menor *makespan* geral, o PEFT alcançou os menores valores *waiting time* para todos os cenários, com o menor valor atingido em $P = 32$. O *look-ahead* otimista (OCT) conseguiu ser extremamente eficiente no escalonamento, provocando um melhor aproveitamento dos recursos.

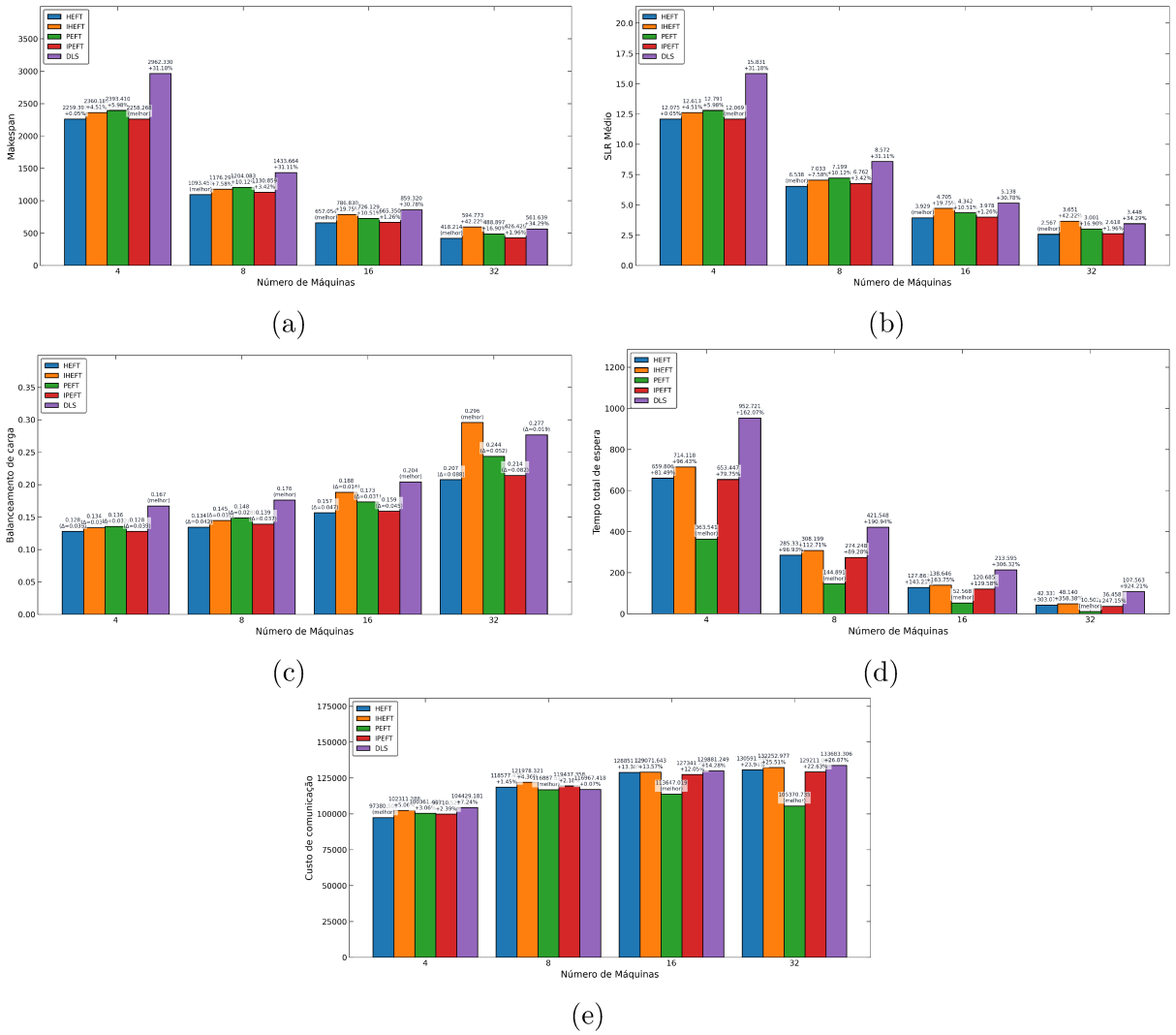


Figura 21 – Resultados - (a) *makespan* / (b) SLR / (c) LB / (d) WT / (e) CC - para o grafo *Montage* (4846 tarefas).

4.4.2.3 Penalidade de Comunicação (CC e LB)

A complexidade do DAG e seu tamanho inverteram o comportamento do CC observado anteriormente. Enquanto o HEFT e o IPEFT obtiveram os melhores valores de CC no grafo menor, nessa instância maior, o cenário com $P = 8$, $P = 16$ e $P = 32$ o PEFT se destacou excepcionalmente, uma economia gigantesca se comparada aos demais algoritmos.

4.5 Análise Comparativa: *Workflows* Reais versus Grafos Sintéticos

O objetivo deste trabalho é fazer uma análise comparativa de alguns algoritmos de escalonamento diante de cenários que se aproximam de ambientes produtivos reais. Para enriquecer esta discussão, esta seção estabelece um paralelo entre os resultados obtidos nesta monografia — que utilizou DAGs provenientes de *workflows* científicos — e o

trabalho de Monteiro (2025), que realizou uma análise análoga utilizando DAGs sintéticos ("*PEFT-like*") escalados de 256 a 4096 tarefas. Ambos os estudos avaliaram heurísticas clássicas (DLS, HEFT, PEFT, IHEFT e IPEFT) sob a ótica de métricas como *makespan*, *waiting time* (WT) e *load balancing* (LB) em ambientes heterogêneos.

4.5.1 Desempenho e Tempos de Espera (*makespan* e WT)

O trabalho de Monteiro (2025) relata que heurísticas baseadas em listas e predição, como HEFT, PEFT e IPEFT, apresentam conjuntamente os melhores *makespans* e WT. Em contrapartida, este trabalho constatou que, embora o HEFT e o IPEFT apresentem *makespan*s sólidos em boa parte dos cenários (com destaque para o IPEFT devido à proteção do caminho crítico), no que diz respeito ao WT, apenas o PEFT apresentou resultados de destaque. O PEFT demonstrou solidez e rigidez no WT na maior parte dos cenários reais analisados, provando que sua estratégia de *look-ahead* é altamente eficiente para mitigar filas.

4.5.2 O Comportamento do IHEFT

No que diz respeito ao IHEFT, Monteiro (2025) constata que a heurística prioriza lacunas de escalonamento e eleva a utilização dos recursos, resultando em menor ociosidade nos processadores. No entanto, este trabalho identificou que o IHEFT obteve resultados de WT ruins em boa parte dos cenários, frequentemente ficando com valores acima dos demais algoritmos — essa característica divergente provavelmente é influenciada pelas topologias dos grafos usados nos trabalhos. Apesar dessa diferença, ambos os trabalhos estão em concordância quanto à métrica de *makespan*; o IHEFT costuma ficar atrás das variantes com *look-ahead* no tempo total de execução.

4.5.3 Balanceamento de Carga no DLS

Um achado empírico que está em concordância com o trabalho de Monteiro (2025) refere-se ao *load balancing* do DLS. Ambos os trabalhos afirmam que o DLS garante um LB muito alto, sem utilizar o *insertion policy*/antevisão do HEFT. Em boa parte dos cenários com limitação de recursos ($P = 4$, $P = 8$ e $P = 16$), o DLS de fato apresenta um *load balancing* substancialmente mais alto que os demais algoritmos. Porém, este trabalho mostrou que em cenários com abundância de recursos ($P = 32$), o comportamento do DLS se altera, com o LB notavelmente se aproximando dos demais algoritmos, ou até ficando menor, mostrando que a sua eficiência de distribuição decai com o aumento de recursos.

4.5.4 Alto Paralelismo e Muitas Tarefas

Por fim, Monteiro (2025) cita que, em cenários com alto paralelismo e com abundância de recursos, o desempenho dos algoritmos HEFT, PEFT e IPEFT converge, ficando praticamente empatado. Este trabalho validou que esse fenômeno também ocorre no mundo real em topologias específicas, como é possível notar no cenário massivo do *workflow Epigenomics* (Figuras 16 e 17a). Por ser um grafo que apresenta várias *pipelines* paralelas contínuas, ao ser submetido a um ambiente com $P = 32$, os algoritmos ficam empatados no que diz respeito ao *makespan*. Isso confirma que a abundância de recursos, somada a um alto grau de paralelismo, satura as otimizações dessas heurísticas de lista.

5 Conclusão

Este trabalho teve como objetivo principal avaliar o desempenho de cinco algoritmos de escalonamento de tarefas (HEFT, IHEFT, PEFT, IPEFT e DLS) em ambientes distribuídos e heterogêneos. O grande diferencial desta monografia foi o foco em cenários reais, substituindo a utilização tradicional de grafos sintéticos por instâncias de *workflows* científicos reais, como o *1000genome*, BWA, *Epigenomics* e *Montage*, que foram extraídos do repositório WfCommons (2024a). Essa metodologia permitiu submeter as heurísticas a topologias reais, expondo gargalos de comunicação e padrões de dependência encontrados em ambientes produtivos.

A análise dos resultados demonstrou que a eficiência de cada algoritmo é diretamente guiada pela topologia do DAG e pela escala da infraestrutura. Em *workflows* com forte característica de *scatter-gather*, como o *1000genome* e o BWA, observou-se uma saturação rápida do paralelismo. Nesses casos, o aumento progressivo de processadores, de 16 para 32 máquinas por exemplo, resultou em ganhos marginais de *makespan*, evidenciando o limite no ganho de performance ao se aumentar o paralelismo, limite representado pela Lei de Amdahl (GUSTAFSON, 2011) em tarefas que dependem de processos sequenciais.

Dentre as heurísticas avaliadas, o IPEFT destacou-se pela sua robustez na minimização do *makespan* e otimização do *scheduling length ratio* (SLR) em grande parte dos cenários. Seu comportamento guiado pela tabela *Pessimistic Cost Table* (PCT) na fase de priorização e pelo mapeamento de nós críticos *critical node parent* (CNP), garantiu um desempenho de ponta em boa parte das instâncias, como nos cenários massivos do *1000genome* e BWA. O HEFT, por sua vez, demonstrou ser uma base sólida e altamente competitiva, empatando ou se sobressaindo com o IPEFT em cenários de altíssima escala como no *Epigenomics* e *Montage*.

Outro achado importante diz respeito ao impacto das estratégias de *look-ahead*. O algoritmo PEFT, através da sua matriz *Optimistic Cost Table* (OCT), provou ser excepcionalmente eficiente no gerenciamento do *waiting time* (WT) principalmente em cenários com grafos de larga escala. Apesar de não atingir sempre o menor *makespan* absoluto, o PEFT evitou que tarefas ficassem bloqueadas desnecessariamente e tomou melhores decisões de localidade de dados, como observado no *workflow* de larga escala do *Montage*.

As políticas de alocação também revelaram falhas estruturais dependendo da topologia. O DLS, restrito por sua política de *non-insertion*, foi penalizado com as maiores taxas de *waiting time* ao não conseguir aproveitar *gaps* entre tarefas, ainda que em cenários isolados tenha alcançado bons custos de comunicação ao agrupar tarefas dependentes.

Da mesma forma, o IHEFT sofreu degradações severas de desempenho no *makespan* em grafos menores, onde sua estratégia de mudança de processador (*crossover*) baseada em um *threshold* gerou congestionamento de execução em vez de evitar as otimizações locais.

No que diz respeito ao *communication cost* (CC), a análise evidenciou que algoritmos que empregam mecanismos que restringem o escalonamento de tarefas entre os processadores, como o limiar de *crossover* do IHEFT, demonstraram eficácia em limitar a transferência de dados e minimizar os custos de rede em cenários específicos de baixa escala, como nos *workflows* BWA e *Epigenomics*.

Em síntese, os resultados comprovam que não existe um único algoritmo ideal para todos os cenários, mas heurísticas que protegem o caminho crítico (como o IPEFT) ou empregam o *look-ahead* (como o PEFT) demonstram maior adaptabilidade. Ao avaliar o problema do escalonamento de tarefas através de aplicações reais, este trabalho entrega contribuições valiosas, confirmando que o desenvolvimento de sistemas distribuídos mais eficientes requer a compreensão não apenas do poder computacional dos recursos, mas da natureza topológica e da característica de comunicação inerentes a cada aplicação.

Referências

A, A. et al. A global reference for human genetic variation. *Nature*, v. 526, p. 68, 10 2015. Citado 2 vezes nas páginas 34 e 35.

ADHIKARI, M.; AMGOTH, T.; SRIRAMA, S. N. A survey on scheduling strategies for workflows in cloud environment and emerging trends. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 52, n. 4, ago. 2019. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3325097>. Citado na página 36.

ALEBRAHIM, S.; AHMAD, I. Task scheduling for heterogeneous computing systems. **J Supercomput**, v. 73, p. 2313–2338, 2017. Citado 2 vezes nas páginas 11 e 16.

_____. Task scheduling for heterogeneous computing systems. **The Journal of Supercomputing**, v. 73, 06 2017. Citado 3 vezes nas páginas 18, 29 e 30.

ALTSCHUL, S. F.; GISH, W.; MILLER, W.; MYERS, E. W.; LIPMAN, D. J. Basic local alignment search tool. **Journal of Molecular Biology**, Elsevier, v. 215, n. 3, p. 403–410, 1990. Citado na página 16.

ARABNEJAD, H.; BARBOSA, J. G. List scheduling algorithm for heterogeneous systems by an optimistic cost table. **IEEE Transactions on Parallel and Distributed Systems**, v. 25, n. 3, p. 682–694, 2014. Citado 4 vezes nas páginas 12, 30, 31 e 32.

AWS. **O que é computação distribuída?** 2025. Disponível em: <https://aws.amazon.com/pt/what-is/distributed-computing/>. Citado 2 vezes nas páginas 11 e 15.

BRAUN, T. D.; SIEGEL, H. J.; BECK, N.; BöLöNI, L. L.; MAHESWARAN, M.; REUTHER, A. I.; ROBERTSON, J. P.; THEYS, M. D.; YAO, B.; HENSGEN, D.; FREUND, R. F. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. **Journal of Parallel and Distributed Computing**, v. 61, n. 6, p. 810–837, 2001. ISSN 0743-7315. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0743731500917143>. Citado 2 vezes nas páginas 11 e 22.

COLEMAN, T.; CASANOVA, H.; POTTIER, L.; KAUSHIK, M.; DEELMAN, E.; SILVA, R. Ferreira da. Wfcommons: A framework for enabling scientific workflow research and development. **Future Generation Computer Systems**, v. 128, p. 16–27, 2022. Citado na página 34.

COSTA, B. C. S. **Avaliação de algoritmos de escalonamento de aplicações paralelas em processadores heterogêneos.** 2022. <https://repositorio.ufu.br/handle/123456789/34620>. Citado 2 vezes nas páginas 22 e 23.

CUNHA, J. V. d. S. **Algoritmo genético para escalonamento de tarefas em ambientes paralelos heterogêneos**. 2024. Disponível em: <https://repositorio.ufu.br/handle/123456789/44002>. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade Federal de Uberlândia, Uberlândia, 2024. Disponível em: <https://repositorio.ufu.br/handle/123456789/44002>. Citado na página 16.

DIJKSTRA, E. W. The structure of the “the”-multiprogramming system. In: **Proceedings of the First ACM Symposium on Operating System Principles**. New York, NY, USA: Association for Computing Machinery, 1967. (SOSP ’67), p. 10.1–10.6. ISBN 9781450373708. Disponível em: <https://doi.org/10.1145/800001.811672>. Citado 2 vezes nas páginas 11 e 15.

FOUNDATION, A. S. **Apache Hadoop**. 2006. Acessado em: 29 mar. 2025. Disponível em: <https://hadoop.apache.org/>. Citado na página 15.

_____. **Apache Spark**. 2009. Acessado em: 29 mar. 2025. Disponível em: <https://spark.apache.org/>. Citado na página 15.

GUSTAFSON, J. L. Amdahl’s law. In: _____. **Encyclopedia of Parallel Computing**. Boston, MA: Springer US, 2011. p. 53–60. ISBN 978-0-387-09766-4. Disponível em: <https://doi.org/10.1007/978-0-387-09766-4_77>. Citado 2 vezes nas páginas 40 e 57.

HAGRAS, T.; JANEčEK, J. Static vs. dynamic list-scheduling performance comparison. **Acta Polytechnica**, v. 43, n. 6, Jan. 2003. Disponível em: <https://ojs.cvut.cz/ojs/index.php/ap/article/view/490>. Citado na página 12.

HESAR, A. Task scheduling using memetic intelligent water drops algorithm based on tabu search: a case study on azure workflows. **Soft Computing**, v. 27, p. 1–17, 04 2023. Citado na página 17.

JUVE, G.; CHERVENAK, A.; DEELMAN, E.; BHARATHI, S.; MEHTA, G.; VAHI, K. Characterizing and profiling scientific workflows. **Future Generation Computer Systems**, v. 29, n. 3, p. 682–692, 2013. ISSN 0167-739X. Special Section: Recent Developments in High Performance Computing and Security. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X12001732>. Citado na página 36.

KUBERNETES. **Kubernetes**. 2014. Acessado em: 29 mar. 2025. Disponível em: <https://kubernetes.io/pt-br/>. Citado na página 15.

L.D., D. B.; Venkata Krishna, P. Honey bee behavior inspired load balancing of tasks in cloud computing environments. **Applied Soft Computing**, v. 13, n. 5, p. 2292–2303, 2013. ISSN 1568-4946. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1568494613000446>. Citado na página 12.

MAO, H.; ALIZADEH, M.; MENACHE, I.; KANDULA, S. Resource management with deep reinforcement learning. In: **Proceedings of the 15th ACM Workshop on Hot Topics in Networks**. New York, NY, USA: Association for Computing Machinery, 2016. (HotNets ’16), p. 50–56. ISBN 9781450346610. Disponível em: <https://doi.org/10.1145/3005745.3005750>. Citado 2 vezes nas páginas 12 e 16.

MOHAMMADJAFARI, A.; KHAJOUIE, P. **Optimizing Task Scheduling in Heterogeneous Computing Environments: A Comparative Analysis of CPU, GPU, and ASIC Platforms Using E2C Simulator**. 2024. Disponível em: <https://arxiv.org/abs/2405.08187>. Citado na página 11.

MONTEIRO, M. C. **Análise comparativa de algoritmos de escalonamento de workflows científicos em processadores heterogêneos**. 140 p. Dissertação (Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação)) —

Universidade Federal de Uberlândia, Uberlândia, Brasil, 2025. Disponível em: <https://repositorio.ufu.br/handle/123456789/47466>. Citado 4 vezes nas páginas 12, 23, 55 e 56.

SIH, G.; LEE, E. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. **IEEE Transactions on Parallel and Distributed Systems**, v. 4, n. 2, p. 175–187, 1993. Citado 2 vezes nas páginas 27 e 28.

TOPCUOGLU, H.; HARIRI, S.; WU, M.-Y. Task scheduling algorithms for heterogeneous processors. In: **Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)**. [S.l.: s.n.], 1999. p. 3–14. Citado 4 vezes nas páginas 11, 19, 28 e 29.

WFCOMMONS. **WfFormat: Collection and curation of open access production workflow executions from various scientific applications. These workflow instances are described using the WfFormat format.** 2024. Disponível em: <https://github.com/WFCommons/WfInstances>. Citado 4 vezes nas páginas 13, 24, 34 e 57.

_____. **WfFormat: Workflow Format Specification and Tools**. 2024. Disponível em: <https://github.com/wfcommons/WfFormat>. Citado 3 vezes nas páginas 24, 25 e 34.

ZHOU, N.; QI, D.; WANG, X.; ZHENG, Z.; LIN, W. A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. **Concurrency and Computation: Practice and Experience**, v. 29, n. 5, p. e3944, 2017. E3944 CPE-15-0408.R2. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3944>. Citado 4 vezes nas páginas 11, 16, 32 e 33.

A Declaração de Uso de Inteligência Artificial Generativa

Para auxiliar no desenvolvimento e aumentar a produtividade durante as etapas de pesquisa e escrita desta monografia, foi utilizado o *Gemini* 3.1 Pro, um modelo de Inteligência Artificial (IA) generativa desenvolvido pelo *Google*. O uso dessa ferramenta foi estritamente instrumental e de apoio para as seguintes frentes:

1. **Revisão gramatical e de formatação:** A IA auxiliou na revisão do texto, sendo instruída a procurar por erros ortográficos, de gramática e inconsistências na padronização (como no uso de negrito, itálico e citações), além de analisar a sintaxe LaTeX.
2. **Assistente (*Copilot*):** A ferramenta atuou como um assistente, ajudando no *brainstorming* da etapa de desenvolvimento e pesquisa e posteriormente na estruturação do texto.
3. **Revisão de consistência do conteúdo:** No fim da escrita, a IA foi empregada para cruzar as informações desta monografia com as referências, a fim de identificar possíveis inconsistências ou erros de conteúdo.
4. **Auxílio na interpretação:** A IA foi utilizada para discutir e auxiliar no entendimento de conceitos complexos, ajudando a preencher lacunas de conhecimento do referencial teórico.

Todo conteúdo gerado pela ferramenta foi revisado criticamente pelo autor, que assume integral responsabilidade pelo trabalho final.