UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Moreira de Camargo

Comparação algoritmos de escalonamento de tarefas para grades computacionais sob diferentes métricas

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Moreira de Camargo

Comparação algoritmos de escalonamento de tarefas para grades computacionais sob diferentes métricas

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

2023

Dedico esse trabalho para meus amigos, familiares e professores.

Agradecimentos

Agradeço a todos meus professores por tudo que me ensinaram, e por toda dedicação a profissão. Agradeço especialmente ao professor Paulo Henrique, meu orientador nesse trabalho, pela paciência e disposição em guiar a direção correta que o trabalho deveria seguir.

Sou grato pelo aprendizado que tive no PET-SI, aos amigos que lá fiz, e aos tutores que sempre estavam dispostos a nos orientar, assim sendo, considero minha passagem pelo PET-SI fundamental na minha formação.

Sou grato as pessoas que foram cruciais para minha formação e meu entusiasmo por aprender, meus pais. Minha mãe Antoniette, que segue eternamente em meu coração, por ser a pessoa mais dedicada, gentil e inteligente que eu já conheci, meu pai Marcos, que foi minha base nos meus momentos de maior fragilidade, que me fez querer persistir e com quem eu posso conversar sobre qualquer coisa.

E por fim, um agradecimento especial ao meu amigo João Gabriel que me acompanhou durante toda faculdade, e com quem continuo a conversar por horas sobre tudo e a ver os melhores e piores filmes quase toda semana, o que me deu forças para persistir.

Resumo

A computação em grade, embora seja vantajosa em diversos cenários, tem como o maior desafio o escalonamento de diferentes tarefas em um sistema heterogêneo de máquinas que sofre mudanças constantes na disponibilidade de recursos computacionais. Portanto, poder comparar as diferentes heurísticas, seus resultados e diferenças nas mais diversas circunstâncias é indispensável para selecionar qual o melhor método para ordenar as tarefas. Diante desse problema frequente nas discussões acadêmicas, esse trabalho pretende analisar algumas das heurísticas clássicas para o escalonamento de tarefas através do aperfeiçoamento e execução de um simulador que as implementam em diferentes cenários, extraindo métricas para as comparações e facilitando futuras implementações de novas heurísticas.

Palavras-chave: Escalonamento de Tarefas. Computação em Grade. Heurísticas. Simulador de escalonamento. Otimização. Utilização de recursos. Cooperação entre máquinas.

Abstract

Grid computing, although advantageous in many scenarios, has as its greatest challenge the scheduling of different tasks in a heterogeneous system of machines that undergo constant changes in the availability of computational resources. Therefore, being able to compare the different heuristics, their results and differences under various circumstances is indispensable to select the best method for ordering the tasks. In view of this frequent problem in academic discussions, this work intends to analyze some of the classical heuristics for scheduling tasks by improving and running a simulator that implements them in different scenarios, extracting metrics for comparison and facilitating future implementations of new heuristics.

Keywords: Task Scheduling. Grid Computation. Heuristics. Escalation simulator. Optimization. Resource utilization. Cooperation between machines.

Sumário

Lista de ilustrações 8

Lista de tabelas 9

Lista de algoritmos 10

1 INTRODUÇÃO 11
1.1 Contextualização e Motivação 11
1.2 Objetivos 12
1.3 Organização do Trabalho 12

2 FUNDAMENTAÇÃO TEÓRICA 13
2.1 Escalonamento de Tarefas em Grades Computacionais 13
2.2 Algoritmos de Escalonamento de Tarefas 15
2.2.1 Minimum Execution Time (MET) 16
2.2.2 Opportunistic Loading Balancing (OLB) 17
2.2.3 Minimum Completion Time (MCT) 18
2.2.4 Min-Min 19
2.2.5 Max-Min 21
2.2.6 Sufferage 22
2.2.7 Min-Mean 24
2.2.8 Min-Var 26
2.2.9 Min-Max 29

3 DESENVOLVIMENTO 31
3.1 Modificações no Simulador 31
3.1.1 Organização das Classes do Simulador 32

4 EXPERIMENTOS 34
4.1 Cenários avaliados 34
4.2 Parâmetros para comparação das heurísticas 35
4.2.1 Cenário 1 - High-High 37
4.2.2 Cenário 1 - High-Low 39
4.2.3 Cenário 1 - Low-High 41
4.2.4 Cenário 1 - Low-Low 42

5 CONCLUSÕES 45

REFERÊNCIAS 46

APÊNDICE A – TABELAS 48

Lista de ilustrações

Figura 1 – Diagrama de Classes do Simulador 32
Figura 2 – Makespan, flowtime e utilização Cenário - High-High 37
Figura 3 – Matching Proximity - Cenário High-High 38
Figura 4 – MPR - Cenário High-High 38
Figura 5 – Makespan, flowtime e utilização Cenário - High-High 39
Figura 6 – Matching Proximity - Cenário High-Low 40
Figura 7 – MPR - Cenário High-Low 40
Figura 8 – Makespan, flowtime e utilização Cenário - Low-High 41
Figura 9 – Matching Proximity - Cenário Low-High 41
Figura 10 – MPR - Cenário Low-High 42
Figura 11 – Makespan, flowtime e utilização Cenário - Low-Low 43
Figura 12 – Matching Proximity - Cenário Low-Low 43
Figura 13 – MPR - Cenário Low-Low 44

Lista de tabelas

Tabela 1 – Dados iniciais dos exemplos dos algoritmos MET, OLB, MCT . . 16
Tabela 2 – Distribuição das tarefas do algoritmo MET 17
Tabela 3 – Distribuição das tarefas do algoritmo OLB 18
Tabela 4 – Distribuição das tarefas do algoritmo MCT 19
Tabela 5 – Dados iniciais dos exemplos dos algoritmos Max-Min, Min-Min Min-Var, Min-Max 20
Tabela 6 – Distribuição das tarefas do algoritmo Min-Min 21
Tabela 7 – Distribuição das tarefas Max-Min 22
Tabela 8 – Dados iniciais dos exemplos do algoritmo Sufferage 24
Tabela 9 – Distribuição das tarefas Sufferage 24
Tabela 10 – Distribuição das tarefas Min-Mean 26
Tabela 11 – Distribuição das tarefas Min-Var 29
Tabela 12 – Distribuição das tarefas Min-Max 30
Tabela 13 – Resultados cenário High-High 49
Tabela 14 – Resultados cenário High-Low 50
Tabela 15 – Resultados cenário Low-High 51
Tabela 16 – Resultados cenário Low-Low 52

Lista de algoritmos

Algoritmo 1 – Pseudocódigo do algoritmo MET 16
Algoritmo 2 – Pseudocódigo do algoritmo OLB 17
Algoritmo 3 – Pseudocódigo do algoritmo MCT 18
Algoritmo 4 – Pseudocódigo do algoritmo *Min-Min* 20
Algoritmo 5 – Pseudocódigo do algoritmo *Max-Min* 21
Algoritmo 6 – Pseudocódigo do algoritmo *Sufferage* 23
Algoritmo 7 – Pseudocódigo do algoritmo *Min-Mean* 25
Algoritmo 8 – Pseudocódigo do algoritmo *Min-Var* 27
Algoritmo 9 – Pseudocódigo do algoritmo *Min-Max* 30

1 Introdução

1.1 Contextualização e Motivação

Computação em grade pode ser compreendida como um modo de estruturar a cooperação entre diferentes máquinas a fim de realizar diferentes tarefas. Em um cenário onde máquinas, com diferentes recursos computacionais (poder de processamento, disponibilidade de armazenamento, memória RAM, entre outros), podem estar em diferentes locais e receber diferentes tipos de tarefas (computação distribuída), se faz necessário pensar em algoritmos para escalonar essas tarefas em paralelo a fim de otimizar o tempo e os recursos disponíveis. Portanto, para a computação em grade, distribuir o poder computacional e otimizar sua utilização é mais relevante do que adquirir mais recursos (SINGH; BAWA, 2011).

Por se tratar de um problema NP-Completo, não se conhece um algoritmo para resolvê-lo em tempo polinomial, embora seja fácil conferir a exatidão das soluções encontradas. Portanto, a fim de obter soluções ótimas ou sub-ótimas em um tempo reduzido, é imprescindível empregar algoritmos de escalonamento que empreguem heurísticas (ZHANG; INOGUCHI; SHEN, 2004), isto é, algoritmos que estimam o custo ou a qualidade de uma solução sem a certeza que é a melhor solução possível. Heurísticas são usadas para tomar decisões rápidas e aproximadas sobre como escalonar as tarefas. Embora as heurísticas possam não garantir a solução ótima para um problema de escalonamento de tarefas, elas encontram soluções satisfatórias em um tempo aceitável. Isso faz com que elas sejam uma boa alternativa para muitas situações. Todavia, a fim de se adaptar a heterogeneidade dos recursos disponíveis e das tarefas as serem realizas em pouco tempo e, se conhecer as aplicações e máquinas disponíveis, é possível optar por heurísticas que consumam menos recurso para escalonar as tarefas e achem boas soluções para cenários com características em comum (XHAFA; ABRAHAM, 2010). Assim sendo, é vantajoso conhecer quais problemas ou tarefas o sistema será encarregado, quais recursos estão disponibilizados pelas máquinas e quão heterogêneo as máquinas e as tarefas são, ou poderão vir a ser.

Neste trabalho, a partir das alterações realizadas por Barra (2022) no simulador de Verma (2010), foram realizadas adaptações que facilitem a implementação e comparação dos resultados resultados, além de permitir que dados para geração de novas instâncias e seus parâmetros sejam mais acessíveis para edição sem demandar

compreensão do código do simulador e suas diversas classes.

1.2 Objetivos

O objetivo principal desse trabalho é analisar e propor alterações do simulador implementado por Verma (2010) e modificado por Barra (2022), sendo elas:

- Parametrizar o simulador para que seja mais prático ao usuário inserir dados sobre as máquinas disponíveis, sobre as tarefas a serem realizadas, e quais heurísticas devem ser executadas;
- Permitir que o usuário possa escolher entre gerar novos dados para realizar a simulação ou utilizar dados existentes em um arquivo;
- Salvar os resultados gerados em um arquivo separado, permitindo resultados de outras pesquisas possam ser comparadas com mais facilidade;
- Revisar algumas implementações de heurísticas para que sejam mais adequadas à teoria e ao seu pseudocódigo.
- Implementar um novo algoritmo para o simulador e utilizar métricas para comparar os resultados em cenários diversos.

1.3 Organização do Trabalho

A organização deste trabalho consiste em quatro capítulos: Fundamentação teórica (Capítulo 2), onde será abordado um pouco mais dos objetivos e desafios da computação em grade, além de alguns conceitos utilizados ao longo de todo o trabalho. O desenvolvimento (Capítulo 3, onde são descritas as modificações que foram feitas no simulador e quais algoritmos foram analisados, com uma explicação breve e um exemplo de um cenário simples utilizando cada um dos algoritmos. Os experimentos (Capítulo 4), nos quais são explicados quais parâmetros foram utilizados para a construção de cada cenário e como os algoritmos se comportaram em cada um dos cenários descritos. Por fim, uma conclusão (Capítulo 5) abrangendo os resultados gerais de algumas heurísticas, como elas podem ser utilizadas, e como o simulador pode ser utilizado em futuras análises e comparações de algoritmos aplicados a computação em grade.

2 Fundamentação Teórica

2.1 Escalonamento de Tarefas em Grades Computacionais

A computação em grade (do inglês, *grid computing*) visa aproveitar recursos computacionais distribuídos geograficamente para resolver problemas complexos e executar tarefas de alto desempenho. Essa abordagem é baseada na ideia de criar uma infraestrutura virtualizada (DONG; AKL, 2006; XHAFA; ABRAHAM, 2008), na qual recursos computacionais, como processadores, armazenamento e redes, são compartilhados entre várias organizações e usuários. O conceito de grade computacional surgiu como uma evolução da computação distribuída e da colaboração em larga escala (DONG; AKL, 2006). Esses ambientes, geralmente, são compostos por recursos heterogêneos e distribuídos em diferentes locais geográficos. Essa distribuição geográfica cria a oportunidade de acessar recursos locais e remotos, permitindo a execução de tarefas de forma mais rápida e eficiente, além de possibilitar a solução de problemas que requerem uma grande quantidade de recursos computacionais.

Esse ambiente oferece várias vantagens, permitindo o compartilhamento de recursos a fim de reduzir custos e aumentar a eficiência na execução de aplicações. Além disso, viabiliza a execução de aplicações de alto desempenho que exigem poder computacional significativo, como simulações complexas, modelagem molecular, análises de dados em larga escala e renderização de imagens. No entanto, a computação em grade também apresenta desafios significativos, como o gerenciamento e o escalonamento eficientes dos recursos distribuídos.

Nesse contexto, o problema de escalonamento de tarefas em grades computacionais consiste em determinar a melhor alocação de tarefas nos recursos disponíveis, levando em consideração diversos fatores, como a capacidade computacional dos recursos, a disponibilidade de rede, as restrições de tempo e as dependências entre as tarefas (DONG; AKL, 2006). Além disso, é necessário considerar a carga de trabalho dos recursos e buscar um balanceamento adequado para evitar sobrecarga ou ociosidade excessiva.

Diversos algoritmos de escalonamento têm sido propostos para lidar com esse problema complexo (DONG; AKL, 2006; XHAFA; ABRAHAM, 2008). Esses algoritmos buscam otimizar diferentes critérios, como o tempo de execução total, a utilização eficiente dos recursos, a minimização da latência da rede e a maximização

da taxa de conclusão das tarefas (KAMALAM; BHASKARAN, 2010). A complexidade do sistema em que será aplicado uma técnica de escalonamento de tarefas, pode se tornar um desafio na hora de aplicar algum dos algoritmos vistos nesse trabalho, por isso algumas propriedades do sistema precisam serem consideradas para além da simples comparação entre os algoritmos a serem analisados neste trabalho, como listado por Dong e Akl (2006):

Heterogeneidade e autonomia: Por se tratar de sistemas que possuem acesso a diferentes máquinas e aplicações através da internet, as redes subjacentes que conectam os nós e distribuem as tarefas também são heterogêneas e interferem na capacidade dos recursos de processar uma tarefa ou de acessar dados relacionados aos recursos da rede. Além disso, diferentemente de um sistema distribuído tradicional, em um sistema de grade os recursos possuem maior autonomia, ou seja, o escalonador de tarefas não possui controle total dos recursos. Portanto, um sistema de grade precisa se adaptar as diretrizes e protocolos locais, seja da rede ou do próprio recurso.

Flexibilidade dos recursos disponíveis: É essencial que o escalonador possa avaliar e estimar as capacidades das máquinas disponíveis considerando as constantes mudanças nas performances desses recursos. Justamente pela autonomia dos recursos nesse tipo de sistema, é necessário considerar que as capacidades de cada recurso estão em constantes mudanças devido a competição entre tarefas que não necessariamente são as mesmas atribuídas pelo sistema de computação em grade, afinal os recursos disponíveis não são exclusivamente dedicados a esse sistema. Outra variável que pode influenciar na disponibilidade dos recursos é o acesso à própria rede que os conecta ao sistema. Esses fatores devem ser considerados no momento de escalonar ou redistribuir tarefas escalonadas anteriormente.

Seleção de recursos e Separação entre computação e dados: Em sistemas tradicionais, o custo de preparação de dados é desprezível ou determinado antes da execução, mas em uma grade isso pode ser significativo. Em um sistema em grade, os recursos além de serem heterogêneos em termos computacionais, os recursos de armazenamento de dados também são, e sofrem dos mesmos problemas relacionados a flexibilidade dos recursos computacionais. Compartilhar os dados entre os recursos nessa rede é um dos fatores que pode impactar no escalonamento de tarefas, já que a preparação dos dados e envio para os diferentes recursos também demanda esforço computacional.

Existem outros fatores que também devem ser considerados em um sistema desse modelo, como a confiabilidade dos recursos disponíveis (DABROWSKI, 2009), como o sistema responderia caso ocorra uma falha ou interrupção no processamento de alguma tarefa, e a escalabilidade do sistema, quais limitações está sujeito caso a demanda ou a quantidade de recursos aumente.

2.2 Algoritmos de Escalonamento de Tarefas

Este trabalho considera o problema de escalonamento heterogêneos, ou seja, o tempo de execução de uma tarefa varia de um processador para outro. Além disso, foca-se em tarefas independentes; portanto, não há eventos de comunicação nem restrições de precedência. Embora seja uma versão simplificada do problema de escalonamento mais geral, que leva em consideração as dependências de tarefas, o escalonamento de tarefa independente é considerado importante dentro de ambientes distribuídos, como centros de supercomputação e as grades computacionais (NESMACHNOW; CANCELA; ALBA, 2010; SOUSA, 2022).

Nesta seção, alguns conceitos e siglas são utilizados diversas vezes nos pseudocódigos e explicações dos algoritmos utilizados neste trabalho. Tais conceitos são listados a seguir:

- **makespan**: Tempo necessário para finalizar todas as tarefas atribuídas.
- **flowtime**: A soma de todos os tempos para finalizar cada uma das tarefas pelas máquinas que as foram designadas.
- **mat[máquina]:** Tempo para a máquina concluir as tarefas que lhe foram atribuídas.
- **etc[tarefa][máquina]:** Tempo para o processamento de determinada tarefa pela máquina

Nas próximas seções, são descritos os algoritmos que foram estudados neste trabalho. Tais algoritmos, em sua maioria, seguem heurísticas gulosas e regras de refinamento de soluções. É importante destacar que o problema de escalonamento de tarefas tem sido tratado, frequentemente, por meio de meta-heurísticas (SOUSA, 2022), como algoritmos genéticos e enxames de partículas, mas esses métodos não são contemplados nesta monografia.

2.2.1 Minimum Execution Time (MET)

O *Minimum Execution Time* (MET) pode ser considerado o algoritmo mais simples, porque utiliza apenas os dados sobre qual máquina irá realizar a tarefa mais rapidamente. Não utiliza, assim, dados sobre o tempo que a máquina irá completar as outras tarefas já designadas a ela (XHAFA; ABRAHAM, 2008). Essa heurística também é conhecida como *Limited Best Assignment* (LBA) ou *User Directed Assignment* (UDA). O Algoritmo 1 mostra o pseudocódigo do MET. Aqui, é utilizado o etc[tarefa][máquina], que é o tempo que a máquina irá realizar a tarefa, não levando em conta o processamento das tarefas atribuídas anteriormente a máquina.

Algoritmo 1: Pseudocódigo do algoritmo MET

```
   Dados: Lista de Tarefas
 1 para tarefa em tarefas faça
 2 |   tempoMínimo = valor máximo positivo;
 3 |   para maquina em máquinas faça
 4 |   |   se (etc[tarefa][máquina] < tempoMínimo) então
 5 |   |   |   tempoMínimo = etc[tarefa][máquina];
 6 |   |   |   maquinaEscolhida = maquina;
 7 |   |   fim
 8 |   fim
 9 |   atribuir tarefa para máquinaEscolhida;
10 fim
```

A fim de ilustrar o funcionamento desse algoritmo, considere a Tabela 1, que traz o tempo de execução de três tarefas em três máquinas distintas.

Tabela 1 – Dados iniciais dos exemplos dos algoritmos MET, OLB, MCT

Tarefa	Máquina 1	Máquina 2	Máquina 3
1	125	130	115
2	40	30	20
3	50	13	10

Fonte: Elaborado pelo Autor (2023)

Inicialmente, o algoritmo seleciona a primeira tarefa e verifica o tempo que cada máquina demandaria para executá-la. Em seguida, escolhe a máquina que demandaria menos tempo, no caso, a máquina 3 que demandaria 115 unidades de tempo, no caso segundos. Prossegue-se para a segunda tarefa e realiza-se o mesmo procedimento: verifica-se o tempo que cada máquina demandaria para executá-la. Escolhe-se novamente a máquina que demandaria menos tempo, que também é a máquina 3, que demandaria 20 segundos. Não se considera o tempo que a máquina

já despendeu na primeira tarefa, pois o algoritmo somente considera o tempo de cada tarefa isoladamente. Realiza-se o mesmo procedimento para a terceira tarefa, e novamente é escolhido a máquina 3 para realizar a tarefa, com tempo de 10 segundos.

Ao final, obtém-se a distribuição mostrada na Tabela 2, onde o mat da máquina 1 é 0, o mat da máquina 2 é 0, e o mat da máquina 3 é a soma dos tempos para executar a tarefa 1, 2 e 3, ou seja, $115 + 20 + 10 = 145$. Por ser o maior mat entre as três máquinas, o makespan (tempo para finalizar a última das tarefas) será de 145, assim como o flowtime (tempo somado para finalizar todas as tarefas).

Tabela 2 – Distribuição das tarefas do algoritmo MET

Máquina	Tarefas	Tempo Total
1	Nenhuma	0
2	Nenhuma	0
3	1, 2 e 3	145

Fonte: Elaborado pelo Autor (2023)

2.2.2 Opportunistic Loading Balancing (OLB)

O *Opportunistic Loading Balancing* (OLB) leva em conta apenas o tempo para as máquinas estarem disponíveis, após processarem todas as tarefas designadas a ela. Portanto, não considera o tempo de processamento da tarefa atual para cada máquina. O objetivo dessa heurística é manter as máquinas sempre ocupadas, independente de qual seja a máquina que obteria o menor tempo para executar determinada tarefa, o que afeta negativamente os resultados como o makespan e o flowtime. O Algortimo 2 ilustra esse método em um pseudocódigo.

Algoritmo 2: Pseudocódigo do algoritmo OLB

```
1  para tarefa em tarefas faça
2      minMatTime = valor máximo positivo;
3      para maquina em máquinas faça
4          se (mat[máquina] < minMatTime) então
5              minMatTime = mat[máquina];
6              máquinaEscolhida = máquina;
7          fim
8      fim
9      atribuir tarefa para máquinaEscolhida;
10 fim
```

Novamente a partir dos dados da Tabela 1, verifica-se o algoritmo OLB comporta-se em cada uma das iterações. Na primeira iteração, a máquina 1 não

possui nenhuma tarefa atribuída a ela, portanto o tempo de conclusão das tarefas (mat) dessa máquina é 0, e mesmo que a máquina 3 possa executar a tarefa mais rapidamente, seu mat também é 0, portanto a tarefa será atribuída para a primeira máquina analisada, no caso, a máquina 1. Todavia, na segunda iteração, a máquina 1 já possui mat igual a 125, e a próxima máquina por ter um mat de 0 será escolhida para processar a tarefa. E na iteração que analisa a terceira tarefa, a máquina 3 tem o menor mat (0), seguida pela máquina 2 (30) e pela máquina 1 (125). O algoritmo escolhe a máquina 3 para executar a tarefa 3 e atualiza o mat dela para 10.

A distribuição final das tarefas é mostrada na Tabela 3. Portanto o mat da máquina 1 é 125, o mat da máquina 2 é 30 e o mat da máquina 3 é igual 10. Por ser o maior mat entre as três máquinas, o makespan fica igual a 125 segundos; já o tempo das execuções somadas, o flowtime, é de 165 segundos.

Tabela 3 – Distribuição das tarefas do algoritmo OLB

Máquina	Tarefas	Tempo Total
1	1	125
2	2	30
3	3	10

Fonte: Elaborado pelo Autor (2023)

2.2.3 Minimum Completion Time (MCT)

A heurística *Minimum Completion Time* (MCT) leva em conta dados do processamento da tarefa atual (**etc**) e o tempo para finalizar o processamento das tarefas que foram incumbidas anteriormente para determinada máquina (**mat[máquina]**). O Algoritmo 3 mostra o pseudocódigo do MCT.

Algoritmo 3: Pseudocódigo do algoritmo MCT

```
1  para tarefa em tarefas faça
2      tempoMínimo = valor máximo positivo;
3      para maquina em máquinas faça
4          se (etc[tarefa][máquina] + mat[máquina] < tempoMínimo)
            então
5              tempoMínimo = etc[tarefa][máquina] + mat[máquina];
6              máquinaEscolhida = máquina;
7          fim
8      fim
9      atribuir tarefa para máquinaEscolhida;
10 fim
```

Pode-se analisar a partir dos dados iniciais da Tabela 1, como o algoritmo MCT comporta-se em cada uma das iterações. Para a primeira tarefa, a escolha da máquina é igual ao MET, porque nenhuma das máquinas possui alguma tarefa atribuída a ela, ou seja, o mat de todas as máquinas é 0. Portanto o é comparado apenas o etc das máquinas para a primeira tarefa, por isso a máquina 3 é escolhida, com o menor etc entre as 3 máquinas, 115 segundos. Entretanto, a partir da segunda iteração, embora a máquina 3 tenha o menor etc, 20 segundos, quando somado ao tempo gasto para executar a tarefa anterior o tempo fica $>$ o que a máquina 2 gastaria para executar a tarefa, por isso a tarefa 2 é atribuída para a máquina 2, aumentando seu mat para 30 segundos. Por fim, para a última tarefa, a máquina 1 demoraria 50 segundos, a máquina 2 demoraria 43 segundos $(13 + 30)$, e a máquina 3 demoraria 125 segundos $(115 + 10)$, portanto a máquina 2 novamente possui um etc somado ao mat $<$ das outras máquinas, assim a tarefa 3 é atribuída a ela.

Como resultado (Tabela 4), para esse cenário com a heurística MCT, o makespan é de 115 segundos, por ser o maior entre os tempos, e o flowtime de 158 segundos $(115 + 43)$.

Tabela 4 – Distribuição das tarefas do algoritmo MCT

Máquina	Tarefas	Tempo Total
1	Nenhuma	0
2	2	43
3	3	115

Fonte: Elaborado pelo Autor (2023)

2.2.4 Min-Min

O Min-Min é uma heurística parecida com MCT, mas que calcula o tempo mínimo ao comparar o tempo entre todas as máquinas para executar cada uma das tarefas somado ao tempo para executar todas as tarefas atribuídas anteriormente. Dessa forma, a atribuição das tarefas não é feita uma tarefa por vez, mas sim entre todas as tarefas qual terá o menor tempo para ser processada, conforme descrito no Algoritmo 4.

Diferentemente das heurísticas anteriores, o Min-Min não analisa os tempos de conclusão separadamente por tarefa. Para escolher qual tarefa será primeiro atribuída, é necessário comparar o tempo mínimo (qual máquina pode executar a tarefa mais rapidamente) de todas as tarefas. Portanto, a partir dos dados iniciais da Tabela 5, a primeira iteração encontra que a tarefa com menor tempo mínimo para ser

Algoritmo 4: Pseudocódigo do algoritmo *Min-Min*

```
1  repita
2      máquinaEscolhida = 0
3      tarefaEscolhida = 0
4      tempoMinimo = valor máximo positivo
5      para tarefa em tarefas não atribuídas faça
6          para maquina em máquinas faça
7              se (etc[tarefa][máquina] + mat[máquina] < tempoMínimo)
                então
8                  tempoMínimo = etc[tarefa][máquina] + mat[máquina];
9                  máquinaEscolhida = máquina;
10                 tarefaEscolhida = tarefa;
11             fim
12         fim
13     fim
14     atribuir tarefa para máquinaEscolhida;
15 até que todas as tarefas tenham sido atribuídas;
```

Tabela 5 – Dados iniciais dos exemplos dos algoritmos Max-Min, Min-Min Min-Var, Min-Max

Tarefa	Máquina 1	Máquina 2	Máquina 3
1	125	50	115
2	40	30	20
3	50	13	10
4	200	100	50

Fonte: Elaborado pelo Autor (2023)

processada é a 3, atribuindo assim ela a máquina 3 que processaria ela nesse tempo. Na segunda iteração, a tarefa 2 foi identificada como a mais rápida a ser processada. Ao somar o mat da máquina com o etc para executar a tarefa, constatou-se que as máquinas 2 e 3 poderiam executar a tarefa utilizando o mesmo tempo para concluir. Como a máquina 2 foi analisada primeiro, ela foi escolhida para processar a tarefa. Na terceira iteração, é escolhida a máquina 3 para processar a tarefa 4, porque o tempo para a máquina 3 executar a tarefa 3 e a 4 seria de 60 segundos, que é menor que o tempo para qualquer máquina executar a tarefa 1. Por último, a tarefa 1 é atribuída a máquina 2, com tempo total de 80 segundos para executar as tarefas atribuídas a ela.

Analisando o resultado gerado pelo Min-Min, conforme mostrado na Tabela 6, verificamos que o makespan é de 80 segundos, e o flowtime de 140 segundos $(80 + 60)$.

Tabela 6 – Distribuição das tarefas do algoritmo Min-Min

Máquina	Tarefas	Tempo Total
1	Nenhuma	0
2	2 e 1	80
3	3 e 4	60

Fonte: Elaborado pelo Autor (2023)

2.2.5 Max-Min

Assim como o Min-Min, o Max-Min também calcula qual máquina irá executar a tarefa em menos tempo, mas ao invés de atribuir a tarefa que seria processada em menor tempo para máquina mais rápida, é atribuído a tarefa que irá levar mais tempo para a máquina que a processaria mais rapidamente. O pseudocódigo desse algortimo é mostrado no Algoritmo 5.

Algoritmo 5: Pseudocódigo do algoritmo *Max-Min*

```
1  repita
2      tempoMinimo = valor máximo positivo;
3      maxMinComplTime = valor mínimo negativo;
4      para tarefa em tarefas faça
5          se (foiRemovida[tarefa]) então
6              continue para a próxima iteração;
7          fim
8          para maquina em máquinas faça
9              se (etc[tarefa][máquina] + mat[máquina] < tempoMínimo)
                então
10                 tempoMínimo = etc[tarefa][máquina] + mat[máquina];
11                 máquinaEscolhida = máquina;
12             fim
13         fim
14         minComplTime[tarefa] = tempoMinimo;;
15         minComplMachine[tarefa] = máquinaEscolhida;
16         se (maxMinComplTime < minComplTime[tarefa]) então
17             maxMinComplTime = minComplTime[tarefa];
18             tarefaEscolhida = tarefa;
19         fim
20     fim
21     atribuir tarefaEscolhida para minComplMachine[tarefaEscolhida];
22     foiRemovida[tarefaEscolhida] = Verdadeiro;
23 até que todas as tarefas tenham sido atribuídas;
```

Para esse exemplo utilizaremos os dados da Tabela Tabela 5. Inicialmente

é realizado uma comparação entre as máquinas para aferir qual delas realiza mais rapidamente a tarefa, e depois é comparado com as outras tarefas qual demoraria mais tempo. Entre todas as tarefas, tanto a primeira quanto a quarta tarefa possuem tempos mínimos de 50 segundos, como a tarefa 1 foi analisada primeiro, ela será a primeira a ser atribuída, neste caso para a máquina 2. Na segunda iteração, por não ter sido atribuída a mesma máquina, a tarefa 3 pode ser atribuída para a máquina 3, que possui o maior tempo mínimo entre todas as máquinas, 50 segundos. Na iteração seguinte, a máquina 1 por ser a única que não teve nenhuma tarefa ainda atribuída, tem vantagem na hora de escolha já que seu mat é 0. A terceira tarefa, por ser a com maior tempo mínimo, é atribuída para a máquina 1. A última iteração é mais simples, pois só temos a tarefa 2 para alocar. Então, basta comparar o tempo mínimo de cada máquina e escolher a que tem o menor valor. Assim, a tarefa 2 vai para a máquina que tiver menor tempo para concluir as tarefas anteriores somada a tarefa atual. Como todas as máquinas até o momento possuem tempo de conclusão das tarefas anteriores igual a 50 segundos, o tempo mínimo para concluir a tarefa 2 é 20 segundos pela máquina 3.

Analisando o resultado gerado pelo Max-Min (Tabela 7), verificamos que o makespan é de 70 segundos, é o flowtime de 170 segundos $(50 + 50 + 50 + 20)$.

Tabela 7 – Distribuição das tarefas Max-Min

Máquina	Tarefas	Tempo Total
1	3	50
2	1	50
3	2 e 4	70

Fonte: Elaborado pelo Autor (2023)

2.2.6 Sufferage

A heurística Sufferage (Algoritmo 6) tem como objetivo atribuir primeiro as tarefas que teriam maior custo se fossem executadas por outra máquina que não a mais rápida para processar a tarefa naquele momento, ou seja, a tarefa seria mais penalizada (*suffer*) se não for atribuída a máquina com melhor MCT.

Para esse exemplo utilizaremos novamente os dados Tabela 8. O critério para analisar esse algoritmo é verificar quais tarefas têm a maior diferença entre o tempo mínimo e o tempo mínimo subsequente para processá-las. Por exemplo, a tarefa 1 pode ser feita em 115 segundos na máquina mais rápida, ou em 125 segundos na segunda mais rápida. A diferença entre esses tempos é pequena, só 10 segundos.

Algoritmo 6: Pseudocódigo do algoritmo *Sufferage*

```
1  repita
2      tempoMinimo1 = valor máximo positivo;
3      máquina1 = -1;
4      tempoMinimo2 = valor máximo positivo;
5      máquina2 = -1;
6      tarefaEscolhida = -1;
7      máquinaEscolhida = -1;
8      sufferageMaximo = valor mínimo negativo;
9      para tarefa em tarefas faça
10         se (foiRemovida[tarefa]) então
11             continue para a próxima iteração;
12         fim
13         para maquina em máquinas faça
14             se (etc[tarefa][máquina] + mat[máquina] < tempoMínimo1)
                então
15                 tempoMínimo1 = etc[tarefa][máquina] + mat[máquina];
16                 máquina1 = máquina;
17             fim
18         fim
19         para maquina em máquinas faça
20             se (máquina != máquina1 E etc[tarefa][máquina] +
                mat[máquina] < tempoMínimo2) então
21                 tempoMínimo2 = etc[tarefa][máquina] + mat[máquina];
22                 máquina2 = máquina;
23             fim
24         fim
25         sufferage = tempoMinimo2 - tempoMinimo1;
26         se ( sufferage > sufferageMaximo) então
27             sufferageMaximo = sufferage;
28             tarefaEscolhida = tarefa;
29             máquinaEscolhida = máquina1;
30         fim
31         tempoMinimo1 = valor máximo positivo;
32         tempoMinimo2 = valor máximo positivo;
33     fim
34     atribuir tarefaEscolhida para máquinaEscolhida;
35     foiRemovida[tarefaEscolhida] = Verdadeiro;
36 até que todas as tarefas tenham sido atribuídas;
```

Tabela 8 – Dados iniciais dos exemplos do algoritmo Sufferage

Tarefa	Máquina 1	Máquina 2	Máquina 3
1	125	130	115
2	40	30	20
3	50	13	10
4	200	100	50

Fonte: Elaborado pelo Autor (2023)

Mas a tarefa 4 tem uma diferença bem maior: na máquina mais rápida, ela leva 50 segundos, e na segunda mais rápida, leva 100 segundos. Então, a primeira coisa que o algoritmo faz é escolher a tarefa 4 e colocá-la na máquina mais rápida, que é a máquina 3. Na segunda iteração, a maior diferença é para a tarefa 3, que leva 13 segundos na máquina 2 (a mais rápida) e 50 segundos na máquina 1 (a segunda mais rápida). Na terceira iteração, a maior diferença é para a tarefa 1, que leva 125 segundos na máquina 1 (a mais rápida) e 143 segundos na máquina 2 (a segunda mais rápida). Por fim, a maior diferença para a tarefa 2, é de 27 (43 segundos na máquina 2, e 70 segundos na máquina 3).

A partir do resultado gerado pelo Sufferage, contata-se que o makespan é de 125 segundos, e o flowtime de 218 segundos ($125 + 43 + 50$), conforme mostrado na Tabela 9.

Tabela 9 – Distribuição das tarefas Sufferage

Máquina	Tarefas	Tempo Total
1	1	125
2	2 e 3	43
3	4	50

Fonte: Elaborado pelo Autor (2023)

2.2.7 Min-Mean

O Min-Mean é uma heurística que visa otimizar os resultados encontrados pelo Min-Min, calculando a média gasta por cada máquina para concluir as tarefas atribuídas a elas pelo Min-Min. Depois para cada máquina, verifica se o tempo de conclusão das terefas dela é maior que a média. Caso seja maior, verifica se atribuir uma das tarefas dela para outra máquina, a média pode diminuir. Portanto, o objetivo dessa heurística é redistribuir melhor as tarefas, para isso otimiza a utilização das máquinas, diminuindo o tempo de ociosidade médio delas. O Algoritmo 7 traz o pseudocódigo dessa heurística.

Algoritmo 7: Pseudocódigo do algoritmo *Min-Mean*

```
1  Execute o Min-Min;
2  tempoConclusãoTotal = 0;
3  para máquina em máquinas faça
4  |  tempoConclusãoTotal += mat[máquina];
5  fim
6  média = tempoConclusãoTotal/ qtdmáquinas;
7  para máquina em máquinas faça
8  |  se (mat[máquina] <= média) então continue para a próxima
   |   iteração;
9  |  para tarefa em (tarefas atribuídas a máquina) faça
10 |  |  delta = 0;
11 |  |  para máquinaCandidata em máquinas faça
12 |  |  |  se ((mat[máquinaCandidata] +
   |  |  |   etc[tarefa][máquinaCandidata]) < média) então
13 |  |  |  |  novoTempoEstimado = (mat[máquinaCandidata] +
   |  |  |  |   etc[tarefa][máquinaCandidata]);
14 |  |  |  |  se ((média - novoTempoEstimado) > delta) então
15 |  |  |  |  |  delta = média - novoTempoEstimado;
16 |  |  |  |  |  máquinaEscolhida = máquinaCandidata;
17 |  |  |  |  fim
18 |  |  |  fim
19 |  |  fim
20 |  |  se (delta > 0) então
21 |  |  |  remover tarefa da máquina;
22 |  |  |  atribuir tarefa para máquinaEscolhida;
23 |  |  fim
24 |  fim
25 fim
```

Utilizando os dados da Tabela 5 para executar o Min-Min, obtemos que o makespan seria 80 e o flowtime 140 (foi utilizado esses mesmos dados para a explicação do Min-Min anteriormente). Para encontrar a média, dividimos o flowtime pelo número de máquinas (3) e arredondamos, para facilitar o entendimento, o valor para o inteiro mais próximo. Assim, a média que utilizaremos para as etapas seguintes é 46. Dessa vez, as iterações são realizadas por máquina. Para a primeira máquina, por não ter nenhuma tarefa atribuída a ela, possui mat igual a 0, menor que a média de 46. Assim, o algoritmo avança para a iteração seguinte, na qual a máquina 2, que tem um tempo de processamento (mat) igual a 80 segundos, superior à média, pode ter suas tarefas redistribuídas para outra máquina. Para cada uma das tarefas anteriormente atribuídas para a máquina 2, é analisado se as outras máquinas se tivessem a tarefa atribuídas a ela, iria diminuir a média. Dessa forma, a máquina 1

passa a ser uma máquina candidata. Se a tarefa 1 fosse transferida para a máquina 1, o tempo de processamento resultante seria maior do que a própria média, logo essa alternativa é eliminada. Se a tarefa 2 fosse transferida para a máquina 1, o tempo de processamento resultante seria de 40 segundos, inferior à média, e a diferença entre a média e esse novo tempo possível seria de 6 segundos, superior ao delta inicial, que é 0. Logo, o valor do delta é atualizado para 6, e a máquina 1 é definida como a máquinaEscolhida.

As mesmas comparações são feitas para a máquina 3, mas se ela receber a tarefa 1 ou 2, somado com o tempo gasto com as tarefas já atribuídas a ela, o tempo seria superior a média. Portanto, a única tarefa que pode ser redistribuída foi a tarefa 2 para a máquina 1. Na iteração seguinte, ao examinar quais tarefas atribuídas à máquina 3 podem ser redistribuídas, observa-se que nenhuma tarefa satisfaz essa condição. A partir do resultado gerado pelo Max-Min, observa-se que o makespan é de 60 segundos, menor que o makespan gerado anteriormente pelo Min-Min (80), e o flowtime de 150 segundos ($40 + 50 + 60$). A Tabela 10 mostra a designação obtida pelo Min-Mean.

Tabela 10 – Distribuição das tarefas Min-Mean

Máquina	Tarefas	Tempo Total
1	2	40
2	1	50
3	3 e 4	60

Fonte: Elaborado pelo Autor (2023)

2.2.8 Min-Var

Pode-se analisar o Min-Var como uma modificação do Min-Mean porque, assim como essa heurística, o Min-Var usa a média como um primeiro filtro para avaliar se uma máquina pode melhorar seu tempo ao distribuir alguma de suas tarefas para outra máquina. Entretanto, para as máquinas que possuem o tempo de conclusão maior que a média, é calculado a distância que o tempo de conclusão da máquina está da variância dos tempos de conclusão.

O Algoritmo 8 mostra o pseudocódigo do Min-Var. Nesse caso, a variância é calculada a partir da soma dos quadrados da diferença entre o tempo de conclusão da máquina e a média dos tempos de conclusão, dividido pela quantidade de máquinas. A fórmula matemática da variância é dada por:

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n} \tag{2.1}$$

sendo que:
σ^2 = variância;
x_i = valor observado da i-ésima máquina;
$\bar{x}$ a é a média aritmética de todos os valores;
n é o número total de observações.

Algoritmo 8: Pseudocódigo do algoritmo *Min-Var*

```
1  Execute o Min-Min;
2  Calcule média e variância;
3  para máquina em máquinas faça
4      se (mat[máquina] < média) então continue para a próxima
         iteração;
5      difMatMedia = (mat[máquina]-média)²;
6      se (difMatMedia < variancia) então continue para a próxima
         iteração;
7      para tarefa em (tarefas atribuídas a máquina) faça
8          para máquinaCandidata em máquinas faça
9              se (máquinaCandidata == máquina) então
10                 continue para a próxima iteração
11             fim
12             difMatMediaPossible = ( (mat[máquinaCandidata] +
                 etc[tarefa][máquinaCandidata]) - média)²;
13             se (difMatMediaPossible > variancia) então
14                 continue para a próxima iteração
15             fim
16             se ((variancia - difMatMediaPossible) >= delta) então
17                 delta = variancia - difMatMediaPossible;
18                 máquinaEscolhida = máquinaCandidata;
19             fim
20         fim
21         se (delta > 0) então
22             remover tarefa da máquina;
23             atribuir tarefa para máquinaEscolhida;
24         fim
25     fim
26 fim
```

Para exemplificar esse algoritmo, utilizaremos os dados da Tabela 5. Assim como o Min-Mean, o Min-Var executa o Min-Min para conseguir os dados da média, 46, e da variância. A variância é uma medida que indica o grau de dispersão dos dados em torno da média. Para calculá-la, segue-se o seguinte procedimento: para cada

valor observado (mat) de cada máquina, subtrai-se a média aritmética de todos os valores, e eleva-se o resultado ao quadrado. Em seguida, soma-se todos os quadrados obtidos e divide-se pelo número total de observações. No nosso exemplo, $\sigma^2 = ((0 - 46)^2 + (80 - 46)^2 + (60 - 46)^2)/3 = 1156$.

A partir do cálculo da variância, pode-se identificar quais máquinas apresentam um escalonamento de tarefas com o tempo total que se desvia consideravelmente da média. Essas máquinas são aquelas cujo valor de $(x_i - \bar{x})^2$ é maior que a variância. Para essas máquinas, pode-se tentar redistribuir as tarefas de forma a otimizar o seu funcionamento e reduzir a dispersão dos dados. Embora a primeira máquina se enquadre nesse critério, ele possui um mat menor que a média, e não possui tarefas para serem redistribuídas para outras máquinas. A única máquina que possui um desempenho(difMatMedia) igual ou maior que a variância é a máquina 2, $(80 - 46)^2 = 1156$. As tarefas 2 e 1 foram atribuídas a ela. A máquina 1 e a 3 passam a ser uma máquinas candidatas, e para avaliar se ela pode receber uma das tarefas da máquina 2, calcula-se a diferença entre a média e o tempo de conclusão das tarefas anteriores, mat, somado ao tempo para concluir a tarefa analisada, etc., e eleva o resultado ao quadrado.

Para a tarefa 1, máquina 1:
difMatMediaPossible $=((0 + 125) - 46)^2 = 79^2 = 6.241$
Como o resultado é maior que a variância, segue para a próxima máquina.

Para a tarefa 1, máquina 3:
difMatMediaPossible $= ((60 + 115) - 46)^2 = 129^2 = 16.641$
Como o resultado é maior que a variância, segue para a próxima tarefa.

Para a tarefa 2, máquina 1:
difMatMediaPossible $= ((0 + 30) - 46)^2 = 16^2 = 256$
Por ser menor que a variância, segue para o calculo do delta
$delta = 1156 - 256 = 900$

Assim, a tarefa 2 é atribuída à máquina 1, que é a mais adequada para executá-la. Como a máquina 3, se receber a tarefa 2, terá um difMatMediaPossible maior que o da máquina 1, então ela não substitui a máquina 1 como a escolhida. Finalmente, a tarefa 2 é atribuída à máquina 1 (Tabela 11). Nas próximas iterações, não há mudança de atribuição de tarefas, porque qualquer alteração aumentaria a dispersão dos resultados em relação a média inicial. Os resultados para esse exemplo

são um makespan de 60 segundos e um flowtime de 150 segundos.

Tabela 11 – Distribuição das tarefas Min-Var

Máquina	Tarefas	Tempo Total
1	2	40
2	1	50
3	3 e 4	60

Fonte: Elaborado pelo Autor (2023)

2.2.9 Min-Max

O Min-Max é uma das heurísticas mais recentes, sendo baseada na proposta de Sahu e Chaturvedi (2011). Ela se assemelha tanto com o Max-Min quanto com o Sufferage em sua base. Em relação ao Max-Min, ao invés de atribuir a tarefa com maior tempo entre os tempos mínimos, é avaliado qual tarefa demoraria mais tempo para ser executado na máquina mais lenta. A ideia também é análoga à heurística Sufferage (que analisa se qual o impacto no resultado se for escolhida a segunda melhor máquina para processar a tarefa), mas no caso do Min-Max tenta-se encontrar qual seria o pior tempo, ou seja, no pior cenário qual tarefa seria mais impactada, e prioriza ela para a a máquina que a processaria em menor tempo. O Algortimo 9 apresenta o pseudocódigo do Min-Max.

Como exemplo, considere novamente os dados da Tabela 5. Primeiramente é calculado qual tarefa terá maior tempo para ser processada pela máquina mais lenta, no caso seria a tarefa 4 pela máquina 1, 200 segundos, portanto a primeira tarefa a ser atribuída será a 4 para a máquina 3, que executaria ela em menor tempo, 50 segundos. A segunda tarefa que demoraria mais tempo para ser executada, seria a tarefa 1 pela máquina 1, em 125 segundos. Portanto ela será a próxima máquina a ser atribuída, e a máquina que executaria ela mais rapidamente seria 2, em 50 segundos também. Seguindo as mesmas etapas, a próxima tarefa a ser atribuída seria a 3 para a máquina 3, em 60 segundos (50+10). Por fim, a tarefa 2 seria atribuída a máquina 1 que a processaria em 40 segundos.

Os resultados para esse exemplo, Tabela 12, são os mesmos do Min-Var (Tabela 11) ou seja, makespan de 60 segundos, e flowtime de 150 segundos.

Algoritmo 9: Pseudocódigo do algoritmo *Min-Max*

```
repita
    maxMinComplTime=valor mínimo negativo;
    para tarefa em tarefas faça
        maxTime=valor mínimo negativo;
        minTime=valor máximo positivo;
        se (foiRemovida[tarefa]) então
            continue para a próxima iteração;
        fim
        para maquina em máquinas faça
            se (etc[tarefa][máquina] + mat[máquina] > maxTime) então
                maxTime = etc[tarefa][máquina] + mat[máquina];
            fim
            se (etc[tarefa][máquina] + mat[máquina] < minTime) então
                minTime = etc[tarefa][máquina] + mat[máquina];
                máquinaMin = máquina;
            fim
        fim
        maxComplTime[tarefa] = maxTime;
        minComplTime[tarefa] = minTime;
        minComplMachine[tarefa] = máquinaMin;
        se (maxComplTime[tarefa] > maxMinComplTime) então
            maxMinComplTime = maxComplTime[tarefa];
            tarefaEscolhida = tarefa;
        fim
    fim
    máquinaEscolhida = minComplMachine[tarefaEscolhida];
    atribuir tarefaEscolhida para máquinaEscolhida;
até que todas as tarefas tenham sido atribuídas;
```

Tabela 12 – Distribuição das tarefas Min-Max

Distribuição das tarefas Min-Max		
Máquina	Tarefas	Tempo Total
1		40
2	1	50
3	4	60

Fonte: Elaborado pelo Autor (2023)

3 Desenvolvimento

3.1 Modificações no Simulador

Para comparar os diferentes algoritmos descritos no Capítulo 2, é necessário implementá-los em um simulador. Nesse sentido, foi utilizado o simulador desenvolvido por Verma (2010). Esse simulador, escrito em linguagem Java, já implementa alguns dos algoritmos estudados neste trabalho; porém, conforme observado nos trabalhos de Caixeta (2018) e Barra (2022), possui algumas limitações que precisaram ser resolvidas. Nesse sentido, foram realizadas algumas alterações em cima de uma versão atualizadas por Barra (2022). Tais alterações permitem ao usuário selecionar um arquivo com as instâncias, bem como na escolha das heurísticas a serem executadas, podendo optar por todas ou por uma específica. Além disso os resultados agora são armazenados em um arquivo *csv*, contendo makespan, flowtime, utilização média das máquinas, tempo computacional, e matching proximity em relação ao MET. Para facilitar a busca e a seleção, os arquivos de instâncias e de resultados estão organizados em pastas separadas.

Alguns métodos do simulador foram alterados, tais como a geração de um novo arquivo de instância, a partir dos parâmetros de quantidade de máquinas, quantidade de tarefas, heterogeneidade das tarefas e máquinas, que originalmente no simulador do Verma (2010) necessitavam de modificações diretas no código e não apresentavam o funcionamento esperado. Agora elas obtêm os dados de entrada pelo usuário, e algumas classes foram reorganizadas para facilitar o entendimento e a manutenção do código. Também foi alterado como é estabelecido o tempo para determinada máquina processar cada tarefa baseado na heterogeneidade das tarefas e máquinas, de forma que esses fatores determinem um intervalo em porcentagem de possíveis valores e não o intervalo em si como antes.

Finalmente, foram realizadas modificações na implementação de alguns algoritmos que não se encontravam de acordo com as referências teóricas. Como consequências, tais algoritmos produziam resultados mais divergentes em relação aos trabalhos de referência originais, em especial o Sufferage (MAHESWARAN *et al.*, 1999) e o Min-Var (KAMALAM; BHASKARAN, 2010). Também foram necessárias pequenas alterações nos algoritmos Min-Mean, Max-Min, MET, MCT.

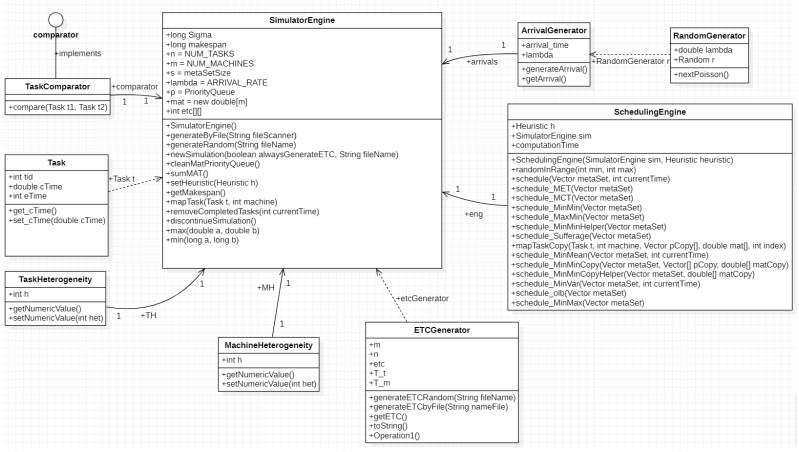
Tais modificações melhoraram consideravelmente o desempenho do simula-

dor. De fato, neste trabalho, foi possível simular a execução, em menos de 50 segundos, cenários compostos por 8192 tarefas e 256 máquinas.

3.1.1 Organização das Classes do Simulador

Conforme mencionado, o simulador foi originalmente implementado em linguagem Java. Na Figura 1, é mostrado o diagrama de classes que representa esse simulador, já com todas as modificações propostas neste trabalho.

Figura 1 – Diagrama de Classes do Simulador



Fonte: Elaborado pelo Autor (2023)

Como é possível observar pela Figura 1, a classe SimulatorEngine é a que mais possui interações com outras classes, isso porque ela concentra os atributos necessários para chamar os métodos das demais classes. É através da inicialização do objeto da classe SimulatorEngine na classe principal que é determinado como o etc será gerado, seja através de um arquivo ou pelo método de geração aleatória com base na heterogeneidade das máquinas e tarefas e outros parâmetros passados pelo usuário.

Os algoritmos de escalonamento são definidos na classe SchedulingEngine, que é invocada para toda heurística a ser simulada. É importante notar que objetos do tipo SchedulingEngine tem associação 1 para 1 com objetos SimulatorEngine, porque o SimulatorEngine armazena as informações como tempo de processamento da tarefa pela máquina, tempo total para a máquina concluir as tarefas a ela designadas e o mapeamento da tarefa definido pela heurística. Essas informações e métodos são essenciais para o escalonamento realizado no SchedulingEngine. O mapeamento da task, calculado pela heurística, é salvo em uma fila de prioridade para

cada uma das máquinas, e para isso é utilizado um taskComparator, que implementa uma interface do tipo comparator.

Algumas classes simples como machineHeterogeneity, taskHeterogeneity e task são inicializadas como atributos do SimulatorEngine, e possuem apenas métodos para definir e obter seus valores. E a classe ArrivalGenerator é utilizada para calcular quantas tarefas apresentam-se para serem atribuidas em determinado intervalo de tempo, o que influencia no calculo do tempo que a heutística utiliza para atribuir as tarefas.

4 Experimentos

4.1 Cenários avaliados

Para realizar os experimentos, foram realizadas 100 simulações utilizando 32 máquinas para processar 1024 tarefas. Diferentes cenários, alterando a heterogeneidade das máquinas e tarefas na elaboração dos dados, foram simulados e comparados entre si.

- **Heterogeneidade de máquinas:** A heterogeneidade de máquinas se refere à variação nas capacidades, recursos e características de diferentes máquinas, como velocidade do processador, quantidade de memória, espaço em disco e largura de banda da rede. Em outras palavras, diferentes máquinas podem ter diferentes configurações de hardware, o que pode afetar significativamente o desempenho de um sistema de computação. Quanto menor a heterogeneidade das máquinas, mais semelhantes em termos de processamento elas são, diferindo pouco o tempo para processar as diferentes tarefas. Já uma maior heterogeneidade de máquinas, significa que haverá maiores diferenças entre elas para, algumas serão bem mais ágeis para realizar certos tipos de tarefas do que outras (SAHU; CHATURVEDI, 2011).
- **Heterogeneidade de tarefas:** A heterogeneidade de tarefas refere-se à variação na complexidade ou na exigência de recursos de diferentes tarefas executadas em um sistema. Algumas tarefas podem ser muito simples e exigir pouco processamento e recursos de memória, enquanto outras podem ser muito complexas e exigir muito tempo de processamento e muitos recursos. Uma alta heterogeneidade nesse caso, implica que o conjunto de tarefas terá grandes diferenças entre si, análogo a diferentes aplicações. E uma baixa heterogeneidade pressupõe que as tarefas possuem requisitos parecidos, portanto, máquinas semelhantes executariam elas em termos ainda mais parecidos do que se a heterogeneidade das tarefas fosse alta (SAHU; CHATURVEDI, 2011).

Portanto, foram considerados os seguintes cenários:

- **High-High:** Alta heterogeneidade de tarefas e alta heterogeneidade de máquinas;

- **High-Low:** Alta heterogeneidade de tarefas e baixa heterogeneidade de máquinas;
- **Low-High:** Baixa heterogeneidade de tarefas e alta heterogeneidade de máquinas;
- **Low-Low:** Baixa heterogeneidade de tarefas e baixa heterogeneidade de máquinas.

4.2 Parâmetros para comparação das heurísticas

Para avaliar e comparar as heurísticas propostas neste trabalho, utilizou-se alguns parâmetros que medem o desempenho das mesmas em um cenário específico. Esses parâmetros podem auxiliar na escolha da heurística mais adequada para cada situação, levando em conta os recursos disponíveis e as tarefas a serem realizadas. O objetivo da analise desses experimentos é tornar o uso do simulador e a análise dos seus resultados mais simples e intuitivos, de modo que se possa definir qual heurística é mais apropriada para otimizar a atribuição de tarefas em função das suas características e dos recursos disponíveis.

Makespan. A métrica mais utilizada para esse tipo de comparação. Ela mostra qual o tempo que a última máquina utilizou para finalizar a última tarefa:

$$Makespan = max\{mat[i], i = 1, ..., M\}$$

mat[i] = Tempo para a máquina i concluir as tarefas que lhe foram atribuídas.; M = quantidade de máquinas

Flowtime. Este valor representa a soma dos tempos de conclusão das máquinas. Heurísticas que buscam atribuir as tarefas atribuindo mais peso ao tempo para o processamento de determinada tarefa pela máquina (etc), tendem a ter flowtimes menores, mesmo que o makespan aumente dessa forma. (SAHU; CHATURVEDI, 2011) :

$$Flowtime = \sum_{i=0}^{M} mat[i]$$

Para facilitar a visualização dos resultados em gráficos, o makespan e o flowtime foram normalizados para valores entre -1 e 1. Sendo 1 o melhor resultado e -1 o pior resultado, dessa forma quanto mais próximo de 1 melhor o resultado.

Utilização das máquinas. Utilização das máquinas é a métrica que mede o quanto as máquinas foram aproveitadas durante a execução das tarefas determinadas pelo algoritmo. Esse conceito depende da quantidade de máquinas, do tempo total para executar todas as tarefas e do tempo para finalizar a última tarefa. O resultado é um valor entre 0 e 1, sendo que quanto mais próximo de 1, maior é o aproveitamento médio das máquinas, ou seja, o quanto elas ficaram ocupadas em relação ao tempo necessário para concluir a última tarefa.

$$Utilização Máquinas. = Flowtime/(Makespan * qtd.máquinas)$$

MPR Nas tabelas, é utilizada a relação de desempenho, também conhecida como performance ratio (PR), para comparar as heurísticas com o algoritmo MET. MPR se refere a *makespan performance ratio*, utilizado para comparar o desempenho de heurísticas com o algoritmo MET em relação ao makespan. O algoritmo MET é simples e calcula o tempo que a máquina gastaria para realizar uma determinada tarefa, sem levar em conta o tempo de processamento das tarefas atribuídas anteriormente. Por isso, ele serve como um bom parâmetro para avaliar o desempenho dos outros algoritmos em relação a ele. Esse critério de comparação é calculado dividindo o makespan de uma heurística pela métrica equivalente do MET. Dessa forma, o MPR do próprio MET é 1. Resultados menores que 1, significam que são resultados melhores que ao do MET. Da mesma forma, resultados maiores que 1, significam que a métrica é pior que a do MET. A distância entre o PR dos diferentes algoritmos indica a proporção da melhora, ou piora, em comparação com o outro.

Matching proximity A heurística MET avalia apenas o tempo de processamento de determinada tarefa pelas máquinas (ETC) para definir qual máquina será responsável por qual tarefa. Portanto, essa heurística sempre atribui a máquina mais rápida para aquela tarefa. Portanto, uma métrica interessante para comparar os algoritmos é validar o quanto a soma do ETC das tarefas atribuídas é maior que o ETC equivalente do MET (XHAFA; ABRAHAM, 2010). Quanto mais tarefas forem atribuídas para máquinas que não sejam as que executariam mais rapidamente a tarefa, maior será a razão entre o ETC da heuristica e o ETC do MET.

$$matchingProximity = \frac{\sum_{i=0}^{Tasks} ETC[i][máquina[i]]}{\sum_{i=0}^{Tasks} ETC.MET[i][máquina[i]]} \tag{4.1}$$

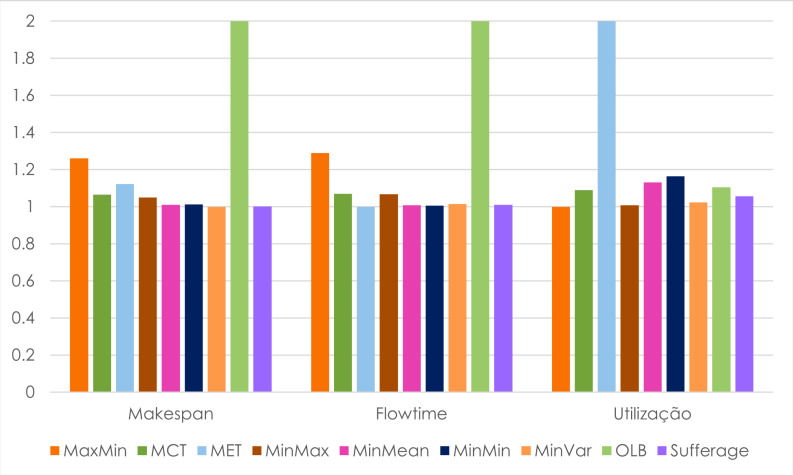
Nos gráficos das Figuras 2, 5, 8 e 11, os dados foram normalizados entre 1 e 2, sendo 1 o melhor resultado para o critério avaliado, e 2 o pior resultado para o critério avaliado, ao comparar os resultados do escalonamento de cada heurística. Como a utilização média era o único critério que quanto maior e mais próximo de 1, melhor o aproveitamento médio das máquinas, para normalizar os resultados divide-se 1 pela utilização média, assim inverte os resultados para que o menor resultado fosse da heurística que melhor aproveitou a capacidade das máquinas, e quanto pior o resultado, maior ele seria.

Para normalizar os resultados entre 1 e 2 foi utilizado o seguinte calculo:

$$Critério\_Normalizado = \frac{(maiorResultado - menorResultado)}{(resultadoAvaliado - menorResultado)} + 1 \qquad (4.2)$$

4.2.1 Cenário 1 - High-High

Figura 2 – Makespan, flowtime e utilização Cenário - High-High

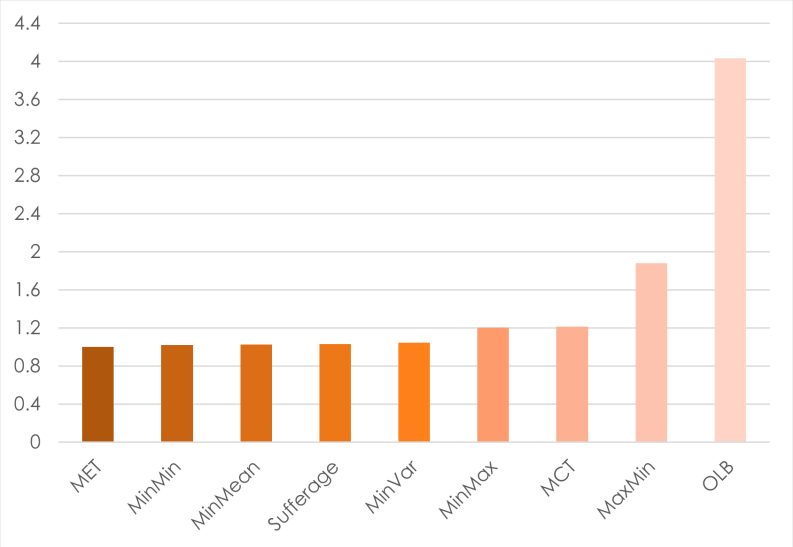


Fonte: Elaborado pelo Autor (2023)

Nesse cenário, com alta heterogeneidade de tarefas e máquinas, podemos ver na Tabela 13 que Min-Var obteve o melhor resultado em relação ao makespan, e Sufferage o segundo melhor resultado. Ao analisar o MPR, (Figura 4), Min-Var obteve um makespan cerca de 27% melhor que o MET. Já o pior algoritmo para esse cenário foi o OLB, quase 2 vezes mais lento que o MET em relação ao makespan. O outro algoritmo que teve um makspan pior que o MET além do OLB foi o Max-Min (cerca de 30% mais lento que o MET).

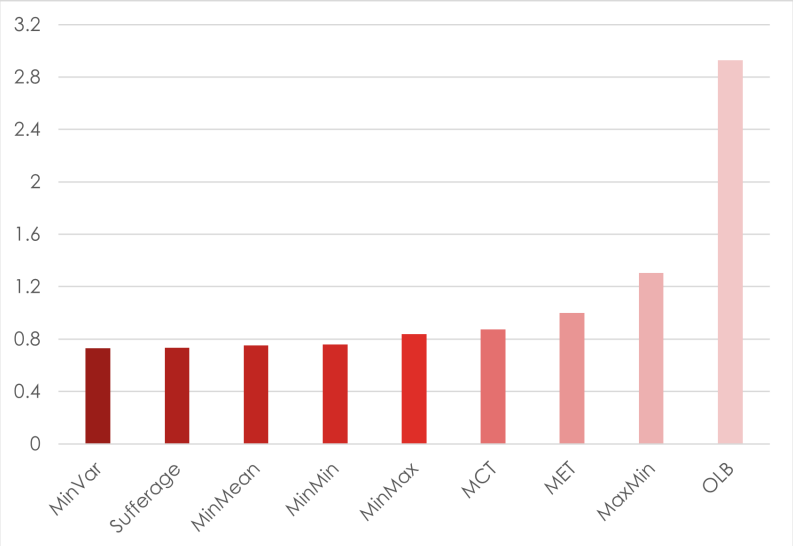
Embora o Min-Min tenha o quarto menor makespan, se analisarmos o matching proximity (Figura 3), o Min-Min obtém o segundo melhor resultado, logo

Figura 3 – Matching Proximity - Cenário High-High



Fonte: Elaborado pelo Autor (2023)

Figura 4 – MPR - Cenário High-High



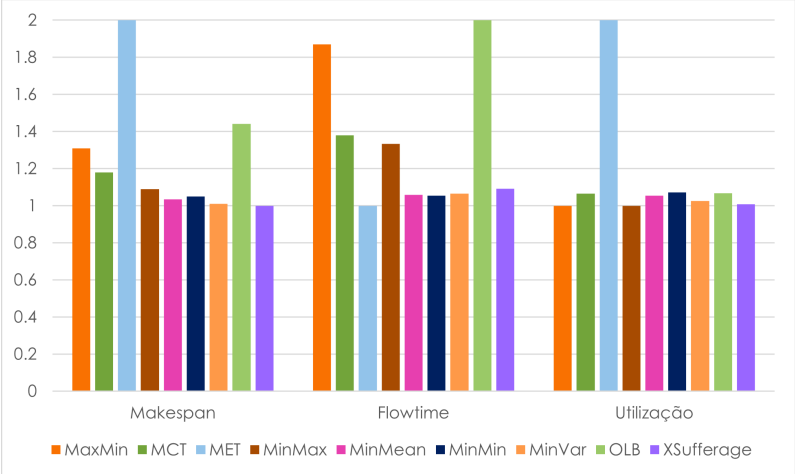
Fonte: Elaborado pelo Autor (2023)

atrás do MET, que sempre tem o melhor resultado possível para essa métrica. E o terceiro mais rápido sendo o Sufferage.

Uma heurística que se destaca na utilização média dos recursos, é o Max-Min, superando o OLB para esse cenário, por conseguir distribuir de maneira mais inteligente as tarefas de forma que as máquinas fiquem mais ocupadas. Considerando as três métricas para esse cenário High-High, as heurística Min-Var e a Min-Mean foram as que mais se destacaram, por terem makespans baixos, conseguirem manter a utilização das máquinas alta e serem relativamente mais simples que o Sufferage, que em questão de tempo computacional foi o que teve piores resultados.

4.2.2 Cenário 1 - High-Low

Figura 5 – Makespan, flowtime e utilização Cenário - High-High



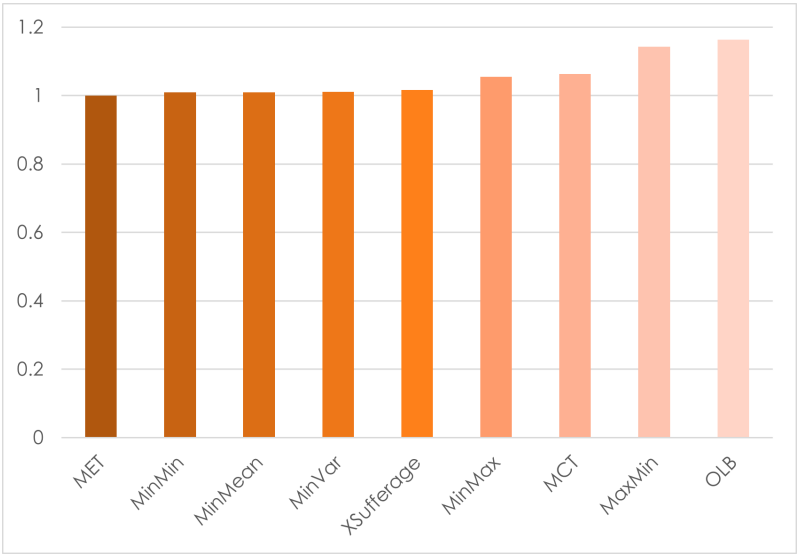
Fonte: Elaborado pelo Autor (2023)

O OBL nesse cenário tem um makespan melhor que o MET, mas continua com resultados ruins para flowtime e matching proximity. Vale notar que a distância entre o matching proximity do MET e dos outros algoritmos diminuiu significativamente nesse cenário, sendo que o pior resultado foi apenas 16% pior que o MET (Tabela 14) e no cenário High-High foi mais de 300% (Tabela 13).

O Sufferage teve uma pequena melhora em relação ao MinVar de acordo com o makespan, e embora o tempo computacional médio dele foi maior que o Min-var, ele não é mais a heurística que mais demorou para ser executado, ficando essa posição com o Min-Max (56.33 milissegundos).

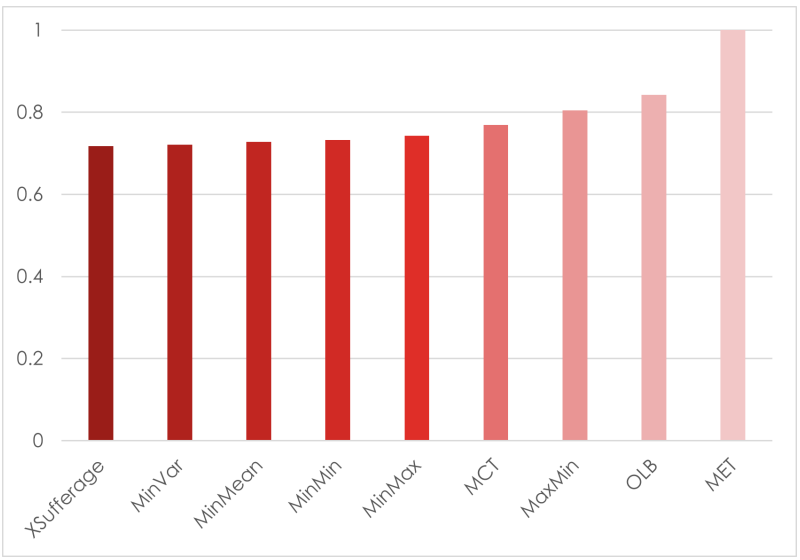
Nesse cenário, o algoritmo que teve pior performance em relação ao makespan foi o MET (Figura 5). Isso pode ser explicado pelo fato de que a diferença

Figura 6 – Matching Proximity - Cenário High-Low



Fonte: Elaborado pelo Autor (2023)

Figura 7 – MPR - Cenário High-Low

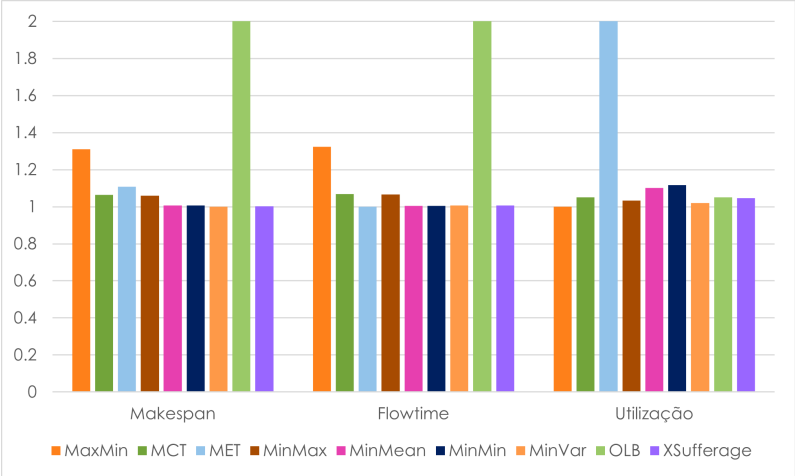


Fonte: Elaborado pelo Autor (2023)

entre as máquinas é pequena, porém o MET tende a selecionar as mesmas máquinas para executar as tarefas, deixando assim mais máquinas com capacidade de processamento similar ociosas.

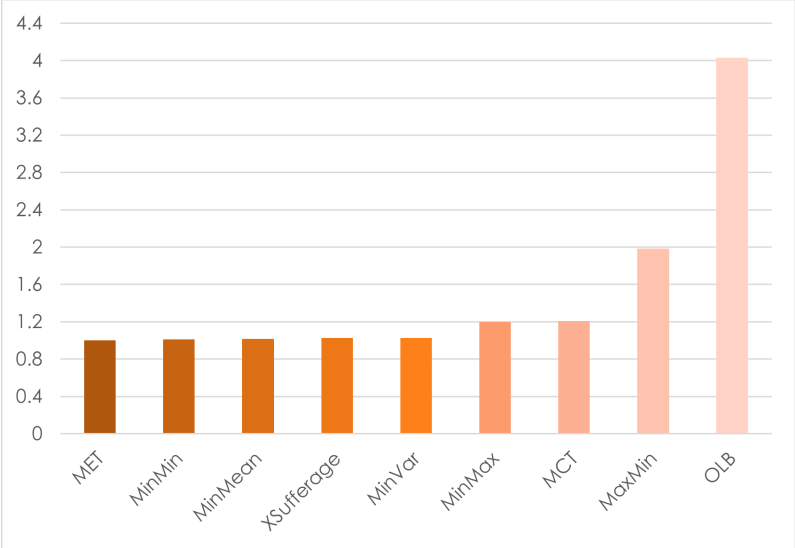
4.2.3 Cenário 1 - Low-High

Figura 8 – Makespan, flowtime e utilização Cenário - Low-High



Fonte: Elaborado pelo Autor (2023)

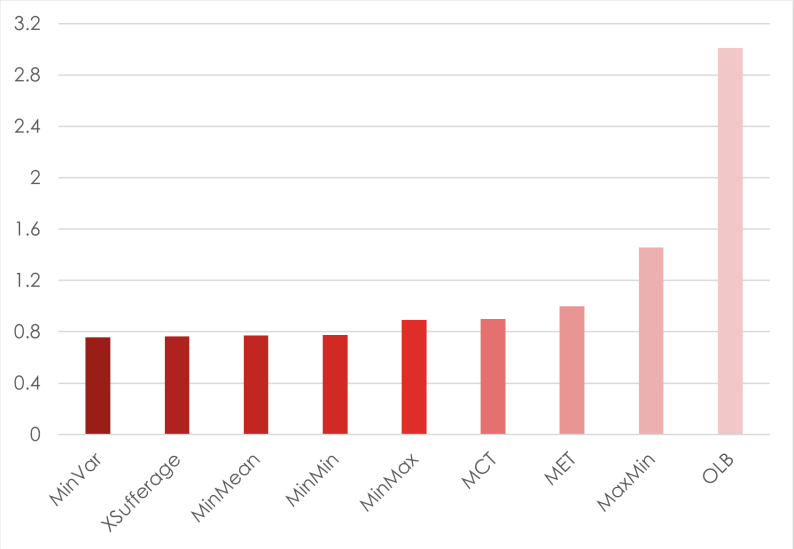
Figura 9 – Matching Proximity - Cenário Low-High



Fonte: Elaborado pelo Autor (2023)

Os resultados obtidos com essa heterogeneidade não apresentam uma grande diferença em relação ao cenário High-High, no que diz respeito à ordem de clas-

Figura 10 – MPR - Cenário Low-High



Fonte: Elaborado pelo Autor (2023)

sificação do makespan e do flowtime, ou seja, para a maioria das heuristicas não houveram mudanças de posições (Figura 8).

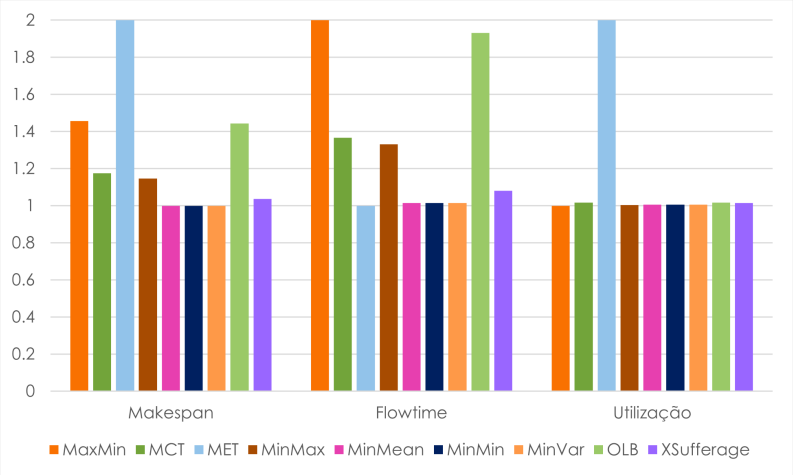
Analisando os resultados do makespan, flowtime e utilização média, os resultados foram mais constantes entre as diversas simulações, o que é visto pelo desvio padrão dos resultados (Tabela 15). Todavia, por ter tarefas mais parecidas e uma alta diversidade de máquinas, o tempo computacional da maioria dos algoritmos (Min-Var, Min-Min, Min-Mean, Max-Mean, Sufferage e Min-Max) aumentou significativamente.

4.2.4 Cenário 1 - Low-Low

Nessa configuração, alguns algoritmos atingem o mesmo resultado de makespan e flowtime, sendo eles Min-Min, Min-Mean e Min-Var de acordo com as Figuras 12 e 13. Como Min-Mean e o Min-Var são heurísticas que utilizam do resultado do Min-Min para a partir dele tentar redistribuir as tarefas em máquinas que estão mais ociosas, considerando a média ou a variância dos tempos de conclusão, é esperado que em um cenário com pouca variação entre as máquinas e as tarefas, essas heurísticas não tenham possibilidade de redistribuição sem piorar os indicadores.

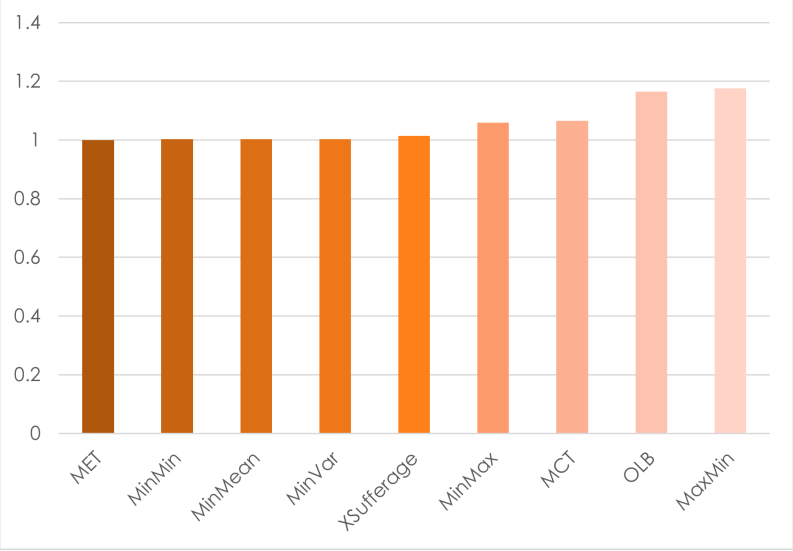
Assim como o cenário High-Low (Figura 7, o MET obteve o pior makespan (Figura 13). Vale notar que o OLB e o MaxMin tiveram uma leve piora nesse critério quando comparado com os outros algoritmos. Ao analisar o gráfico de Matching

Figura 11 – Makespan, flowtime e utilização Cenário - Low-Low



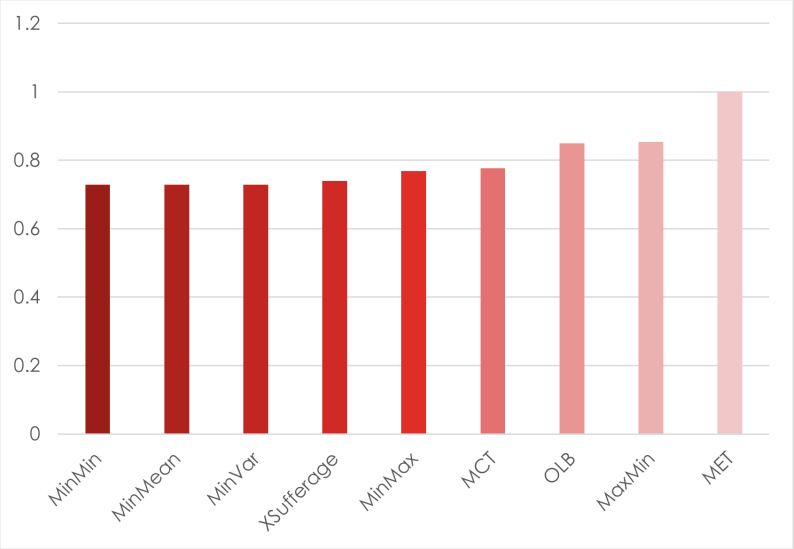
Fonte: Elaborado pelo Autor (2023)

Figura 12 – Matching Proximity - Cenário Low-Low



Fonte: Elaborado pelo Autor (2023)

Figura 13 – MPR - Cenário Low-Low



Fonte: Elaborado pelo Autor (2023)

proximity (Figura 12), pode-se notar uma similaridade bem alta entre os algoritmos em relação a soma do tempo de conclusão das máquinas, com os algoritmos mais distantes (OLB e Max-Min) sendo menos de 18% piores que o MET.

5 Conclusões

Ao comparar as nove heurísticas em cenários diversos para atribuição das tarefas em um sistema de grade computacioanl, pode-se compreender melhor suas diferenças, quais situações podem ser melhor aplicados e como algoritmos de escalonamento mais complexos podem ser criados ou adaptados a partir do entendimento dos pontos fortes e fracos desses algoritmos clássicos.

Os algoritmos que derivam do Min-Min (Min-Var e Min-Mean) foram os que obtiveram melhores resultados para quase todos os cenários se analisado o makespan. Além disso, embora o Sufferage obteve resultados bem próximos, o tempo computacional médio em relação aos outros algorimos, com exceção do Max-Min e do Min-Max, foi consideravelmente maior. Portanto, a menos que se conheça bem o cenário a ser aplicado, não é recomendável utilizar a heurística Sufferage. Em cenários com tarefas custosas, que podem ser ainda mais dispendiosas caso não sejam atribuídas para a máquina mais rápida, o Sufferage deve ser melhor empregado, compensando assim seu custo para atribuir as tarefas.

O algoritmo MET, embora sempre consiga o menor flowtime, e seja rápido na atribuição de tarefas, não considera a ocupação de cada máquina para designar a tarefa, portanto utiliza poucas máquinas que têm a disposição, e tende a demorar para completar todas as tarefas.

Em relação a utilização média das máquinas, com exceção do MET, todas as heurísticas tiveram resultados altos, destacando-se o Max-Min e o Min-Max, que consiguiram manter a utilização dos recursos bem alta até finalizar a última tarefa.

Embora os experimentos realizados neste trabalho tenham sido feitos com cenários relativamente controlados, é encorajado o uso do simulador para comparar os algoritmos em cenários mais semelhantes ao que seriam aplicados. É relevante destacar que com as alterações feitas no simulador, foi facilitado para o usuário escolher um arquivo com os dados de tempo gasto por cada máquina para realizar cada tarefa, ou para o usuário criar novos cenários digitando apenas os parâmetros necessários. E também é encorajado a inclusão de novos algoritmos no simulador em trabalhos futuros.

Referências

BARRA, M. V. de A. *Comparação de algoritmos para o problema de escalonamento de tarefas em grades computacionais*. 46 p. Monografia (Trabalho de Conclusão de Curso) — Universidade Federal de Uberlândia, Uberlândia, MG, 2022. Disponível em: <https://repositorio.ufu.br/handle/123456789/36150>. Acesso em: 15 jun. 2023. Citado 3 vezes nas páginas 11, 12 e 31.

CAIXETA, C. de M. *Implementação de modificações em um simulador de escalonamento de tarefas em sistemas distribuídos*. 26 p. Monografia (Trabalho de Conclusão de Curso) — Universidade Federal de Uberlândia, Uberlândia, MG, 2018. Disponível em: <https://repositorio.ufu.br/handle/123456789/23174>. Acesso em: 15 jun. 2023. Citado na página 31.

DABROWSKI, C. Reliability in grid computing systems. *Concurrency and Computation: Practice & Experience* — A Special Issue from the Open Grid Forum, John Wiley and Sons Ltd., West Sussex, United Kingdom, v. 21, n. 8, p. 927–959, jun. 2009. Disponível em: <https://doi.org/10.1002/cpe.1410>. Acesso em: 10 jun. 2023. Citado na página 15.

DONG, F.; AKL, S. G. *Scheduling Algorithms for Grid Computing: State of the Art and Open Problems*. 45 p. Monografia (Relatório Técnico) — Queen's University, Kingston, Ontario, jan. 2006. Disponível em: <https://research.cs.queensu.ca/TechReports/Reports/2006-504.pdf>. Acesso em: 15 jun. 2023. Citado 2 vezes nas páginas 13 e 14.

KAMALAM, G. K.; BHASKARAN, V. M. A new heuristic approach: Min-mean algorithm for scheduling meta-tasks on heterogeneous computing systems. *International Journal of Computer Science and Network Security*, v. 10, n. 1, p. 24–31, jan. 2010. Disponível em: <http://paper.ijcsns.org/07_book/201001/20100104.pdf>. Acesso em: 21 jun. 2023. Citado 2 vezes nas páginas 14 e 31.

MAHESWARAN, M.; ALI, S.; SIEGAL, H. J.; HENSGEN, D.; FREUND, R. F. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: *Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)*. IEEE Comput. Soc, 1999. Disponível em: <https://doi.org/10.1109/HCW.1999.765094>. Acesso em: 21 jun. 2023. Citado na página 31.

NESMACHNOW, S.; CANCELA, H.; ALBA, E. Heterogeneous computing scheduling with evolutionary algorithms. *Soft Computing*, Springer Science and Business Media LLC, v. 15, n. 4, p. 685–701, mar. 2010. Disponível em: <https://doi.org/10.1007/s00500-010-0594-y>. Acesso em: 21 jun. 2023. Citado na página 15.

SAHU, R.; CHATURVEDI, A. K. Many-objective comparison of twelve grid scheduling heuristics. *International Journal of Computer Applications*, v. 13, n. 6,

p. 9–17, jan. 2011. Disponível em: <http://dx.doi.org/10.5120/1787-2467>. Acesso em: 19 jun. 2023. Citado 3 vezes nas páginas 29, 34 e 35.

SINGH, S.; BAWA, R. K. An efficient job scheduling algorithm for grid computing. In: *Proceedings of the International Conference on Advances in Computing and Artificial Intelligence*. ACM, 2011. Disponível em: <https://doi.org/10.1145/2007052.2007097>. Acesso em: 19 jun. 2023. Citado na página 11.

SOUSA, J. J. R. *Um algoritmo genético híbrido para otimização do escalonamento de tarefas independentes em máquinas heterogêneas*. 60 p. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia, MG, set. 2022. Disponível em: <https://doi.org/10.14393/ufu.di.2022.582>. Acesso em: 19 jun. 2023. Citado na página 15.

VERMA, A. *Distributed Scheduling Simulator*. 2010. Disponível em: <https://github.com/dapurv5/distributedscheduling>. Acesso em: 15 fev. 2023. Citado 3 vezes nas páginas 11, 12 e 31.

XHAFA, F.; ABRAHAM, A. Meta-heuristics for grid scheduling problems. In: *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 2008. p. 1–37. Disponível em: <https://doi.org/10.1007/978-3-540-69277-5_1>. Acesso em: 18 jun. 2023. Citado 2 vezes nas páginas 13 e 16.

_____. Computational models and heuristic methods for grid scheduling problems. *Future Generation Computer Systems*, Elsevier BV, v. 26, n. 4, p. 608–621, abr. 2010. Disponível em: <https://doi.org/10.1016/j.future.2009.11.005>. Acesso em: 18 jun. 2023. Citado 2 vezes nas páginas 11 e 36.

ZHANG, Y.; INOGUCHI, Y.; SHEN, H. A dynamic task scheduling algorithm for Grid computing system. In: *Parallel and Distributed Processing and Applications*. Springer Berlin Heidelberg, 2004. p. 578–583. Disponível em: <https://doi.org/10.1007/978-3-540-30566-8_69>. Acesso em: 18 jun. 2023. Citado na página 11.

APÊNDICE A – Tabelas

Neste capítulo, são apresentados todos os resultados numéricos obtidos no experimento. Esses resultados foram apresentados na forma de gráficos no Capítulo 4; nesse caso, os valores foram normalizados. Aqui, os dados são mostrados sem qualquer normalização, representando o real valor obtido nas simulações.

Tabela 13 – Resultados cenário High-High

Heuristica	Makespan	Matching Proximity	MPR	FlowTime	Utilização	Tempo Computacional (ms)
MET	114668.487 ± 7.99%	1.000	1.000	2540197.55 ± 1.65%	0.697 ± 7.31%	0.13 ± 280.638%
MCT	100178.004 ± 2.04%	1.211	0.874	3075919.63 ± 1.72%	0.96 ± 1.06%	0.14 ± 267.643%
MinMin	86872.256 ± 1.97%	1.019	0.758	2586282.91 ± 1.67%	0.931 ± 1.12%	19.29 ± 39.903%
Sufferage	84165.554 ± 1.98%	1.031	0.734	2617941.12 ± 1.66%	0.973 ± 1.31%	67.12 ± 18.154%
MinMean	86179.147 ± 2.12%	1.025	0.752	2602289.23 ± 1.68%	0.944 ± 1.26%	19.06 ± 25.519%
MaxMin	149543.101±1.90%	1.877	1.305	4767104.1±1.89%	0.997±0.08%	36.1 ± 23.254%
MinVar	83758.95±1.78%	1.042	0.731	2645034.53±1.73%	0.987±0.46%	18.93 ± 22.266%
OLB	335577.65±2.74%	4.031	2.927	10238808.6±2.57%	0.954±0.92%	0.15 ± 238.048%
MinMax	96131.947±1.76%	1.204	0.839	3057322.49±1.77%	0.994±0.14%	65.78 ± 9.653%

Fonte: Elaborado pelo Autor (2023)

Tabela 14 – Resultados cenário High-Low

Heuristica	Makespan	Matching Proximity	MPR	FlowTime	Utilização	Tempo Computacional (ms)
MET	390516.267±6.89%	1	1	8780898.91±1.57%	0.706±6.25%	0.13 ± 320.041%
MCT	300016.235±1.55%	1.063	0.769	9325924.7±1.59%	0.972±0.42%	0.15 ± 272.846%
MinMin	285720.719±1.59%	1.009	0.732	8858280.58±1.55%	0.969±0.34%	18.52 ± 35.519%
XSufferage	280137.728±1.55%	1.016	0.718	8914263.2±1.58%	0.995±0.13%	48.75 ± 21.696%
MinMean	283994.428±1.61%	1.01	0.728	8865225.8±1.55%	0.976±0.45%	19.77 ± 25.239%
MaxMin	314241.372±1.69%	1.143	0.805	10029489.27±1.68%	0.998±0.07%	35 ± 23.544%
MinVar	281247.937±1.55%	1.011	0.721	8876157.08±1.55%	0.987±0.09%	19.77 ± 23.691%
OLB	328820.991±1.53%	1.164	0.843	10216269.18±1.60%	0.971±0.56%	0.11 ± 284.446%
MinMax	290133.09±1.61%	1.055	0.743	9259134.56±1.61%	0.998±0.05%	56.33 ± 12.473%

Fonte: Elaborado pelo Autor (2023)

Tabela 15 – Resultados cenário Low-High

Heuristica	Makespan	Matching Proximity	MPR	FlowTime	Utilização	Tempo Computacional (ms)
MET	109651.183±5.63%	1	1	2545458.98±0.62%	0.728±5.53%	0.22 ± 218.75%
MinMin	84793.82±1.07%	1.012	0.774	2574523.35±0.65%	0.949±0.85%	30.52 ± 44.454%
MinMean	84474.594±1.06%	1.014	0.771	2578571.33±0.64%	0.954±0.80%	30.8 ± 53.455%
MCT	98739.622±1.19%	1.205	0.901	3065230.4±1.00%	0.971±0.73%	0.2 ± 200%
OLB	330230.888±1.65%	4.028	3.012	10251716.34±1.68%	0.971±0.46%	0.2 ± 200%
XSufferage	83721.504±0.95%	1.024	0.764	2605973.09±0.66%	0.973±0.69%	100.52 ± 39.569%
MinMax	97604.742±0.98%	1.198	0.891	3049422.7±0.94%	0.977±0.47%	106.87 ± 38.526%
MinVar	82953.742±1.14%	1.024	0.757	2605929.94±0.74%	0.982±0.90%	29.46 ± 53.271%
MaxMin	159522.861±1.28%	1.983	1.455	5045388.49±1.28%	0.989±0.19%	62.83 ± 61.819%

Fonte: Elaborado pelo Autor (2023)

Tabela 16 – Resultados cenário Low-Low

Heuristica	Makespan	Matching Proximity	MPR	FlowTime	Utilização	Tempo Computacional (ms)
MET	383567.175±6.25%	1	1	8794032.08±0.25%	0.72±6.07%	0.13 ± 300.985%
MCT	297758.033±0.38%	1.065	0.777	9359058.74±0.32%	0.983±0.21%	0.1 ± 331.663%
MinMin	279414.524±0.39%	1.003	0.729	8817264.27±0.25%	0.987±0.32%	20.28 ± 38.323%
XSufferage	283272.215±0.36%	1.014	0.739	8917028.33±0.26%	0.984±0.22%	55.89 ± 25.131%
MinMean	279414.524±0.39%	1.003	0.729	8817264.27±0.25%	0.987±0.32%	20.36 ± 24.307%
MaxMin	326929.571±0.42%	1.176	0.853	10339480.09±0.36%	0.989±0.18%	37.18 ± 22.706%
MinVar	279414.524±0.39%	1.003	0.729	8817264.27±0.25%	0.987±0.32%	20.77 ± 33.674%
OLB	325690.595±0.49%	1.164	0.85	10234610±0.39%	0.983±0.29%	0.2 ± 200%
MinMax	294628.503±0.39%	1.059	0.769	9307135.92±0.35%	0.988±0.15%	65.14 ± 19.558%

Fonte: Elaborado pelo Autor (2023)