UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Moreira de Camargo

Comparação de algoritmos para o problema de escalonamento de tarefas em grades computacionais

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Moreira de Camargo

Comparação de algoritmos para o problema de escalonamento de tarefas em grades computacionais

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 Ciência da Computação.

Orientador: Paulo Henrique Ribeiro Gabriel

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2023

Dedico esse trabalho para meus amigos, familiares e professores.

Agradecimentos

...

Resumo

Resumo

Palavras-chave: Escalonamento de Tarefas. Grades Computacionais. Modelo ETC. Heurísticas.

Abstract

Abstract

Keywords: Task scheduling. Computational Grids. ETC Model. Heuristics.

Sumário

Lista de algoritmos

Lista de ilustrações

Lista de tabelas

1 Introdução

1.1 Contextualização e Motivação

(BRAUN *et al.*, 2001; XHAFA; ABRAHAM, 2010). Verma (2010) – Exemplo de citação - Trocar as citações

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[1].

Por se tratar de um problema NP-Completo (detalhar mais), é fácil conferir a exatidão das soluções encontradas, mas não existe uma maneira de encontrar a melhor solução possível sem verificar todas as possibilidades. Consequentemente, para que se encontre soluções boas, ou quase ótimas, em um tempo menor, é necessário aplicar algoritmos de escalonamento que utilizem funções heurísticas [3] (detalhar mais). Todavia, essas heurísticas precisam se adaptar a heterogeneidade dos recursos disponíveis e das tarefas as serem realizas em pouco tempo e, se conhecer as aplicações e máquinas disponíveis, aplicar heurísticas que consumam menos recurso para escalonar as tarefas e achem boas soluções para cenários com características em comum. 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.

1.2 Objetivos

O objetivo principal desse trabalho é analisar e propor alterações do simulador implementado por Verma[7] e modificado por Barra[8], 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 obtidos 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. Foi refeito o algoritmo para o Sufferage e Min-Var, e pequenas modificações para melhorar a performance em Min-Mean, Max-Min, MET, MCT.
- Propor um novo algoritmo (está com nome de Min-Max, mas não é o algoritmo clássico para tomada de decisão)

1.3 Organização do Trabalho

2 Fundamentação Teórica

2.1 Visão Geral do Escalonamento de Tarefas

2.2 Métricas para Comparação dos Algoritmos

Os algoritmos analisados e comparados nesta monografia têm particularidades e vantagens em determinados aspectos, se comparados aos demais. Por isso, algumas métricas serão utilizadas com o objetivo de se obter uma comparação dos algoritmos:

Makespan. Essa é a métrica mais utilizada para a comparação de algoritmos

$$makespan = max\{C_{[i]}, i = 1, ..., M\}$$

Flowtime. Este valor representa a soma dos tempos de conclusão das máquinas. (SAHU; CHATURVEDI, 2011), sendo definido por:

$$F = \sum_{i=0}^{M} C_i$$

Tempo de Computação. Essa métrica representa o tempo em que o algoritmo levou para realizar o escalonamento, ou seja, o tempo para designar todas as tarefas para as máquinas. Esse valor geralmente é bem pequeno, na ordem de microssegundos, mas é interessante ser levado em conta, pois em um cenário real, o algoritmo é executado diversas vezes.

3 Desenvolvimento

3.1 Técnicas para Comparação dos Algoritmos

3.2 Modificações no Simulador

3.3 Algoritmos de Escalonamento de Tarefas

- **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

3.3.1 Minimum Execution Time (MET)

Este método pode ser considerado o mais simples, porque utiliza apenas os dados de qual máquina irá realizar a tarefa mais rapidamente, mas não utiliza dados sobre o tempo que a máquina irá completar as outras tarefas já designadas a ela. Esta heurística também é conhecida como Limited Best Assignment (LBA) ou User Directed Assignment (UDA).

Algoritmo 1: Pseudocódigo do algoritmo *Minimum Execution Time(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
```

Neste pseudocódigo, 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.

3.3.1.1 Exemplo Simples MET

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

A partir dos dados da tabela anterior, podemos verificar como o algoritmo MET comporta-se em cada uma das iterações: Inicia-se pela primeira tarefa e verifica-se o tempo que cada máquina demandaria para executá-la. Escolhe-se 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.

Distribuição das tarefas MET		
Máquina	Tarefas	Tempo Total
1	Nenhuma	0
2	Nenhuma	0
3	1, 2 e 3	145

Portanto o mat da máquina 1 seria 0, o mat da máquina 2 seria 0, e o mat da máquina 3 seria a soma dos tempos para executar a tarefa 1, 2 e 3 115+20+10, ou seja 15. Por ser o maior mat entre as três máquinas, o makespan (tempo para finalizar a última das tarefas) seria de 145, assim como o flowtime (tempo somado para finalizar todas as tarefas).

3.3.2 OLB (Opportunistic Loading Balancing)

Este método 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 má-

quina que obteria o menor tempo para executar determinada tarefa, o que afeta negativamente os resultados como o makespan e o flowtime.

Algoritmo 2: Pseudocódigo do algoritmo *Opportunistic Loading Balancing(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
```

Neste pseudocódigo, é utilizado o **mat[máquina]**, que é o tempo que a máquina irá realizar o processamento das tarefas atribuídas anteriormente a ela.

3.3.2.1 Exemplo Simples OLB

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

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.

Distribuição das tarefas OLB		
Máquina	Tarefas	Tempo Total
1	1	125
2	2	30
3	3	10

Portanto o mat da máquina 1 seria 125, o mat da máquina 2 seria 30 e o mat da máquina 3 seria 10. a Por ser o maior mat entre as três máquinas, o makespan seria de 125. E o tempo das execuções somadas, flowtime, seria 165.

3.3.3 Minimum Completion Time (MCT)

Esta heurística leva em conta dados do processamento da tarefa atual (**etc[tarefa][máquina]**) e o tempo para finalizar o processamento das tarefas que foram incumbidas anteriormente para determinada máquina (**mat[máquina]**). Este método, também é conhecido como fast greedy, proposto inicialmente para sistemas SmartNet (Freund et al., 1998). (Immediate$_m$ode$_s$cheduling$_i$n$_g$rid$_s$ystems.pdf)

Algoritmo 3: Pseudocódigo do algoritmo *Minimum Completion Time (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
```

3.3.3.1 Exemplo Simples MCT

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

Para a primeira tarefa, a escolha da máquina é igual ao MET, porque nenhuma das máquinas possuem 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.

Distribuição das tarefas MCT		
Máquina	Tarefas	Tempo Total
1	Nenhuma	0
2	2	43
3	3	115

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).

3.3.4 Min-Min

É uma heurística parecida com MCT, mas o tempo Mínimo é calculado 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

Algoritmo 4: Pseudocódigo do algoritmo *Min-Min*

```
1  repita
2      máquinaEscolhida = 0 tarefaEscolhida = 0 tempoMinimo = valor
        máximo positivo para tarefa em tarefas faça
3          se (foiRemovida[tarefa]) então
4              continue para a próxima iteração;
5          fim
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     foiRemovida[tarefaEscolhida] = Verdadeiro;
16 até houver tarefas não atribuidas;
```

3.3.4.1 Exemplo Simples Min-Min

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

Neste e nos próximos exemplos, foi modificado a tabela de dados para facilitar a análise de heurísticas mais complexas e suas respectivas particularidades. 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 primeira iteração encontra que a tarefa com menor tempo mínimo para ser 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, $<$ o tempo para 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.

Distribuição das tarefas Min-Min		
Máquina	Tarefas	Tempo Total
1	Nenhuma	0
2	2 e 1	80
3	3 e 4	60

Analisando o resultado gerado pelo Min-Min, verificamos que o makespan é de 80 segundos, e o flowtime de 140 segundos (80+60).

3.3.5 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.

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 houver tarefas não atribuidas;
```

3.3.5.1 Exemplo Simples Max-Min

Dados iniciais Max-Min			
Tarefa	Máquina 1	Máquina 2	Máquina 3
1	125	50	115
2	40	30	20
3	50	13	10
4	200	100	50

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 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.

Distribuição das tarefas Max-Min		
Máquina	Tarefas	Tempo Total
1	3	50
2	1	50
3	2 e 4	70

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

3.3.6 Sufferage

Essa heurística 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.

3.3.6.1 Exemplo Simples Sufferage

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

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. 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

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] < tempoMínimo1) então
21                 tempoMínimo2 = etc[tarefa][máquina] + mat[máquina];
22                 máquina2 = máquina;
23             fim
24         fim
25         sufferage = tempoMinimo1 - tempoMinimo2;
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 houver tarefas não atribuidas;
```

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).

Distribuição das tarefas Sufferage		
Máquina	Tarefas	Tempo Total
1	1	125
2	2 e 3	43
3	4	50

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).

3.3.7 Min-Mean

Essa heurística tenta 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 $>$ 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

3.3.7.1 Exemplo Simples Min-Mean

Dados iniciais Min-Mean			
Tarefa	Máquina 1	Máquina 2	Máquina 3
1	125	130	115
2	40	30	20
3	50	13	10
4	200	100	50

Utilizando os dados da tabela 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,

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
```

por não ter nenhuma tarefa atribuída a ela, possui mat igual a 0, < 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

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

A partir do resultado gerado pelo Max-Min, observa-se que o makespan é de 60 segundos, $<$ o makespan gerado anteriormente pelo Min-Min (80), e o flowtime de 150 segundos (40+50+60).

3.3.8 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 $>$ a média, é calculado a distância que o tempo de conclusão da máquina está da variância dos tempos de conclusão. 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:

$$\sigma^2 = \sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{3.1}$$

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 continue para a
                 próxima iteração;
10             difMatMediaPossible = ( (mat[máquinaCandidata] +
                 etc[tarefa][máquinaCandidata]) - média)²;
11             se (difMatMediaPossible > variancia) então continue para a
                 próxima iteração;
12             senão
13                 (difMatMediaPossible <= variancia)
14             fim
15             se ((variancia - difMatMediaPossible) >= delta) então
16                 delta = variancia - difMatMediaPossible;
17                 máquinaEscolhida = máquinaCandidata;
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
```

3.3.8.1 Exemplo Simples Min-Var

Dados iniciais Min-Var			
Tarefa	Máquina 1	Máquina 2	Máquina 3
1	125	50	115
2	40	30	20
3	50	13	10
4	200	100	50

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, $\sigma^2 = ((0\text{-}46)^2 + (80\text{-}46)^2 + (60\text{-}46)^2)/3 = 1156$

A partir do cálculo da variância, pode-se identificar quais máquinas apresentam um desempenho mais distante da variância. 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 igual ou maior que a variância é a máquina 2, $(80\text{-}46)^2 = 1156$. 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. E a tarefa 2 é por fim atribuída a máquina 1. Nas próximas iterações, não há mudança de atribuição de tarefas, porque qualquer alteração aumentaria a dispersão dos resultados em relação a média inicial.

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

Os resultados para esse exemplo são os mesmos do Min-Mean, makespan de 60 segundos, e flowtime de 150 segundos.

4 Experimentos

4.1 Parâmetros utilizados

Para a realização da simulação, alguns parâmetros são necessários para retratar de modo mais fiel possível a realidade. Quando se trata de um cenário real, não se pode simplesmente assumir que as máquinas terão o mesmo desempenho e, ainda que tenham desempenhos diferentes. Logo, é importante considerar a variação de seus desempenhos, pois isto também fará diferença para os resultados de cada algoritmo de escalonamento. Visto isso, é necessário compreender o conceito de heterogeneidade de tarefas e heterogeneidade de máquinas, delineados na sequência.

- **Heterogeneidade de Tarefas:** Representa a variação dos tempos estimados de processamento das tarefas dada uma máquina. Em ambientes com alta heterogeneidade de tarefas, tipos diferentes de tarefas são submetidas para serem executadas no sistema, podendo ser programas simples ou tarefas largas e complexas que exijam tempos de processamento grandes para serem processados. Já em ambientes com baixa heterogeneidade de tarefas, as tarefas submetidas para serem processadas possuem complexidades similares, fazendo com que os tempos estimados de processamento sejam parecidos (SAHU; CHATURVEDI, 2011).
- **Heterogeneidade de Máquinas:** Representa a variação dos tempos estimados de processamento das tarefas pelas máquinas. Portanto, um ambiente que tenha *recursos* similares, terá, para cada tarefa, tempos estimados de processamento pelas máquinas bem parecidos, resultando em uma baixa heterogeneidade de máquinas. Já um ambiente que tenha *recursos* computacionais de tipos e capacidades muito diferentes, terá, para cada tarefa, tempos estimados de processamento pelas máquinas muito diferentes, resultando em uma alta heterogeneidade de máquinas (SAHU; CHATURVEDI, 2011).

Partindo do exposto, pode-se definir quatro cenários que são utilizados nos experimentos:

- **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.

Além disso, torna-se relevante definir o número de máquinas que estarão disponíveis para processamento e o número de tarefas que serão processadas. Para o experimento, foram processadas 1024 tarefas, obedecendo a um processo de chegada de *Poisson* (BUSSAB; MORETTIN, 2017). A simulação foi feita utilizando 16, 32 e 64 máquinas. Para cada cenário, a simulação foi executada 4000 vezes e o valor apresentado nesta seção corresponde à média e ao desvio padrão dessas simulações.

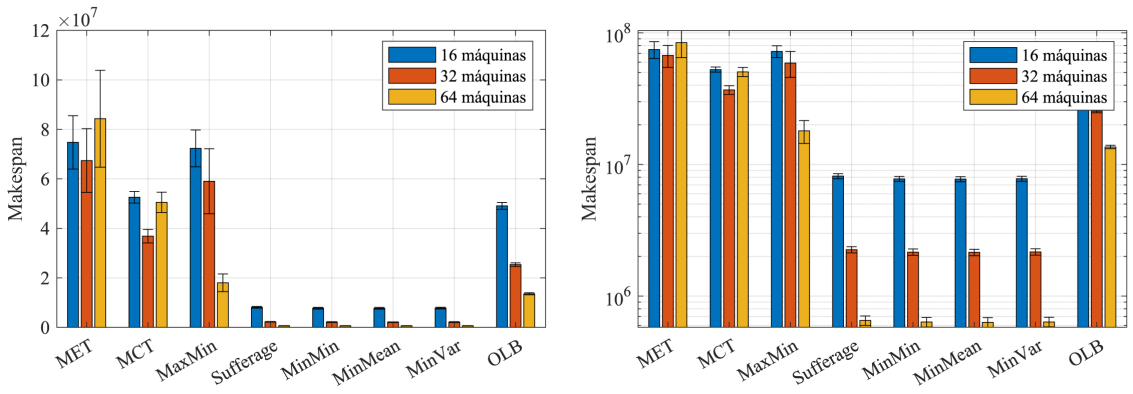
4.1.1 Cenário 1 - High-High

Nesse cenário, considera-se que as tarefas e as máquinas possuem alta heterogeneidade. Dessa maneira, o simulador utiliza um valor numérico alto para a geração dos tempos estimados de processamento para cada tarefa, fazendo com que o tamanho das tarefas e os tempos estimados de execução sejam grandes e com alta variação.

Na Figura 1, pode-se observar um destaque positivo para os algoritmos *Sufferage*, *MinMin*, *MinMean* e *MinVar*. Tais algoritmos obtiveram resultados relevantes para a métrica *makespan*, sendo consideravelmente grande a diferença para os demais algoritmos que não obtiveram bons resultados, ou seja, o *MET*, *MCT*, *MaxMin* e *OLB*. Ainda na Figura 1, compreende-se que o algoritmo *MET* possui o maior desvio padrão relacionado ao *makespan* médio das 4000 simulações, o que mostra, além do alto *makespan*, uma certa instabilidade de seus resultados.

Na Tabela 1, nota-se com mais clareza a diferença do *makespan* para cada algoritmo nesse cenário. Embora os quatro melhores algoritmos nesse quesito possuam resultados similares se comparados aos demais, observa-se que o algoritmo *MinMean* registrou o menor *makespan* para o Cenário 1.

Figura 1 – Makespan dos algoritmos para o Cenário 1 em escala padrão e logarítmica.



Fonte: Autoria própria (2022).

Tabela 1 – Makespan dos algoritmos para o Cenário 1.

Makespans - Cenário 1			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	74.779.309 ± 14,4%	67.428.254 ±19,08%	84.322.346 ±23,17%
MCT	52.620.616 ± 4,48%	36.839.735 ± 7,43%	50.578.711 ± 8,11%
MaxMin	72.347.675 ±10,32%	59.025.611 ±22,28%	18.032.800 ±19,75%
Sufferage	8.148.184 ± 4,39%	2.250.954 ± 5,40%	652.690 ± 8,11%
MinMin	7.777.209 ± 4,38%	2.159.827 ± 5,63%	635.586 ± 8,38%
MinMean	7.741.812 ± 4,40%	2.149.163 ± 5,58%	632.003 ± 8,39%
MinVar	7.798.592 ± 4,41%	2.166.791 ± 5,63%	636.291 ± 8,37%
OLB	49.100.83 ± 2,80%	25.389.003 ± 2,87%	13.580.320 ± 3,08%

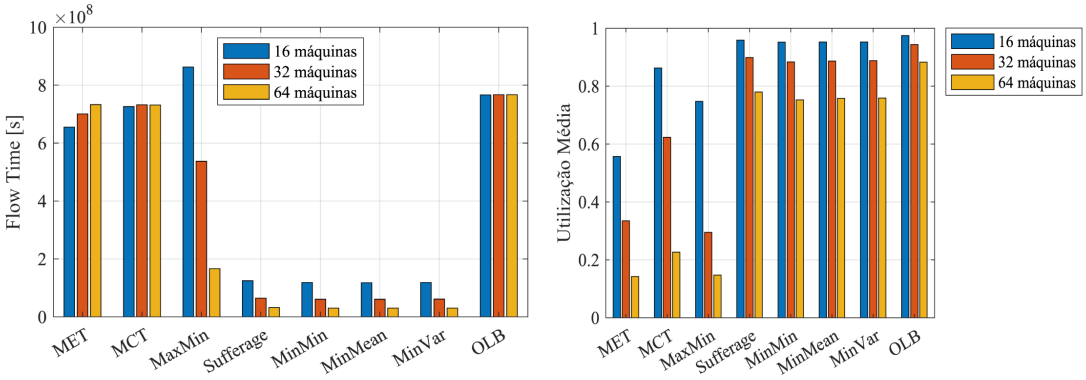
Fonte: Autoria própria (2022).

Na Figura 2, pode-se observar o *Flowtime* dos algoritmos no cenário 1. Os quatro melhores algoritmos em termos de *makespan* nesse cenário também são os melhores para a métrica *Flowtime*, mas é interessante destacar que nessa métrica utilizando 64 máquinas, o algoritmo *MinMin* conquistou melhor resultado quando comparado ao *MinMean*. Esse resultado prático é fiel à teoria, já que o MinMin designa as tarefas tarefas com menor tempo de processamento para a máquina em que a irá processar mais rapidamente.

Ainda na Figura 2, observa-se a *Utilização de Recurso* média dos algoritmos no cenário 1. O algoritmo *OLB* obteve os melhores resultados em todas as simulações, o que faz total sentido, já que tal algoritmo designa as tarefas para a primeira máquina que estiver disponível. Os destaques negativos para essa métrica ficam com os algoritmos *MET*, *MCT* e *MaxMin*, com valores chegando até a *14,3%*

de *Utilização de Recurso* média

Figura 2 – Flowtime e Utilização de Recurso dos algoritmos para o Cenário 1.



Fonte: Autoria própria (2022).

Tabela 2 – Flowtime dos algoritmos para o Cenário 1.

Flowtime - Cenário 1			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	655.468.344	700.838.627	733.134.240
MCT	726.551.360	732.761.599	731.969.319
MaxMin	863.090.535	537.694.382	166.822.956
Sufferage	125.023.245	64.734.962	32.459.409
MinMin	118.486.175	61.068.680	30.503.604
MinMean	118.118.516	60.950.437	30.521.256
MinVar	118.954.053	61.539.312	30.766.698
OLB	766.610.984	767.234.416	767.261.976

Fonte: Autoria própria (2022).

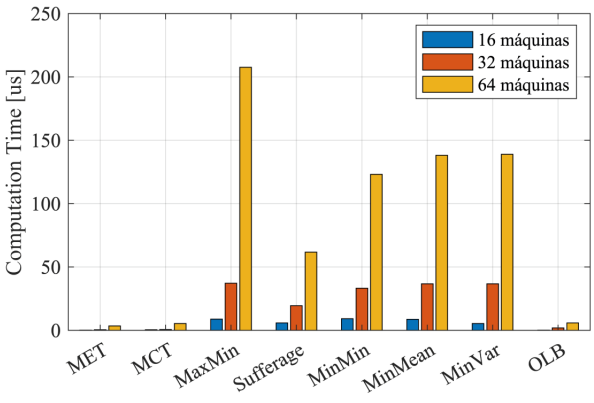
Tabela 3 – Utilização de Recurso dos algoritmos para o Cenário 1.

Utilização Média - Cenário 1			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	0,557	0,335	0,1438
MCT	0,863	0,624	0,2273
MaxMin	0,748	0,295	0,148
Sufferage	0,959	0,899	0,78
MinMin	0,952	0,884	0,753
MinMean	0,953	0,887	0,758
MinVar	0,953	0,888	0,759
OLB	0,975	0,944	0,883

Fonte: Autoria própria (2022).

Na Figura 3, é destacado o *Tempo de Comutação* dos algoritmos para o Cenário 1. Os valores estão representados em microssegundos e, nessa métrica, os algoritmos *MET*, *MCT* e *OLB* conquistaram melhores resultados. Um fato interessante é que os quatro melhores algoritmos em termos de *makespan* obtiveram altos valores de *Tempo de Computação* se comparados aos demais. O destaque negativo fica novamente com o *MaxMin*, que obteve o pior resultado, mesmo também não possuindo bons valores de *makespan*

Figura 3 – Tempo de Computação dos algoritmos para o Cenário 1.



Fonte: Autoria própria (2022).

Tabela 4 – Tempo de Computação dos algoritmos para o Cenário 1.

Tempo de Computação em microssegundos - Cenário 1			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	0,25	0,50	3,50
MCT	0,50	0,75	5,50
MaxMin	9,00	37,25	207,75
Sufferage	6,00	19,50	61,75
MinMin	9,25	33,25	123,25
MinMean	8,75	36,75	138,25
MinVar	5,50	36,75	139,00
OLB	0,25	2,00	6,00

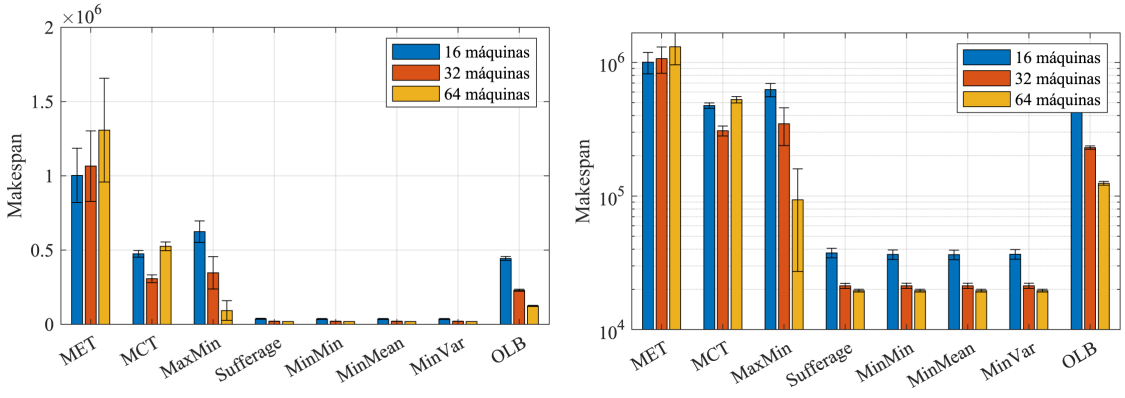
Fonte: Autoria própria (2022).

4.1.2 Cenário 2 - High-Low

Nesse cenário, entende-se que as tarefas possuem alta heterogeneidade e as máquinas possuem baixa heterogeneidade. Com isso, as tarefas continuam tendo complexidades bem discrepantes, mas desta vez, as máquinas são semelhantes.

Na Figura 4, percebe-se que os mesmos quatro algoritmos que foram os melhores no cenário anterior, também são os melhores do cenário 2 em termos de *makespan*. São eles o *Sufferage*, *MinMin*, *MinMean* e *MinVar*. Entretanto, pode-se notar uma melhoria considerável para os algoritmos *MCT*, *MaxMin* e *OLB*, quando comparados aos demais. O *desvio padrão* segue com as mesmas afirmações, sendo considerável a melhora do *MaxMin* e, por outro lado, o *MET* teve um *desvio padrão* ainda pior.

Figura 4 – Makespan dos algoritmos para o Cenário 2 em escala padrão e logarítmica



Fonte: Autoria própria (2022).

Na Tabela 5 são exibidos os *makespans* dos algoritmos no cenário 2. Nota-se que o algoritmo *MinMean* ainda obteve os melhores resultados dessa métrica. Entretanto, pode-se perceber que os outros três melhores algoritmos obtiveram resultados bem similares, sendo que na simulação com 32 máquinas, o *Sufferage* chegou a ser melhor. O destaque negativo continua com o algoritmo *MET*.

Tabela 5 – Makespan dos algoritmos para o Cenário 2.

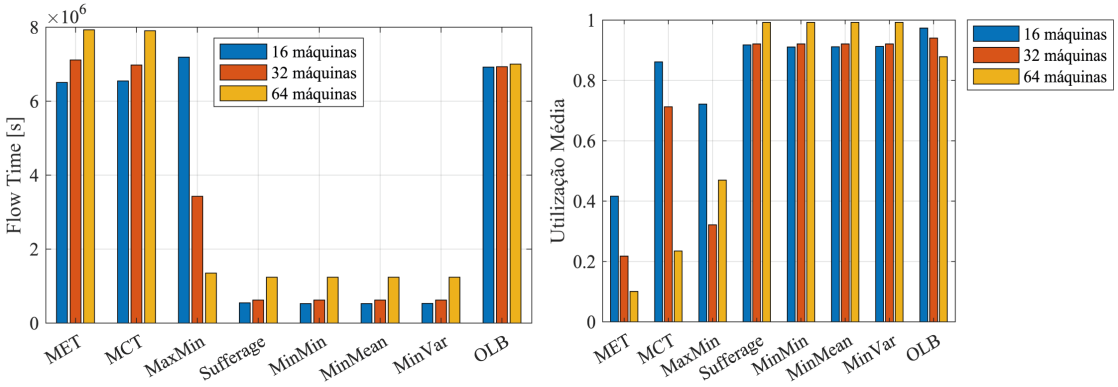
Makespans - Cenário 2			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	1003484 ±18,22%	1066013 ±22,30%	1308322 ±26,68%
MCT	475142 ± 4,70%	307737± 8,55%	526115 ± 5,54%
MaxMin	624552 ±11,49%	347307±31,30%	93535 ±70,91%
Sufferage	37520 ± 8,30%	21281 ± 4,46%	19588 ± 2,46%
MinMin	36474 ± 8,28%	21284 ± 4,47%	19588 ± 2,46%
MinMean	36362 ± 8,26%	21284 ± 4,47%	19588 ± 2,46%
MinVar	36658 ± 8,23%	21284 ± 4,47%	19588 ± 2,46%
OLB	444066 ± 2,92%	230155 ± 3,05%	124477 ± 3,17%

Fonte: Autoria própria (2022).

Na Figura 5, pode-se observar o *flowtime* dos algoritmos para o cenário 2. Também nesse cenário, os quatro algoritmos que registraram menor *flowtime* foram o *Sufferage*, *MinMin*, *MinMean* e *MinVar*, mas com discrepância menor em relação aos demais algoritmos. O algoritmo com menor *flowtime* registrado foi o *MinMean*.

Continuando na Figura 5, outra métrica presente é a *utilização de recurso* média. No cenário 2, como as máquinas possuem características parecidas, a escolha das máquinas tende a ser mais distribuída dentro dos algoritmos. Por isso, em geral, os algoritmos obtiveram resultados maiores de *utilização de recurso*. Um fato interessante é que os algoritmos *Sufferage*, *MinMin*, *MinMean* e *MinVar* registraram valores maiores do que o algoritmo *OLB* quando se tem 64 máquinas.

Figura 5 – Flowtime e Utilização de Recurso dos algoritmos para o Cenário 2.



Fonte: Autoria própria (2022).

Tabela 6 – Flowtime dos algoritmos para Cenário 2.

Flowtime - Cenário 2			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	6.506.446,88	7.115.075,16	7.929.267,289
MCT	6.548.787,562	6.981.108,822	7.904.398,738
MaxMin	7.192.686,633	3.430.984,182	1.352.771,7
Sufferage	550.970,597	626.642,828	1.245.223,706
MinMin	531.185,796	626.568,863	1.245.223,609
MinMean	530.640,84	626.568,482	1.245.223,583
MinVar	535.114,596	626.571,385	1.245.223,583
OLB	6.923.587,029	693.6475,257	7.002.882,688

Fonte: Autoria própria (2022).

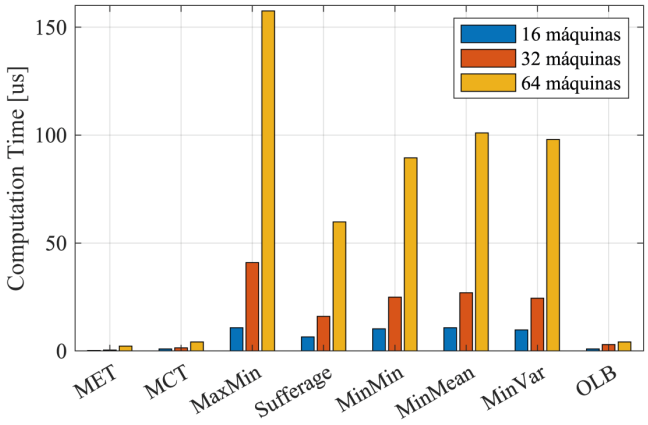
Tabela 7 – Utilização de Recurso dos algoritmos para o Cenário 2.

Utilização Média - Cenário 2			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	0,417	0,218	0,101
MCT	0,862	0,713	0,235
MaxMin	0,722	0,322	0,47
Sufferage	0,918	0,921	0,993
MinMin	0,911	0,921	0,993
MinMean	0,912	0,921	0,993
MinVar	0,913	0,921	0,993
OLB	0,974	0,941	0,879

Fonte: Autoria própria (2022).

Em termos de *tempo de computação* dos algoritmos para o cenário 2, foram registrados valores menores para os algoritmos em geral, como observado na Figura 6. Pode-se observar então que, quanto maior a heterogeneidade das máquinas, mais tempo os algoritmos levarão para escalonar as tarefas.

Figura 6 – Tempo de Computação dos algoritmos para o Cenário 2.



Fonte: Autoria própria (2022).

Tabela 8 – Tempo de Computação dos algoritmos para o Cenário 2.

Tempo de Computação em microssegundos - Cenário 2			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	0,25	0,50	2,25
MCT	1,00	1,50	4,25
MaxMin	10,75	41,00	157,50
Sufferage	6,50	16,00	59,75
MinMin	10,25	25,00	89,50
MinMean	10,75	27,00	101,00
MinVar	9,75	24,50	98,00
OLB	1,00	3,00	4,25

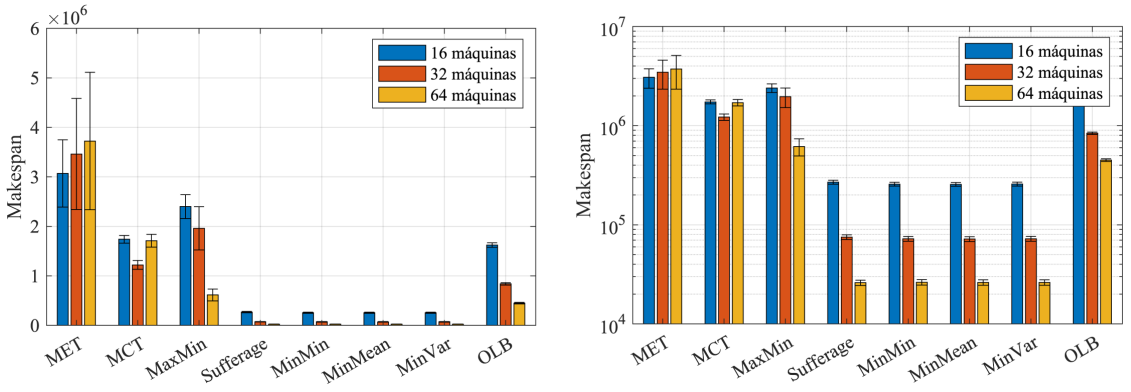
Fonte: Autoria própria (2022).

4.1.3 Cenário 3 - Low-High

No terceiro cenário, observa-se que as tarefas possuem baixa heterogeneidade e as máquinas possuem alta heterogeneidade. Com isso, as *máquinas* possuem grandes discrepâncias de performance, enquanto as *tarefas* possuem complexidades similares.

Na Figura 7, pode-se observar que a relação dos algoritmos de menor *makespan* continuam a mesma para o cenário 3. Os algoritmos *Sufferage*, *MinMin*, *MinMean* e *MinVar* registraram os melhores valores dessa métrica. Nota-se também que os algoritmos *MET* e *MCT* obtiveram melhores resultados de *desvio padrão* quando comparados ao cenário 2.

Figura 7 – Makespan dos algoritmos para o Cenário 3 em escala padrão e logarítmica



Fonte: Autoria própria (2022).

Os *makespans* dos algoritmos no cenário 3 são mostrados na Tabela 9. Observa-se que o algoritmo *MinMean* ainda obteve os melhores resultados para essa métrica. No entanto, os outros três melhores algoritmos também obtiveram resultados similares, sendo que para 64 máquinas, o *Sufferage* conquistou o menor *makespan*. O destaque negativo continua com o algoritmo *MET*.

Tabela 9 – Makespan dos algoritmos para o Cenário 3.

Makespans - Cenário 3			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	3071873 ±22,13%	3462391 ±32,42%	3725248 ±37,27%
MCT	1739622 ± 4,51%	1222678 ± 7,42%	1709533 ± 7,52%
MaxMin	2400663 ±10,09%	1960212 ±22,28%	615326 ±19,54%
Sufferage	269455 ± 4,47%	75188 ± 5,43%	26058 ± 6,23%
MinMin	257246 ± 4,45%	72242 ± 5,61%	26331 ± 6,39%
MinMean	256029 ± 4,49%	71919 ± 5,66%	26181 ± 6,40%
MinVar	257716 ± 4,47%	72365 ± 5,67%	26189 ± 6,38%
OLB	1622044 ± 2,82%	839243 ± 2,91%	450042 ± 3,09%

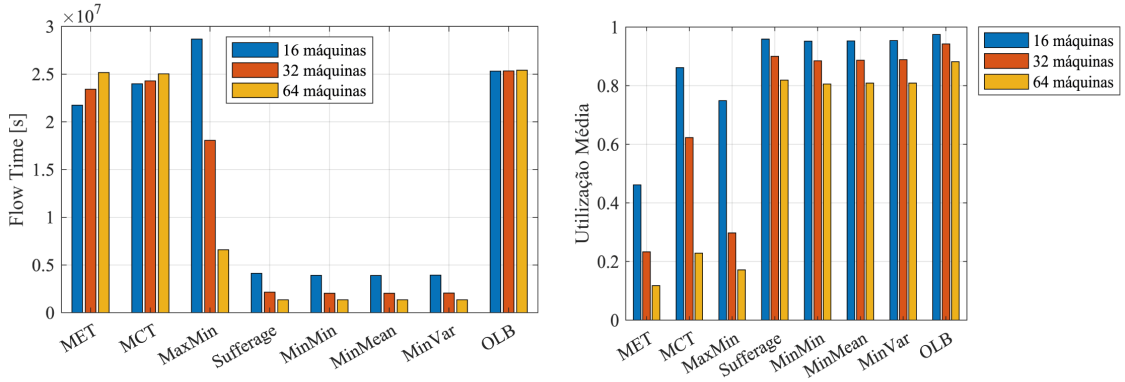
Fonte: Autoria própria (2022).

Na Figura 8, é exibido o *flowtime* de cada algoritmo para o cenário 3. Os resultados para esse cenário não tiveram tantas alterações, sendo que os quatro

algoritmos que registraram menor *flowtime* foram o *Sufferage*, *MinMin*, *MinMean* e *MinVar*. O algoritmo com menor *flowtime* registrado foi o *MinMean*.

Também na Figura 8, observa-se a *utilização de recurso* média. No cenário 3, o algoritmo *OLB* voltou a ter a maior *utilização de recurso* média em comparação aos demais.

Figura 8 – Flowtime e Utilização de Recurso dos algoritmos para o Cenário 3.



Fonte: Autoria própria (2022).

Tabela 10 – Flowtime dos algoritmos para Cenário 3.

Flowtime - Cenário 3			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	21.746.496,734	23.419.955,323	25.162.575,257
MCT	23.992.040,123	24.299.102,192	25.038.362,981
MaxMin	28.672.793,861	18.073.506,051	6.600.984,367
Sufferage	4.134.703,314	2.164.406,68	1.361.448,849
MinMin	3.920.227,939	2.044.074,9	1.354.644,352
MinMean	3.906.596,954	2.040.751,603	1.351.468,927
MinVar	3.934.618,528	2.056.760,697	1.352.565,934
OLB	25322558,862	25.346.969,288	25.408.947,853

Fonte: Autoria própria (2022).

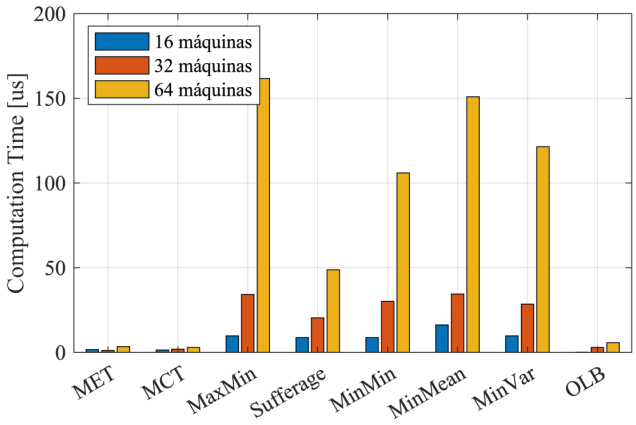
Tabela 11 – Utilização de Recurso dos algoritmos para o Cenário 3.

Utilização Média - Cenário 3			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	0,462	0,233	0,118
MCT	0,862	0,623	0,229
MaxMin	0,749	0,298	0,172
Sufferage	0,959	0,9	0,819
MinMin	0,952	0,885	0,806
MinMean	0,953	0,887	0,809
MinVar	0,954	0,889	0,809
OLB	0,975	0,943	0,882

Fonte: Autoria própria (2022).

Em termos de *tempo de computação* dos algoritmos para o cenário 3, é notável que, em geral, o tempo de computação para os algoritmos subiu, sendo que o *MaxMin* continua com os maiores tempos de computação em comparação aos demais algoritmos.

Figura 9 – Tempo de Computação dos algoritmos para o Cenário 3.



Fonte: Autoria própria (2022).

https://pt.overleaf.com/project/64387f4cbe0c5d39cc90cd20

Tempo de Computação em microssegundos - Cenário 3			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	1,75	1,25	3,50
MCT	1,50	2,00	3,00
MaxMin	9,75	34,25	161,75
Sufferage	8,75	20,50	48,75
MinMin	8,75	30,25	106,00
MinMean	16,25	34,50	151,00
MinVar	9,75	28,50	121,50
OLB	0,25	3,00	5,75

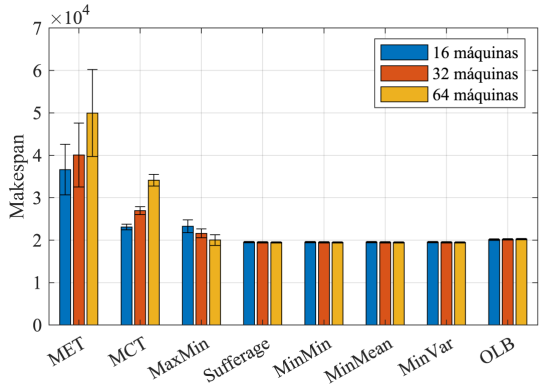
Fonte: Autoria própria (2022).

4.1.4 Cenário 4 - Low-Low

No quarto cenário, as tarefas e as máquinas possuem baixa heterogeneidade. Com isso, as *máquinas* são bem similares, assim como as *tarefas* possuem complexidades similares.

Na Figura 10, pode-se observar uma diferença considerável para o cenário 3. Os algoritmos em geral possuem resultados similares de *makespan*, com exceção do *MET* e do *MCT*. Quando as tarefas e máquinas possuem baixa heterogeneidade, as escolhas baseadas em performance das máquinas e tamanho das tarefas não são mais tão impactantes quanto antes. Por isso, o algoritmo *OLB* consegue ter uma performance bem similar aos outros algoritmos que nos outros cenários eram melhores. Ainda assim, pode-se considerar que os algoritmos *Sufferage*, *MinMin*, *MinMean* e *MinVar* possuíram juntos os melhores resultados de *makespan*.

Figura 10 – Makespan dos algoritmos para o Cenário 4.



Fonte: Autoria própria (2022).

Os *makespans* dos algoritmos no cenário 4 são mostrados na Tabela 12. Pode-se observar a semelhança dos resultados para os quatro melhores algoritmos nessa

métrica.

Tabela 12 – Makespan dos algoritmos para o Cenário 4.

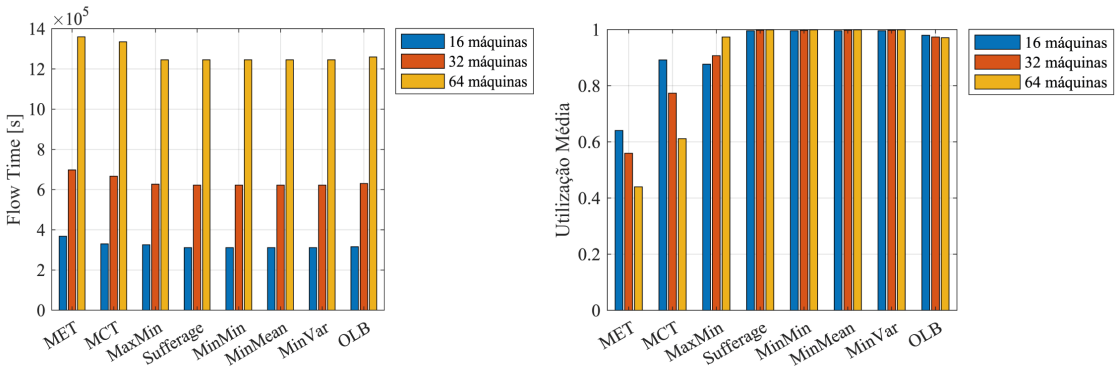
Makespans - Cenário 4			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	36632 ±16,30%	40082 ±18,78%	49966 ±20,54%
MCT	23108 ± 2,93%	26941 ± 3,47%	34121 ± 4,03%
MaxMin	23274 ± 6,47%	21616 ± 4,84%	20026 ± 6,22%
Sufferage	19544 ± 0,75%	19499 ± 0,75%	19456 ± 0,72%
MinMin	19544 ± 0,75%	19500 ± 0,75%	19456 ± 0,72%
MinMean	19544 ± 0,75%	19499 ± 0,75%	19456 ± 0,72%
MinVar	19544 ± 0,75%	19499 ± 0,75%	19456 ± 0,72%
OLB	20144 ± 0,95%	20200 ± 0,85%	20247 ± 0,76%

Fonte: Autoria própria (2022).

Na Figura 11, é exibido *flowtime* de cada algoritmo para o cenário 4. É interessante notar que todos os algoritmos possuem valor de *flowtime* bem parecidos. A diferença real fica na quantidade de máquinas disponíveis para processar, já que tanto as tarefas quanto as máquinas são similares.

Ainda na Figura 11, pode-se observar que a *utilização de recurso* média é bem alta para a maioria dos algoritmos.

Figura 11 – Flowtime e Utilização de Recurso dos algoritmos para o Cenário 4.



Fonte: Autoria própria (2022).

Tabela 13 – Flowtime dos algoritmos para Cenário 4.

Flowtime - Cenário 4			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	367.864,308	697.863,703	1.359.339,045
MCT	329.880,012	666.593,675	1.334.972,058
MaxMin	325.699,505	626.537,973	1.245.488,156
Sufferage	311.487,768	622.750,391	1.244.947,055
MinMin	311.477,667	622.749,306	1.244.947,053
MinMean	311.478,18	622.749,307	1.244.947,053
MinVar	311.478,716	622.749,315	1.244.947,053
OLB	316.078,442	630.064,534	1.259.208,167

Fonte: Autoria própria (2022).

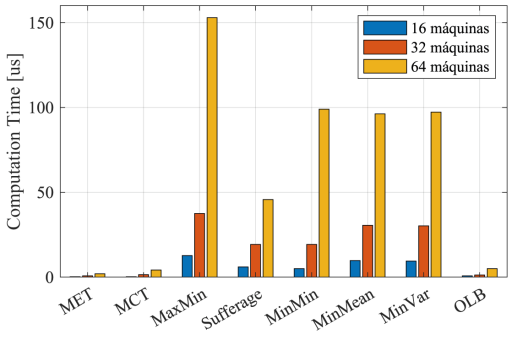
Tabela 14 – Utilização de Recurso dos algoritmos para o Cenário 4.

Utilização Média - Cenário 4			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	0,641	0,56	0,44
MCT	0,892	0,774	0,612
MaxMin	0,877	0,907	0,974
Sufferage	0,996	0,998	0,999
MinMin	0,996	0,997	0,999
MinMean	0,996	0,998	0,999
MinVar	0,996	0,998	0,999
OLB	0,98	0,974	0,971

Fonte: Autoria própria (2022).

Sobre o *tempo de computação* dos algoritmos para o cenário 4, pode-se perceber que o tempo de computação para os algoritmos foi inferior ao cenário 3, com o *MaxMin* sendo o algoritmo com maior tempo de computação para o cenário.

Figura 12 – Tempo de Computação dos algoritmos para o Cenário 4.



Fonte: Autoria própria (2022).

Tempo de Computação em microssegundos - Cenário 4			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	0,25	0,75	2,00
MCT	0,25	1,50	4,25
MaxMin	12,75	37,50	153,00
Sufferage	6,00	19,25	45,75
MinMin	5,00	19,25	99,00
MinMean	9,75	30,50	96,25
MinVar	9,50	30,25	97,25
OLB	0,75	1,25	5,00

Fonte: Autoria própria (2022).

5 Conclusões

Neste trabalho, foram analisados oito algoritmos para o problema de escalonamento de tarefas em grades computacionais. Vários cenários foram simulados a fim de entender o comportamento dos algoritmos em situações variadas. Além disso, cinco métricas foram utilizadas para a comparação dos resultados.

Em termos de *makespan*, quatro algoritmos foram destaques em quase todos os cenários: *Sufferage*, *MinMin*, *MinMean* e *MinVar*, sendo que o algoritmo com menor *makespan* médio foi o *MinMean*. É interessante destacar que, mesmo em cenários onde o *MinMean* não obteve o menor *makespan*, o seu resultado foi bem similar ao primeiro colocado, ou seja, ele demonstrou ótimos resultados para todos os cenários. Já o algoritmo *MET* obteve os maiores valores de *makespan*, mostrando assim que sua simplicidade não foi convertida em valores baixos de *makespan*.

Em relação ao *desvio padrão*, o algoritmo que obteve menores valores para os quatro cenários foi o *Sufferage*, seguido pelo *MinMin*, *Minvar* e *MinMean*. Por outro lado, os algoritmos *MET* e *MaxMin* obtiveram resultados altos de *desvio padrão*, mostrando que seus resultados são instáveis.

Para o *Flowtime*, o algoritmo que obteve melhor resultado em mais casos foi o *MinMean*, mas pode-se observar que em alguns cenários, o *MinMin* chegou a ser melhor nessa métrica. O algoritmo *MCT* obteve os piores resultados nessa métrica.

Outra métrica interessante é a *Utilização de Recursos*, pois ela apresenta um indicativo de como os recursos disponíveis estão sendo aproveitados. Neste quesito, o algoritmo *OLB* foi o que obteve maiores valores de *utilização de recurso* média, o que faz sentido, pois tal algoritmo designa uma tarefa para a primeira máquina que estiver disponível. Os algoritmos que obtiveram menor valor de *utilização de recurso*, ou seja, os que mais deixaram os recursos ociosos, foram o *MET* e o *MaxMin*, sendo que em alguns momentos, foram registrados valores próximos de *10%* de utilização média.

Sobre o *tempo de computação*, pode-se observar que os algoritmos que obtiveram melhores resultados nas métricas anteriores também são os que necessitam de maior *tempo de computação* para serem executados. Os algoritmos que obtiveram maiores valores nessa métrica foram o *MinMean*, *MinMin* e *MinVar*. Já o algoritmo que obteve melhores resultados nessa métrica foi o *MET*, o que também faz total sentido pela sua simplicidade.

Partindo do exposto, pode-se notar que alguns algoritmos conseguiram se destacar dos demais em geral, sendo eles o *Sufferage*, *MinMin*, *MinMean* e *MinVar*. Para as métricas de *makespan* e *flowtime*, o algoritmo *MinMean* conseguiu registrar os melhores resultados.

Referências

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*, Elsevier BV, v. 61, n. 6, p. 810–837, jun. 2001. Citado na página 11.

BUSSAB, W. de O.; MORETTIN, P. A. *Estatística Básica*. 9. ed. São Paulo: Saraiva Uni, 2017. 568 p. ISBN 978-8547220228. Citado na página 30.

SAHU, R.; CHATURVEDI, A. K. Many objective comparison of twelve grid scheduling heuristics. *International Journal of Computer Applications*, Foundation of Computer Science, v. 13, n. 6, p. 9–17, jan. 2011. Citado 2 vezes nas páginas 13 e 29.

VERMA, A. *Distributed Scheduling Simulator*. 2010. Disponível em: <https://github.com/dapurv5/distributedscheduling>. Acesso em: 13 ago. 2022. Citado na página 11.

XHAFA, F.; ABRAHAM, A. Computational models and heuristic methods for grid scheduling problems. *Future Generation Computer Systems*, Elsevier BV, v. 26, n. 4, p. 608–621, abr. 2010. Citado na página 11.