UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Marcos Victor de Aquino Barra

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

Uberlândia, Brasil

2022

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Marcos Victor de Aquino Barra

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

2022

Dedico este trabalho aos meus meus pais Sebastião e Geraldina,
à minha esposa Daniele e aos meus irmãos Janayna e Pedro,
pelo apoio à realização deste trabalho.

Agradecimentos

A Deus, pelo dom da vida, por sua graça abundante em minha vida e família, por cuidar e estar comigo em todos os momentos e por ter me dado condições de realizar este trabalho.

À minha esposa, Daniele Rosa Ribeiro Barra, pelo amor, companheirismo e incentivo em inúmeros momentos de minha vida.

Aos meus pais, Sebastião Lázaro Barra e Geraldina Lima de Aquino Barra, por me proporcionarem tanto amor e carinho, por acreditarem nos meus sonhos e me darem condições de ingressar nessa faculdade.

Aos meus irmãos, Janayna de Aquino Barra Braga e Pedro Henrique Aquino Barra, pelo incentivo, apoio, conselhos e companheirismo em todas as fases de minha vida.

Ao professor Paulo Henrique Ribeiro Gabriel, pelo incentivo e orientação durante a realização deste trabalho.

Aos meus amigos, Fabrício Fernandes Ziliotti e Guilherme Raimondi, pela parceria e amizade iniciadas neste curso que seguirão para toda a vida.

Resumo

O escalonamento de tarefas em grades computacionais é um tema amplamente discutido na comunidade científica. Visto que as tarefas processadas por um computador podem possuir características distintas ou similares de complexidade, assim como as próprias máquinas podem possuir recursos computacionais diferentes ou próximos, é necessário então que haja um escalonamento de tarefas a fim de que todas as tarefas possam ser executadas pelas máquinas disponíveis da melhor forma possível, de acordo com as métricas selecionadas. A partir do exposto, este trabalho de conclusão de curso tem por objetivo apresentar alguns algoritmos já conhecidos na comunidade científica. Esses algoritmos foram comparados utilizando técnicas, métricas e cenários diversos, para que haja entendimento das vantagens e desvantagens deles em cada um dos cenários descritos.

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

Abstract

Task scheduling in computational grids is widely discussed in the scientific community. Since the tasks processed by a computer may have different or similar characteristics of complexity, and the machines themselves may have different or close computing resources, it is then necessary to schedule tasks so that all tasks can be executed. By the available machines in the best possible way, according to the selected metrics. In this sense, this work presents some algorithms already known in the scientific community. We compare this algorithms using different techniques, metrics, and scenarios to understand the advantages and disadvantages of them in each of the described scenarios.

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

Sumário

Lista de algoritmos 8

Lista de ilustrações 9

Lista de tabelas 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 Visão Geral do Escalonamento de Tarefas 13
2.2 Métricas para Comparação dos Algoritmos 15

3 DESENVOLVIMENTO 17
3.1 Técnicas para Comparação dos Algoritmos 17
3.2 Modificações no Simulador 17
3.3 Algoritmos de Escalonamento de Tarefas 18
3.3.1 Minimum Execution Time (MET) 18
3.3.2 Minimum Completion Time (MCT) 18
3.3.3 Min-Min 19
3.3.4 Max-Min 20
3.3.5 XSuffrage 20
3.3.6 Opportunistic Load Balancing (OLB) 21
3.3.7 Min-Mean 23
3.3.8 Min-Var 23

4 EXPERIMENTOS 26
4.1 Parâmetros utilizados 26
4.1.1 Cenário 1 - High-High 27
4.1.2 Cenário 2 - High-Low 31
4.1.3 Cenário 3 - Low-High 34
4.1.4 Cenário 4 - Low-Low 38

5 CONCLUSÕES 42

REFERÊNCIAS 44

Lista de algoritmos

1 Pseudocódigo do algoritmo *Minimum Execution Time* 19
2 Pseudocódigo do algoritmo *Minimum Completion Time* 19
3 Pseudocódigo do algoritmo *Min-Min* 20
4 Pseudocódigo do algoritmo *Max-Min* 21
5 Pseudocódigo do algoritmo *XSuffrage* 22
6 Pseudocódigo do algoritmo *Opportunistic Load Balancing* 23
7 Pseudocódigo do algoritmo *Min-Mean* 24
8 Pseudocódigo do algoritmo *Min-Var* 25

Lista de ilustrações

Figura 1 – Ilustração do funcionamento de uma grade computacional. 14
Figura 2 – Makespan dos algoritmos para o Cenário 1 em escala padrão e logarítmica. 28
Figura 3 – Flowtime e Utilização de Recurso dos algoritmos para o Cenário 1. 29
Figura 4 – Tempo de Computação dos algoritmos para o Cenário 1. 30
Figura 5 – Makespan dos algoritmos para o Cenário 2 em escala padrão e logarítmica 31
Figura 6 – Flowtime e Utilização de Recurso dos algoritmos para o Cenário 2. 33
Figura 7 – Tempo de Computação dos algoritmos para o Cenário 2. 34
Figura 8 – Makespan dos algoritmos para o Cenário 3 em escala padrão e logarítmica 35
Figura 9 – Flowtime e Utilização de Recurso dos algoritmos para o Cenário 3. 36
Figura 10 – Tempo de Computação dos algoritmos para o Cenário 3. 37
Figura 11 – Makespan dos algoritmos para o Cenário 4. 38
Figura 12 – Flowtime e Utilização de Recurso dos algoritmos para o Cenário 4. 39
Figura 13 – Tempo de Computação dos algoritmos para o Cenário 4. 41

Lista de tabelas

Tabela 1 – Makespan dos algoritmos para o Cenário 1. 28
Tabela 2 – Flowtime dos algoritmos para o Cenário 1. 29
Tabela 3 – Utilização de Recurso dos algoritmos para o Cenário 1. 30
Tabela 4 – Tempo de Computação dos algoritmos para o Cenário 1. 31
Tabela 5 – Makespan dos algoritmos para o Cenário 2. 32
Tabela 6 – Flowtime dos algoritmos para Cenário 2. 33
Tabela 7 – Utilização de Recurso dos algoritmos para o Cenário 2. 33
Tabela 8 – Tempo de Computação dos algoritmos para o Cenário 2. 34
Tabela 9 – Makespan dos algoritmos para o Cenário 3. 35
Tabela 10 – Flowtime dos algoritmos para Cenário 3. 36
Tabela 11 – Utilização de Recurso dos algoritmos para o Cenário 3. 37
Tabela 12 – Makespan dos algoritmos para o Cenário 4. 39
Tabela 13 – Flowtime dos algoritmos para Cenário 4. 40
Tabela 14 – Utilização de Recurso dos algoritmos para o Cenário 4. 40

1 Introdução

1.1 Contextualização e Motivação

A computação em grade é um modelo computacional que tem por objetivo alcançar uma grande taxa de processamento por meio da divisão de tarefas entre máquinas (podendo ser localmente ou em nuvem). Dessa forma, as tarefas são executadas paralelamente, o que traz vários ganhos se comparado à execução em apenas um computador. Para isso, é necessário escolher qual processador irá executar determinada tarefa, a fim de tentar aproveitar ao máximo os recursos disponíveis (XHAFA; ABRAHAM, 2010).

O escalonamento de tarefas é um tema bem difundido atualmente, gerando diversas contribuições por parte comunidade científica (BRAUN *et al.*, 2001). Esta atividade gira em torno da decisão de qual processador (núcleo) será escolhido para processar determinada tarefa. Vale destacar que esta decisão não é trivial, visto que vários fatores podem tornar um escalonamento muito bom ou completamente inviável. Desse modo, vários algoritmos têm sido propostos visando apresentar certos tipos de benefício. Por outro lado, esses algoritmos apresentam lacunas e pontos de melhorias em outras métricas (BRAUN *et al.*, 2001; XHAFA; ABRAHAM, 2010).

Os algoritmos existentes analisaram as tarefas disponíveis para processamento e apontam qual núcleo do processador devem tratá-las. Alguns algoritmos são mais simples, baseando as escolhas no menor tempo de execução das tarefas. Alternativamente, alguns algoritmos mais elaborados analisam, por exemplo, qual seria o impacto causado caso uma tarefa não fosse designada a certo recurso. Por conseguinte, pode-se pontuar a extrema relevância de uma análise de benefícios e desvantagens desses algoritmos com vistas à otimização de suas execuções.

Dessa forma, analisam-se neste trabalho diversos algoritmos utilizando um simulador disponibilizado por Verma (2010). Neste simulador, alguns algoritmos já estão implementados e é possível obter alguns resultados provenientes da execução dos mesmos. No entanto, o simulador não possui alguns algoritmos que são convenientes às análises deste trabalho, além de trazer apenas a métrica de média dos resultados. Por isso, entende-se como relevante uma modificação do simulador para que, além de suportar mais algoritmos de interesse, possa também trazer mais métricas estatísticas, enriquecendo ainda mais os resultados e discussões desse estudo.

1.2 Objetivos

Este trabalho de conclusão de curso tem por objetivo geral apresentar o problema do escalonamento de tarefas, aplicando diversos algoritmos já existentes e os comparando através do simulador proposto por Verma (2010), para que seja possível perceber o cenário de uso ideal para cada um deles. Os objetivos específicos deste trabalho são:

- Apresentar e detalhar os principais algoritmos utilizados para o escalonamento de tarefas;
- Modificar o simulador proposto por Verma (2010) para que possa comparar mais algoritmos que são relevantes para o estudo;
- Modificar o simulador para que outra métrica seja implementada além do tempo de resposta;
- Trazer um estudo comparativo sobre os algoritmos, utilizando as modificações propostas para o simulador.

1.3 Organização do Trabalho

Este trabalho de conclusão de curso está dividido em 5 capítulos. Além do exposto neste capítulo introdutório, o trabalho está separado da seguinte forma: No Capítulo 2, serão apresentados conceitos relevantes sobre escalonamento de tarefas para entendimento do que será realizado nos próximos capítulos. No Capítulo 3, serão apresentados os algoritmos selecionados para o estudo e comparação, além da apresentação do simulador utilizado para os experimentos e as modificações nele executadas. No Capítulo 4, serão exibidos os resultados dos experimentos realizados utilizando o simulador proposto e os algoritmos estudados. O Capítulo 5 traz uma conclusão sobre os experimentos realizados e como os algoritmos podem ser comparados considerando todos os cenários.

2 Fundamentação Teórica

2.1 Visão Geral do Escalonamento de Tarefas

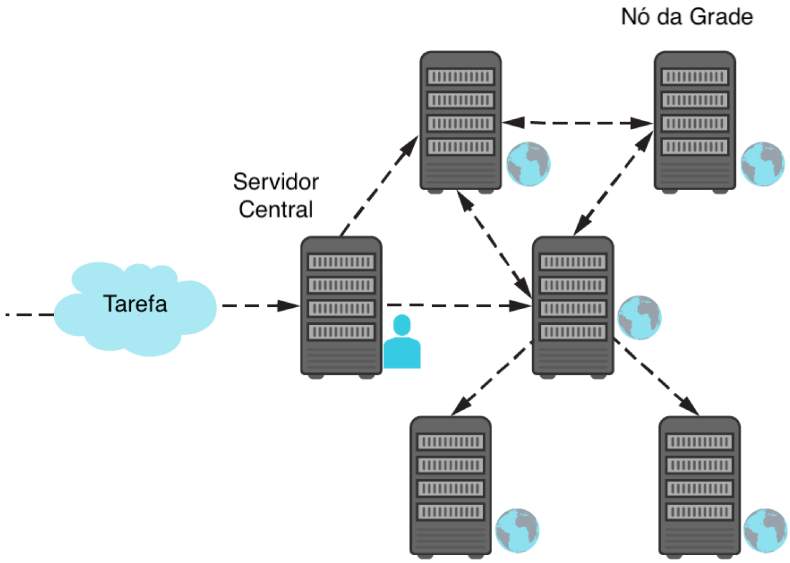
A computação em grade (do inglês, *Grid Computing*) começou a ser planejada devido à diminuição do custo dos componentes computacionais e à evolução das tecnologias de processamento e comunicação, já que está cada vez mais acessível o uso de vários recursos computacionais para que uma tarefa seja processada (STOCKINGER, 2007; DONG; AKL, 2006). Essa tecnologia fornece uma maneira de dividir uma aplicação computacional em muitas tarefas menores e distribuí-las para um supercomputador virtual que consiste em muitos computadores pequenos conectados em uma rede comum (OPENSTAX, 2018). Com isso, tem-se uma infraestrutura de hardware e software que agrupa e integra computadores e aplicativos de várias fontes, aproveitando a energia não utilizada em PCs e redes existentes.

Essa estrutura distribui recursos computacionais, mas mantém o controle central do processo. Esse servidor central divide uma tarefa em subtarefas, atribui essas tarefas a computadores na grade, combina os resultados e passa para a próxima tarefa até que a aplicação seja concluída (STOCKINGER, 2007; OPENSTAX, 2018). A Figura 1 ilustra o comportamento dessa infraestrutura.

Um dos grandes desafios da implantação de uma grade computacional é o escalonamento de tarefas. Cabe ao algoritmo de escalonamento responder à seguinte questão: “Para que seja processado determinado grupo de tarefas, e dado que alguns critérios de otimização são mais importantes do que outros, qual(is) processador(es) deve(m) ser utilizado(s)?” Com isso, torna-se importante apontar a definição de alguns termos que serão utilizados nesta monografia (DONG; AKL, 2006; XHAFA; ABRAHAM, 2010):

- Uma **tarefa** é uma unidade atômica a ser agendada pelo escalonador e designada a um recurso;
- As **propriedades** de uma tarefa são parâmetros computacionais, como requerimento de CPU ou Memória, prioridade, entre outros;
- Uma **aplicação** é um conjunto de tarefas que serão designadas a um conjunto de recursos;

Figura 1 – Ilustração do funcionamento de uma grade computacional.



Fonte: OpenStax (2018), sob a Creative Commons Attribution 4.0 International License.

- Um **recurso** é algo necessário para que uma operação seja processada (Processador, dispositivo de armazenamento, rede, etc.);
- Um **nó** é uma entidade composta por um ou mais recursos;
- Um **escalonamento de tarefas** é o mapeamento de quais recursos serão responsáveis por executarem determinadas tarefas.

Além desses termos, existem alguns conceitos que afetam o desempenho do escalonador. Entre esses, destacam-se (DONG; AKL, 2006):

- **Heterogeneidade:**, Atualmente os recursos são largamente distribuídos em vários domínios na Internet (tanto recursos computacionais e de armazenamento, quanto redes subjacentes), fazendo com que surjam diferentes possibilidades para que esses recursos sejam acessados.
- **Autonomia:** O cenário ideal para o escalonador de tarefas seria aquele em que fosse possível ter o controle e visão totais de todos os recursos disponíveis. Porém, não é o que acontece hoje em dia, visto que vários desses recursos são autônomos. Com isso, o escalonador deve respeitar diferentes políticas de acesso e configurações de prioridade, fazendo com que a estimativa de custo das tarefas em determinado recurso fique ainda mais difícil.

- **Dependência de Tarefas:** Outro fator que é crítico e que deve ser analisado com muita cautela pelo escalonador, fazendo com que se torne mais difícil ainda a estimativa, é a dependência de tarefas. Ou seja, algumas tarefas precisam que outras sejam executadas antes para que elas possam ser processadas. Com isso, o algoritmo deve levar em conta, além das métricas já propostas por ele, a dependência que as tarefas possuem entre elas.

No contexto desta monografia, lida-se com o problema de escalonamento de tarefa independentes (XHAFA; ABRAHAM, 2010). Esse problema é considerado importante dentro de ambientes distribuídos, aparecendo frequentemente em cenários compostos por múltiplos processadores autônomos executando o mesmo programa com diferentes entradas de dados (do inglês, *Single-program Multiple-data*, SPMD), comuns em processamento multimídia e em mineração de dados (NESMACHNOW; CANCELA; ALBA, 2010). Mesmo não lidando com a dependência entre tarefas, esse problema ainda é considerado desafiador. Trata-se, de fato, de um problema NP-difícil, pois é uma reformulação do *Multiprocessor Scheduling*, descrito por Garey e Johnson (1979).

Devido a essa complexidade, diversas heurísticas têm sido exploradas ao longo dos anos (SAHU; CHATURVEDI, 2011; VIR *et al.*, 2019). Nesse sentido, faz-se necessário o desenvolvimento de uma metodologia unificada para comparar tais heurísticas, de modo a observar os cenários em que cada uma se destaca. Para uma correta comparação dessas heurísticas, é necessário, primeiramente, definir quais critérios de otimização serão empregados. Na próxima seção, são descritos os principais objetivos encontrados na literatura.

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, pois ela determina qual o tempo em que a última *máquina* terminou de executar a última *tarefa*. Em outras palavras, o *makespan* aponta quanto tempo levou para que todas as tarefas fossem processadas. Seja M o número de processa-

dores (ou máquinas) e $C_{[i]}$ o tempo em que a i-ésima máquina concluiu toda a sua execução. Logo, tem-se a definição (XHAFA; ABRAHAM, 2010):

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

Utilização Média de Recurso. Essa métrica é interessante para se entender o quanto as máquinas estão sendo utilizadas. Para isso, verifica-se quanto tempo todas as máquinas ficaram ocupadas em relação ao seu tempo ocioso (BARDSIRI; HASHEMI, 2012). É importante que se tenha um algoritmo com boa utilização de recurso, ou seja, próximo de 1:

$$util = \frac{\sum_{i=0}^{M} C_i}{makespan.M}$$

Flowtime. Este valor representa a soma dos tempos de conclusão das máquinas. O *Flowtime* mede a qualidade de serviço do sistema de escalonamento (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

A fim de possibilitar uma comparação entre os algoritmos, torna-se necessário um simulador de escalonamento de tarefas que suporte o nível de complexidade apresentado. Verma (2010) apresenta um simulador que calcula o desempenho de vários algoritmos de escalonamento, possibilitando assim a comparação. Sua utilidade é notável, visto que implementa diversos algoritmos e permite ser estendido para outros. Os algoritmos que não estavam implementados originalmente pelo autor foram implementados neste trabalho, trazendo assim uma contribuição para o simulador original.

Como medida de comparação, o simulador fornece como resultado o *makespan* médio baseado em várias simulações para cada algoritmo. Com isto, é possível saber quanto tempo uma tarefa levará em média para ser processada em determinado algoritmo. Porém, como apresentado anteriormente, outras métricas além do *makespan* podem ser relevantes para o estudo. Por isso, as métricas discutidas anteriormente também foram implementadas neste trabalho, trazendo mais uma contribuição para o simulador original. Além disso, destaca-se que a média não é, por si só, uma medida muito confiável, pois ignora grandes variações nos resultados; por esse motivo, o simulador foi estendido para apresentar, também, o desvio-padrão (BUSSAB; MORETTIN, 2017) das medidas utilizadas.

3.2 Modificações no Simulador

Como foi apresentado anteriormente, algumas alterações no código fonte do simulador proposto por Verma (2010) foram necessárias. O simulador originalmente foi desenvolvido utilizando a versão 8 da linguagem de programação Java. Trata-se de uma versão antiga da linguagem, com diversas bibliotecas desatualizadas e "depreciadas", ou seja, cuja utilização não é mais recomendada. Por esse motivo, a primeira grande modificação foi a alteração da versão do Java utilizada no projeto para a versão 11 da linguagem, possibilitando a utilização de várias bibliotecas e implementações de estruturas de dados novas para facilitar o desenvolvimento das alterações (BORGES; ERICKSON, 2022).

Logo em seguida, outras métricas relevantes foram implementadas. Assim,

flowtime, *utilização de recurso* e *tempo de computação*, que ainda não estavam presentes no simulador original, foram incrementadas ao código fonte. Também foi incorporado o *desvio padrão*, métrica estatística complementar à média. Por último, para entender o funcionamento do simulador e como estendê-lo, foi acrescentado o algoritmo *Opportunistic Load Balancing* (OLB), seguindo os mesmos padrões e práticas que já estavam sendo adotadas por Verma (2010). Mudanças estruturais em classes específicas também foram necessárias para atender aos novos requisitos, assim como melhorias de código, com o objetivo de seguir boas práticas de programação e código limpo.

3.3 Algoritmos de Escalonamento de Tarefas

Para este trabalho, são analisados algoritmos de escalonamento de tarefas independentes. O tema já recebeu várias discussões e contribuições da comunidade. Assim sendo, analisam-se alguns algoritmos para tarefas independentes, vários deles propostos por Braun *et al.* (2001). Também são apresentados os seus respectivos pseudo-códigos. Para isso, duas variáveis básicas são utilizadas para facilitar o entendimento:

- **mat[maquina]:** Este é o tempo estimado em que a máquina irá finalizar o processamento das tarefas que estão designadas para ela.
- **etc[tarefa][maquina]:** Este é o tempo estimado que a máquina irá levar para processar esta tarefa.

3.3.1 Minimum Execution Time (MET)

O Algoritmo MET designa uma tarefa para o recurso que terá o melhor tempo de execução estimado para essa tarefa, não importando se o recurso está disponível ou não no momento. A motivação deste algoritmo é dar para cada tarefa a melhor máquina, porém isso pode gerar um desbalanceamento nas máquinas e fazer com que uma máquina fique sobrecarregada;

3.3.2 Minimum Completion Time (MCT)

O Algoritmo MCT designa cada tarefa para o recurso com o tempo mínimo estimado de conclusão para a tarefa. Ou seja, não necessariamente a tarefa será designada à máquina que processaria em menos tempo, pois essa máquina pode

Algoritmo 1: Pseudocódigo do algoritmo *Minimum Execution Time*

```
   Dados: Lista de Tarefas
 1 tempoMinimo = Integer.MAX_VALUE;
 2 maquinaEscolhida = 0;
 3 para tarefa em tarefas faça
 4 |   para maquina em máquinas faça
 5 |   |   se etc[tarefa][maquina] < tempoMinimo então
 6 |   |   |   tempoMinimo = tempoEstimado[tarefa][maquina];
 7 |   |   |   maquinaEscolhida = maquina;
 8 |   |   fim
 9 |   fim
10 |   Designar tarefa para a maquinaEscolhida;
11 fim
```

estar ocupada no momento, fazendo com que talvez seja mais viável uma máquina mais lenta, porém disponível, executar essa tarefa.

Algoritmo 2: Pseudocódigo do algoritmo *Minimum Completion Time*

```
   Dados: Lista de Tarefas
 1 tempoMinimo = Integer.MAX_VALUE;
 2 maquinaEscolhida = 0;
 3 para tarefa em tarefas faça
 4 |   para maquina em máquinas faça
 5 |   |   se (etc[tarefa][maquina] + mat[maquina]) < tempoMinimo então
 6 |   |   |   tempoMinimo = etc[tarefa][maquina] + mat[maquina];
 7 |   |   |   maquinaEscolhida = maquina;
 8 |   |   fim
 9 |   fim
10 |   Designar tarefa para a maquinaEscolhida;
11 fim
```

3.3.3 Min-Min

Este algoritmo começa com um conjunto N de tarefas não designadas. Após isso, é encontrado o conjunto M de tempos mínimos de conclusão (similar ao MCT) para cada tarefa em N. Depois, a tarefa em N que possui o menor tempo em M é designada a máquina correspondente (por isso se chama Min-Min). Depois disso, a tarefa é retirada do conjunto e o processo é repetido até que todas as tarefas sejam processadas. É importante destacar que o Min-Min é um dos algoritmos de escalonamento mais antigos e estudados dessa área, tendo sido proposto na década de 1970 por Ibarra e Kim (1977).

Algoritmo 3: Pseudocódigo do algoritmo *Min-Min*

```
   Dados: Lista de Tarefas
 1 boolean[] foiRemovida = boolean[numero_tarefas];
 2 repita
 3 |  tempoMinimo = Integer.MAX_VALUE;
 4 |  maquinaEscolhida = -1;
 5 |  tarefaEscolhida = -1;
 6 |  para tarefa em tarefas faça
 7 |  |  para maquina em máquinas faça
 8 |  |  |  se (etc[tarefa][maquina] + mat[maquina]) < tempoMinimo
   |  |  |   então
 9 |  |  |  |  tempoMinimo = etc[tarefa][maquina] + mat[maquina];
10 |  |  |  |  maquinaEscolhida = maquina;
11 |  |  |  |  tarefaEscolhida = -1;
12 |  |  |  fim
13 |  |  fim
14 |  fim
15 |  Designar tarefaEscolhida para a maquinaEscolhida;
16 |  foiRemovida[tarefaEscolhida] = true;
17 até todas as tarefas forem designadas;
```

3.3.4 Max-Min

Também proposto por Ibarra e Kim (1977), este algoritmo é bem parecido com o Min-Min. Ele também inicia o conjunto N de tarefas não designadas e o conjunto M de tempos mínimos de conclusão para cada tarefa em N. Porém, ao invés de selecionar a tarefa em N que possui o menor tempo em M, ele seleciona a tarefa que possui o maior tempo. A ideia desse algoritmo é compensar a penalidade que seria gerada, fazendo com que a tarefa "mais lenta"possa ser executada pela máquina "mais rápida".

3.3.5 XSuffrage

Este algoritmo tem a intenção de designar primeiro uma tarefa a um recurso sabendo que, caso não seja designado, essa tarefa seria a mais penalizada dentre as outras (CASANOVA *et al.*, 2000). Para fazer isso, é calculado um valor Suffrage, que é igual à diferença entre o melhor MCT de uma tarefa para o segundo melhor MCT dessa tarefa, ou seja, qual seria a penalidade caso essa tarefa não fosse designada ao melhor MCT. Com isso, o algoritmo seleciona a tarefa que possui o valor Suffrage mais alto.

Algoritmo 4: Pseudocódigo do algoritmo *Max-Min*

```
Dados: Lista de Tarefas
boolean[] foiRemovida = boolean[numero_tarefas];
int[] minCompletionTime = int[numero_tarefas];
int[] minCompletionMachine = int[numero_tarefas];
repita
    tarefaEscolhida = -1;
    para tarefa em tarefas faça
        tempoMinimo = Integer.MAX_VALUE;
        maquinaEscolhida = -1;
        para maquina em máquinas faça
            se (etc[tarefa][maquina] + mat[maquina]) < tempoMinimo
             então
                tempoMinimo = etc[tarefa][maquina] + mat[maquina];
                maquinaEscolhida = maquina;
            fim
        fim
        minCompletionTime[tarefa] = tempoMinimo;
        minCompletionMachine[tarefa] = maquinaEscolhida;
    fim
    maxMinCompletionTime = Integer.MIN_VALUE;
    para tarefa em tarefas faça
        se maxMinCompletionTime < minCompletionTime[tarefa]
         então
            maxMinCompletionTime = minCompletionTime[tarefa];
            tarefaEscolhida = tarefa;
        fim
    fim
    Designar tarefaEscolhida para a maquinaEscolhida;
    foiRemovida[tarefaEscolhida] = true;
até todas as tarefas forem designadas;
```

3.3.6 Opportunistic Load Balancing (OLB)

Este algoritmo tem a intenção de maximizar o tempo em que cada máquina está ocupada. Para isso, para cada tarefa, o algoritmo a designa para a próxima máquina que está estimada para estar disponível, independentemente do tempo de execução desta tarefa na máquina. Uma vantagem do OLB é a sua simplicidade, visto que fazer a designação dessas tarefas é barato. Entretanto, por não considerar os tempos de execução de cada tarefa, a utilização desse algoritmo pode gerar resultados ruins de *makespan* (BARDSIRI; HASHEMI, 2012).

Algoritmo 5: Pseudocódigo do algoritmo *XSuffrage*

```
   Dados: Lista de Tarefas
 1 boolean[] foiRemovida = boolean[numero_tarefas];
 2 repita
 3     sufferageMaximo = Integer.MIN_VALUE;
 4     tarefaEscolhida = -1;
 5     maquinaEscolhida = -1;
 6     para tarefa em tarefas faça
 7         tempoMinimo1 = Integer.MAX_VALUE;
 8         maquina1 = -1;
 9         tempoMinimo2 = Integer.MAX_VALUE;
10         maquina2 = -1;
11         para maquina em máquinas faça
12             se (etc[tarefa][maquina] + mat[maquina]) < tempoMinimo1
                então
13                 tempoMinimo1 = etc[tarefa][maquina] + mat[maquina];
14                 maquina1 = maquina;
15             fim
16         fim
17         para maquina em máquinas faça
18             se maquina != maquina1 E (etc[tarefa][maquina] +
                mat[maquina]) < tempoMinimo2 então
19                 tempoMinimo2 = etc[tarefa][maquina] + mat[maquina];
20                 maquina2 = maquina;
21             fim
22         fim
23         sufferage = tempoMinimo1 - tempoMinimo2;
24         se sufferage > sufferageMaximo então
25             sufferageMaximo = sufferage;
26             tarefaEscolhida = tarefa;
27             maquinaEscolhida = maquina1;
28         fim
29     fim
30     Designar tarefaEscolhida para a maquinaEscolhida;
31     foiRemovida[tarefaEscolhida] = true;
32 até todas as tarefas serem designadas;
```

Algoritmo 6: Pseudocódigo do algoritmo *Opportunistic Load Balancing*

```
   Dados: Lista de Tarefas
 1 para tarefa em tarefas faça
 2 |   matMinimo = Integer.MAX_VALUE;
 3 |   maquinaEscolhida = -1;
 4 |   para maquina em máquinas faça
 5 |   |   se mat[maquina] < matMinimo então
 6 |   |   |   matMinimo = mat[maquina];
 7 |   |   |   maquinaEscolhida = maquina;
 8 |   |   fim
 9 |   fim
10 |   Designar tarefa para a maquinaEscolhida;
11 fim
```

3.3.7 Min-Mean

Este algoritmo, proposto por Kamalam e Bhaskaran (2010), possui duas fases. Em um primeiro momento, ele utiliza o Min-Min para realizar um primeiro mapeamento com as tarefas que possuem menor tempo estimado de conclusão. Após isso, é calculada a média $\overline{m}$ dos tempos estimados de conclusão das máquinas do sistema. Então, para cada máquina, verifica-se se o tempo estimado de conclusão das tarefas dessa máquina é maior do que a média. Caso $C_{[i]} > \overline{m}$, então inicia-se o processo de remanejamento das tarefas desta máquina. Esse processo consistem em pegar todas as tarefas já designadas a esta máquina e transferi-las para outras máquinas, de forma que a nova máquina selecionada pra uma tarefa será aquela que possuir o menor novo tempo estimado de conclusão.

3.3.8 Min-Var

Este algoritmo é similar ao Min-Mean. Ele também começa utilizando o Min-Min para realizar um primeiro mapeamento das tarefas. Após isso, é calculada a variância $\hat{\sigma}^2$ dos tempos estimados de conclusão das máquinas do sistema. Com isso, o processo continua conforme descrito no algoritmo Min-Mean, verificando, porém, se $C_{[i]} > \hat{\sigma}^2$.

Algoritmo 7: Pseudocódigo do algoritmo *Min-Mean*

```
Dados: Lista de Tarefas
boolean[] foiRemovida = boolean[numero_tarefas];
repita
    tempoMinimo = Integer.MAX_VALUE;
    maquinaEscolhida = -1;
    tarefaEscolhida = -1;
    para tarefa em tarefas faça
        para maquina em máquinas faça
            se (etc[tarefa][maquina] + mat[maquina]) < tempoMinimo
             então
                tempoMinimo = etc[tarefa][maquina] + mat[maquina];
                maquinaEscolhida = maquina;
                tarefaEscolhida = -1;
            fim
        fim
    fim
    Designar tarefaEscolhida para a maquinaEscolhida;
    foiRemovida[tarefaEscolhida] = true;
até todas as tarefas forem designadas;
mediaMaquinas = Calcular média de tempos de conclusão estimadas
 (mat) de todas as máquinas;
para maquina em maquinas faça
    se mat[maquina] <= mediaMaquinas então
        Pula para a Próxima máquina;
    fim
    para tarefa em Tarefas designadas a essa máquina faça
        maquinaEscolhida = -1;
        delta = 0;
        para maquinaCandidata em maquinas faça
            se mat[maquinaCandidata] >= mediaMaquinas então
                Pula para a Próxima máquina candidata;
            fim
            novoTempoEstimado = mat[maquinaCandidata] +
             etc[tarefa][maquinaCandidata];
            se novoTempoEstimado < mediaMaquinas então
                se (mediaMaquinas - novoTempoEstimado) > delta então
                    delta = mediaMaquinas - novoTempoEstimado;
                    maquinaEscolhida = maquinaCandidata;
                fim
            fim
        fim
        Designar tarefaEscolhida para a maquinaEscolhida;
    fim
fim
```

Algoritmo 8: Pseudocódigo do algoritmo *Min-Var*

```
   Dados: Lista de Tarefas
 1 boolean[] foiRemovida = boolean[numero_tarefas];
 2 repita
 3     tempoMinimo = Integer.MAX_VALUE;
 4     maquinaEscolhida = -1;
 5     tarefaEscolhida = -1;
 6     para tarefa em tarefas faça
 7         para maquina em máquinas faça
 8             se (etc[tarefa][maquina] + mat[maquina]) < tempoMinimo
                então
 9                 tempoMinimo = etc[tarefa][maquina] + mat[maquina];
10                 maquinaEscolhida = maquina;
11                 tarefaEscolhida = -1;
12             fim
13         fim
14     fim
15     Designar tarefaEscolhida para a maquinaEscolhida;
16     foiRemovida[tarefaEscolhida] = true;
17 até todas as tarefas forem designadas;
18 mediaMaquinas = Calcular média de tempos de conclusão estimadas
     (mat) de todas as máquinas;
19 para maquina em maquinas faça
20     se mat[maquina] <= mediaMaquinas então
21         Pula para a Próxima máquina;
22     fim
23     para tarefa em Tarefas designadas a essa máquina faça
24         maquinaEscolhida = -1;
25         delta = 0;
26         para maquinaCandidata em maquinas faça
27             se mat[maquinaCandidata] >= mediaMaquinas então
28                 Pula para a Próxima máquina candidata;
29             fim
30             novoTempoEstimado = mat[maquinaCandidata] +
                 etc[tarefa][maquinaCandidata];
31             se novoTempoEstimado < mediaMaquinas então
32                 se (mediaMaquinas - novoTempoEstimado) > delta então
33                     delta = mediaMaquinas - novoTempoEstimado;
34                     maquinaEscolhida = maquinaCandidata;
35                 fim
36             fim
37         fim
38         Designar tarefaEscolhida para a maquinaEscolhida;
39     fim
40 fim
```

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.

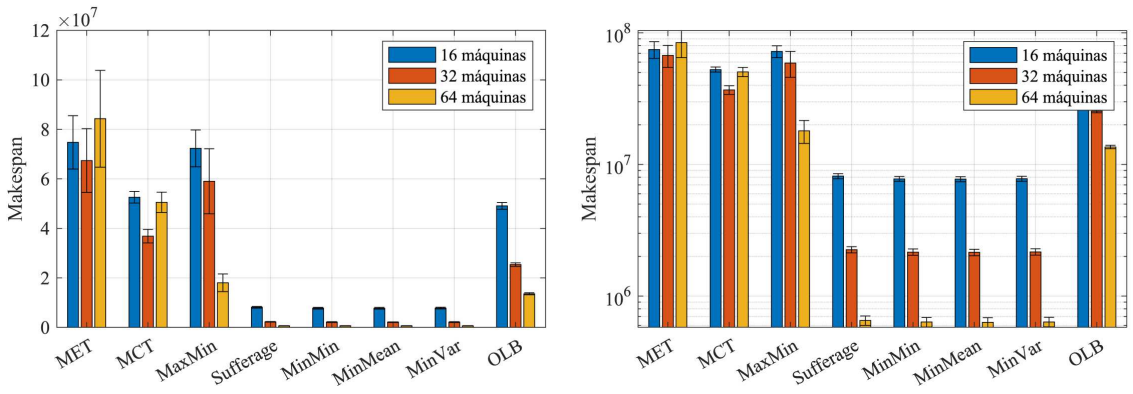
4.1.1 Cenário 1 - High-High

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 2, pode-se observar um destaque positivo para os algoritmos *XSufferage*, *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 2, 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 2 – 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%
XSufferage	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%

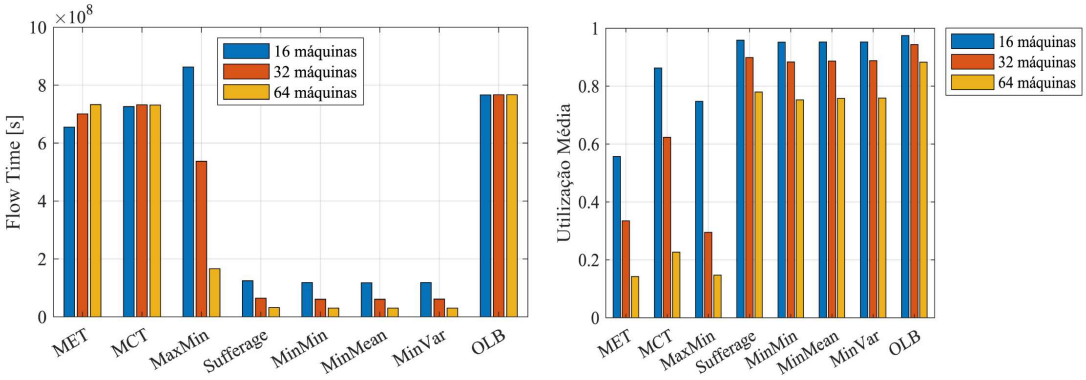
Fonte: Autoria própria (2022).

Na Figura 3, 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 3, 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 3 – 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
XSufferage	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).

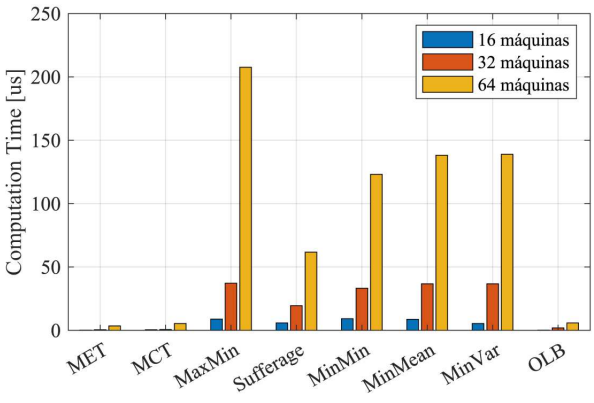
Tabela 3 – Utilização de Recurso dos algoritmos para o Cenário 1.

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
XSufferage	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 4, 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 4 – 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
XSufferage	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

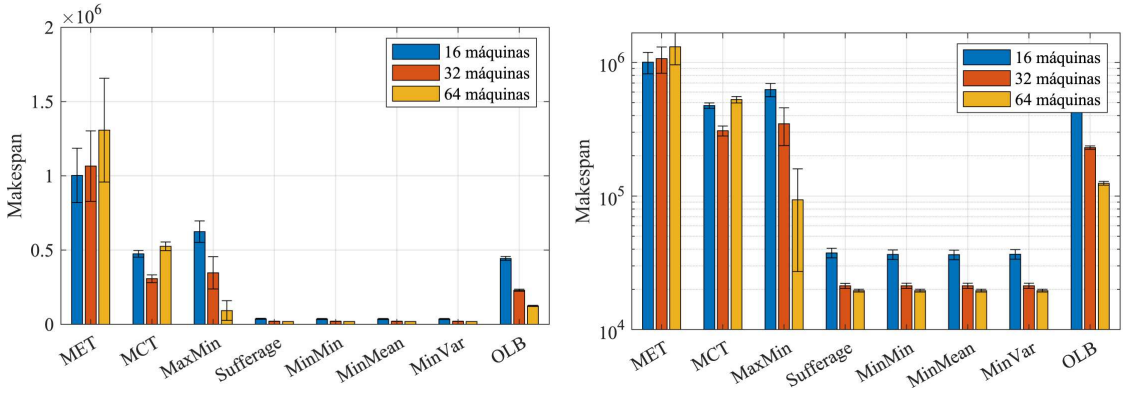
Fonte: Autoria própria (2022).

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 5, 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 *XSufferage*, *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 5 – 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 *XSufferage* chegou a ser melhor. O destaque negativo continua com o algoritmo *MET*.

Tabela 5 – Makespan dos algoritmos para o Cenário 2.

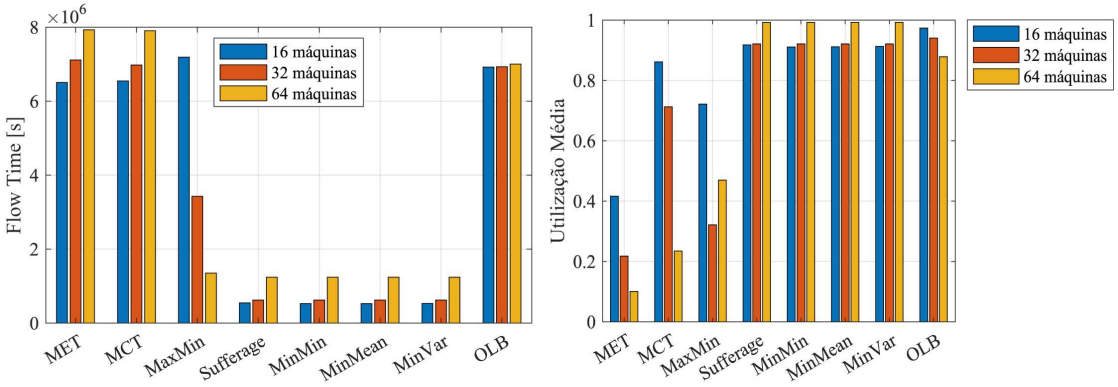
Makespans - Cenário 2			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	1003484 ±18,22%	1066013 ±22,30%	1308322 ±26,68%
MCT	475142 ± 4,70%	307737± 8,55%	526115 ± 5,54%
MaxMin	624552 ±11,49%	347307±31,30%	93535 ±70,91%
XSufferage	37520 ± 8,30%	21281 ± 4,46%	19588 ± 2,46%
MinMin	36474 ± 8,28%	21284 ± 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 6, pode-se observar o *flowtime* dos algoritmos para o cenário 2. Também nesse cenário, os quatro algoritmos que registraram menor *flowtime* foram o *XSufferage*, *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 6, 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 *XSufferage*, *MinMin*, *MinMean* e *MinVar* registraram valores maiores do que o algoritmo *OLB* quando se tem 64 máquinas.

Figura 6 – 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
XSufferage	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).

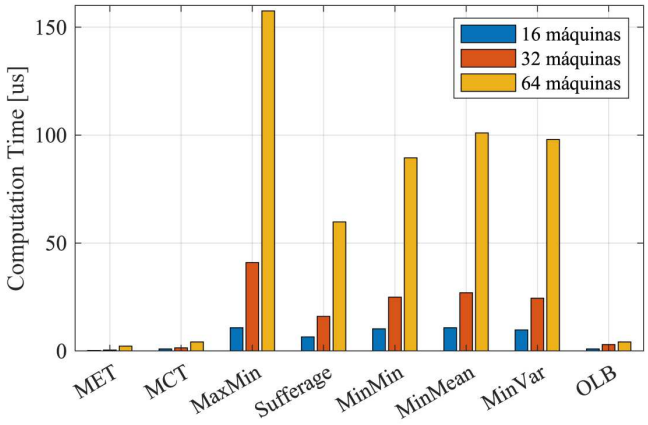
Tabela 7 – Utilização de Recurso dos algoritmos para o Cenário 2.

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
XSufferage	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 7. Pode-se observar então que, quanto maior a heterogeneidade das máquinas, mais tempo os algoritmos levarão para escalonar as tarefas.

Figura 7 – 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
XSufferage	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

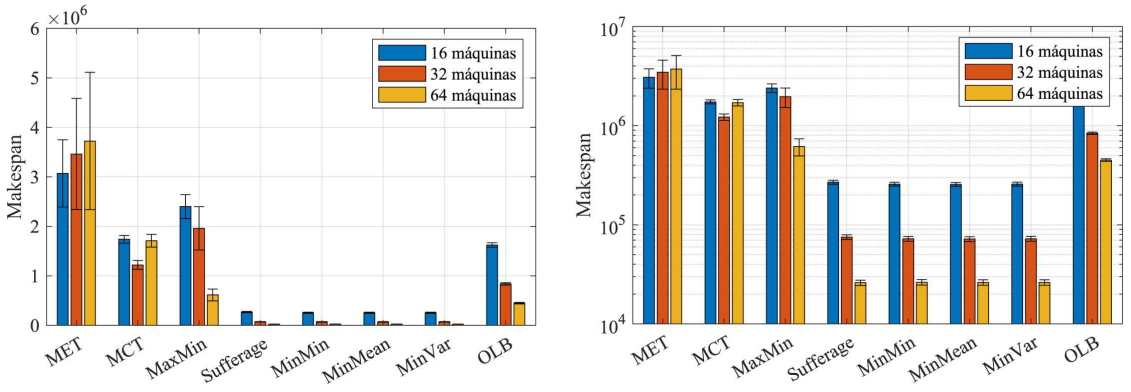
Fonte: Autoria própria (2022).

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 8, pode-se observar que a relação dos algoritmos de menor *makespan* continuam a mesma para o cenário 3. Os algoritmos *XSufferage*, *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 8 – 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 *XSufferage* 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%
XSufferage	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%

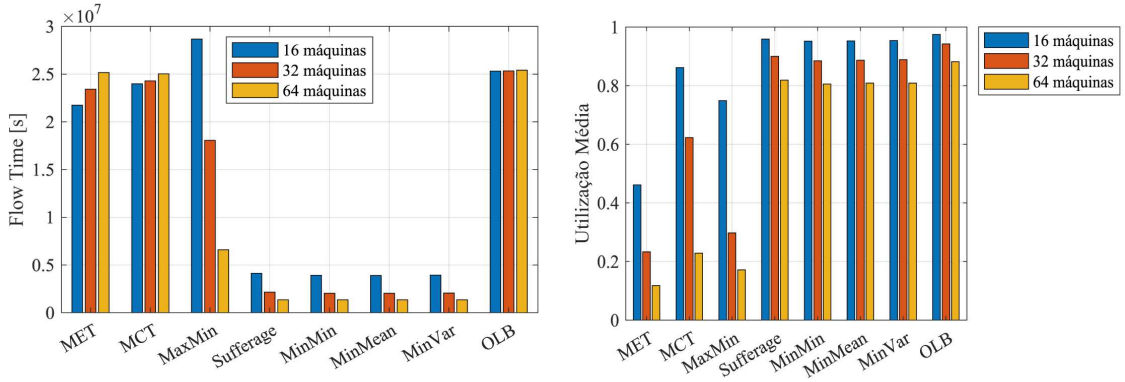
Fonte: Autoria própria (2022).

Na Figura 9, 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 *XSufferage*, *MinMin*, *MinMean* e *MinVar*. O algoritmo com menor *flowtime* registrado foi o *MinMean*.

Também na Figura 9, 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 9 – 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
XSufferage	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).

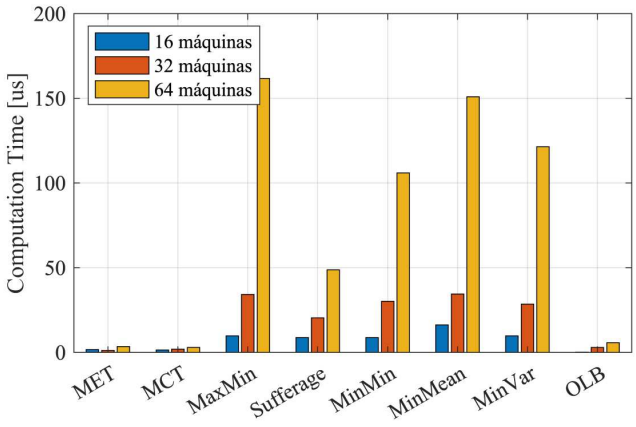
Tabela 11 – Utilização de Recurso dos algoritmos para o Cenário 3.

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
XSufferage	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 10 – Tempo de Computação dos algoritmos para o Cenário 3.



Fonte: Autoria própria (2022).

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
XSufferage	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

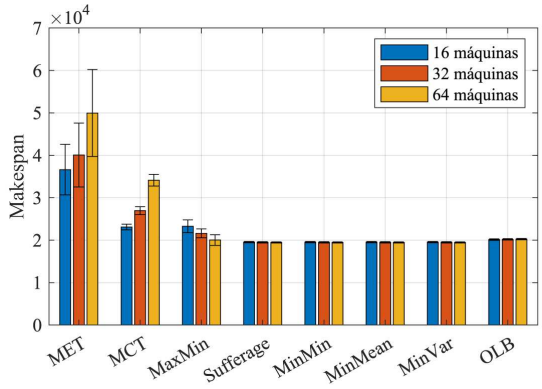
Fonte: Autoria própria (2022).

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 11, 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 *XSufferage*, *MinMin*, *MinMean* e *MinVar* possuíram juntos os melhores resultados de *makespan*.

Figura 11 – 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.

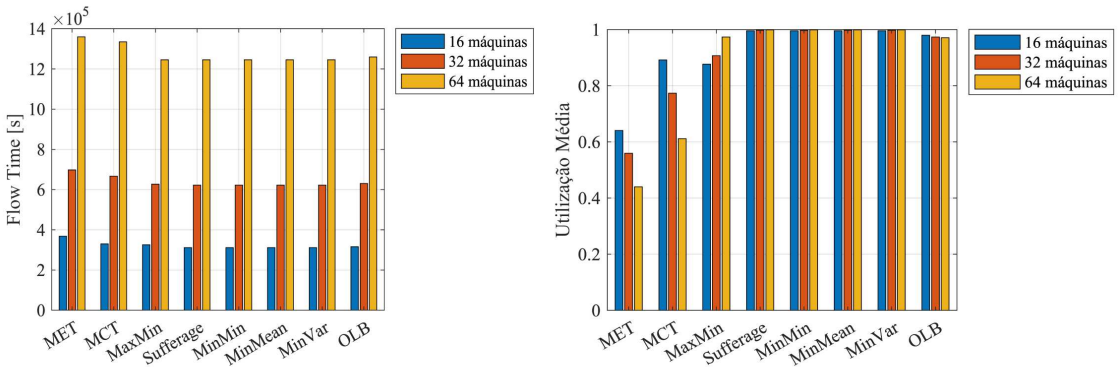
Makespans - Cenário 4			
Algoritmo	16 máquinas	32 máquinas	64 máquinas
MET	36632 ±16,30%	40082 ±18,78%	49966 ±20,54%
MCT	23108 ± 2,93%	26941 ± 3,47%	34121 ± 4,03%
MaxMin	23274 ± 6,47%	21616 ± 4,84%	20026 ± 6,22%
XSufferage	19544 ± 0,75%	19499 ± 0,75%	19456 ± 0,72%
MinMin	19544 ± 0,75%	19500 ± 0,75%	19456 ± 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 12, 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 12, pode-se observar que a *utilização de recurso* média é bem alta para a maioria dos algoritmos.

Figura 12 – 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
XSufferage	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).

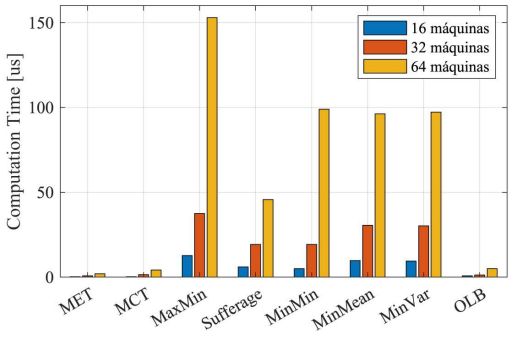
Tabela 14 – Utilização de Recurso dos algoritmos para o Cenário 4.

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
XSufferage	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 13 – 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
XSufferage	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: *XSufferage*, *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 *XSufferage*, 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 *XSufferage*, *MinMin*, *MinMean* e *MinVar*. Para as métricas de *makespan* e *flowtime*, o algoritmo *MinMean* conseguiu registrar os melhores resultados.

Referências

BARDSIRI, A. K.; HASHEMI, S. M. A comparative study on seven static mapping heuristics for grid scheduling problem. *International Journal of Software Engineering and Its Applications*, v. 6, n. 4, p. 247–254, jan. 2012. Citado 2 vezes nas páginas 16 e 21.

BORGES, B.; ERICKSON, K. *Motivos para migrar para Java 11 e além*. 2022. Disponível em: <https://docs.microsoft.com/pt-br/java/openjdk/reasons-to-move-to-java-11>. Acesso em: 13 ago. 2022. Citado na página 17.

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 2 vezes nas páginas 11 e 18.

BUSSAB, W. de O.; MORETTIN, P. A. *Estatística Báscia*. 9. ed. São Paulo: Saraiva Uni, 2017. 568 p. ISBN 978-8547220228. Citado 2 vezes nas páginas 17 e 27.

CASANOVA, H.; LEGRAND, A.; ZAGORODNOV, D.; BERMAN, F. Heuristics for scheduling parameter sweep applications in grid environments. In: *Proceedings 9th Heterogeneous Computing Workshop*. EUA: IEEE Comput. Soc, 2000. p. 15. Citado na página 20.

DONG, F.; AKL, S. G. Scheduling algorithms for grid computing: State of the art and open problems. *Technical Report No. 2006-504*, p. 1–10, 2006. Citado 2 vezes nas páginas 13 e 14.

GAREY, M. R.; JOHNSON, D. S. *Computers and intractability*: a guide to the theory of np-completeness. EUA: W. H. Freeman, 1979. 338 p. (A Series of Books in the Mathematical Sciences). ISBN 0716710447. Citado na página 15.

IBARRA, O. H.; KIM, C. E. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, Association for Computing Machinery (ACM), v. 24, n. 2, p. 280–289, abr. 1977. Citado 2 vezes nas páginas 19 e 20.

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, 2010. Citado na página 23.

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. Citado na página 15.

OPENSTAX. *Introduction to Business*. Rice University: XanEdu Publishing Inc, 2018. 732 p. ISBN 9781593995485. Disponível em: <https://openstax.org/books/introduction-business/pages/13-6-trends-in-information-technology>. Acesso em: 13 ago. 2022. Citado 2 vezes nas páginas 13 e 14.

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 3 vezes nas páginas 15, 16 e 26.

STOCKINGER, H. Defining the grid: a snapshot on the current view. *The Journal of Supercomputing*, Springer Science and Business Media LLC, v. 42, n. 1, p. 3–17, mar. 2007. Citado na página 13.

VERMA, A. *Distributed Scheduling Simulator*. 2010. Disponível em: <https://github.com/dapurv5/distributedscheduling>. Acesso em: 13 ago. 2022. Citado 4 vezes nas páginas 11, 12, 17 e 18.

VIR, R.; VASUDEVA, R.; SHARMA, V.; SANDEEP. Optimised scheduling algorithms and techniques in grid computing. In: BENAVENTE-PECES, C.; SLAMA, S. B.; ZAFAR, B. (Ed.). *Proceedings of the 1st International Conference on Smart Innovation, Ergonomics and Applied Human Factors*. Cham: Springer International Publishing, 2019, (Smart Innovation, Systems and Technologies, v. 150). p. 231–244. Citado na página 15.

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 4 vezes nas páginas 11, 13, 15 e 16.