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.

**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 Xsuffrage 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)

Algoritmos/heurísticas

**MET (Minimum Execution Time)**

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).

**Pseudocódigo MET**

para tarefa em tarefas {

tempoMínimo = valor máximo positivo

para máquina em máquinas {

se (etc[tarefa][máquina] < tempoMínimo) {

tempoMínimo = etc[tarefa][máquina]

máquinaEscolhida = máquina

}

}

atribuir tarefa para máquinaEscolhida

}

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.

**Exemplo Simples MET:**

(Essa tabela é utilizada para MET, OLB, MCT, MIN-MIN)

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.

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).

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

**Pseudocódigo OLB**

Para tarefa em tarefas {

minMatTime = valor máximo positivo

Para máquina em máquinas {

Se (mat[máquina] < minMatTime) {

minMatTime = mat[máquina]

máquinaEscolhida = máquina

}

}

atribuir tarefa para máquinaEscolhida

}

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

**Exemplo Simples 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.

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.

**MCT (Minimum Completion Time)**

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\_mode\_scheduling\_in\_grid\_systems.pdf)

**Pseudocódigo MCT**

Para tarefa em tarefas {

tempoMínimo = valor máximo positivo

para máquina em máquinas {

se (etc[tarefa][máquina] + mat[máquina] < tempoMínimo) {

tempoMínimo = etc[tarefa][máquina] + mat[máquina]

máquinaEscolhida = máquina

}

}

atribuir tarefa para máquinaEscolhida

}

**Exemplo simples 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 maior que 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 menor que das outras máquinas, assim a tarefa 3 é atribuída a ela.

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).

**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.

**Pseudocódigo Min-Min**

Faça{

máquinaEscolhida = 0

tarefaEscolhida = 0

tempoMinimo = valor máximo positivo

para tarefa em tarefas{

se(foiRemovida[tarefa])

continue para a próxima iteração

para máquina em máquinas{

se (etc[tarefa][máquina] + mat[máquina] < tempoMinimo){

tempoMinimo = etc[tarefa][máquina] + mat[máquina]

máquinaEscolhida = máquina

tarefaEscolhida = tarefa

}

}

}

atribuir tarefaEscolhida para máquinaEscolhida

foiRemovida[tarefaEscolhida] = Verdadeiro

}Enquanto(houver tarefas não atribuidas)

**Exemplo simples Min-Min:**

Tarefa Máquina 1 Máquina 2 Máquina 3

1 125 50 115

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, menor que 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.

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).

**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.

**Pseudocódigo Max-Min**

Faça{

tempoMinimo = valor máximo positivo

maxMinComplTime = valor mínimo negativo;

para tarefa em tarefas {

se(foiRemovida[tarefa])

continue para a próxima iteração

para máquina em máquinas {

se (etc[tarefa][máquina] + mat[máquina] < tempoMinimo) {

tempoMinimo = etc[tarefa][máquina] + mat[máquina]

máquinaEscolhida = máquina

}

}

minComplTime[tarefa] = tempoMinimo;

minComplMachine[tarefa] = máquinaEscolhida;

se (maxMinComplTime < minComplTime[tarefa]) {

maxMinComplTime = minComplTime[tarefa];

tarefaEscolhida = tarefa;

}

}

atribuir tarefaEscolhida para minComplMachine[tarefaEscolhida]

foiRemovida[tarefaEscolhida] = Verdadeiro

}Enquanto(houver tarefas não atribuidas)

**Exemplo simples 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.

Máquina Tarefas Tempo total

1 3 50

2 1 50

3 4 50+20

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).

**XSuffrage**

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 se não for atribuída a máquina com melhor MCT.

**Pseudocódigo XSuffrage**

Faça{

tempoMinimo1 = valor máximo positivo;

máquina1 = -1;

tempoMinimo2 = valor máximo positivo;

máquina2 = -1;

tarefaEscolhida = -1;

máquinaEscolhida = -1;

sufferageMaximo = valor mínimo negativo

para tarefa em tarefas faça{

para máquina em máquinas faça{

se ((etc[tarefa][máquina] + mat[máquina]) < tempoMinimo1) {

tempoMinimo1 = etc[tarefa][máquina] + mat[máquina];

máquina1 = máquina;

}

}

para máquina em máquinas faça{

se (máquina != máquina1 E (etc[tarefa][máquina] + mat[máquina]) < tempoMinimo1) {

tempoMinimo2 = etc[tarefa][máquina] + mat[máquina];

máquina2 = máquina;

}

}

sufferage = tempoMinimo1 - tempoMinimo2;

se sufferage > sufferageMaximo{

sufferageMaximo = sufferage;

tarefaEscolhida = tarefa;

máquinaEscolhida = máquina1;

}

tempoMinimo1 = valor máximo positivo;

tempoMinimo2 = valor máximo positivo;

}

atribuir tarefaEscolhida para máquinaEscolhida

foiRemovida[tarefaEscolhida] = Verdadeiro

}Enquanto(houver tarefas não atribuidas)

**Exemplo simples XSuffrage:**

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 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).

Máquina Tarefas Tempo total

1 1 125

2 3 43

3 4 50

A partir do resultado gerado pelo Max-Min, contata-se que o makespan é de 125 segundos, e o flowtime de 218 segundos (125+43+50).

**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 maior que a média. Caso seja maior, verifica se atribuir uma das tarefas dela para outra máquina, a média pode diminuir. Portanto, o objetivo dessa heurística é redistribuir melhor as tarefas, para isso otimiza a utilização das máquinas, diminuindo o tempo de ociosidade médio delas

**Pseudocódigo Min-Mean**

Execute o Min-Min

tempoConclusãoTotal = 0

Para máquina em máquinas

tempoConclusãoTotal += mat[máquina]

média = tempoConclusãoTotal/ qtdmáquinas;

Para máquina em máquinas {

Se (mat[máquina] <= média)

continue para a próxima iteração

Para tarefa em (tarefas atribuídas a máquina) {

delta = 0

Para máquinaCandidata em máquinas {

Se (máquinaCandidata == máquina OU mat[máquinaCandidata] > média)

continue para a próxima iteração

Se ((mat[máquinaCandidata] + etc[tarefa][máquinaCandidata]) < média) {

novoTempoEstimado = (mat[máquinaCandidata] + etc[tarefa][máquinaCandidata])

Se((média - novoTempoEstimado) > delta){

delta = média - novoTempoEstimado

máquinaEscolhida = máquinaCandidata

}

}

}

Se (delta > 0){

remover tarefa da máquina

atribuir tarefa para máquinaEscolhida

}

}

}

**Exemplo simples 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

Resultado Min-Min

Máquina Tarefas Tempo total

1 nenhuma 0

2 2 e 1 80

3 3 e 4 60

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, por não ter nenhuma tarefa atribuída a ela, possui mat igual a 0, menor que a média de 46. Assim, o algoritmo avança para a iteração seguinte, na qual a máquina 2, que tem um tempo de processamento (mat) igual a 80 segundos, superior à média, pode ter suas tarefas redistribuídas para outra máquina. Para cada uma das tarefas anteriormente atribuídas para a máquina 2, é analisado se as outras máquinas se tivessem a tarefa atribuídas a ela, iria diminuir a média. Dessa forma, a máquina 1 passa a ser uma máquina candidata. Se a tarefa 1 fosse transferida para a máquina 1, o tempo de processamento resultante seria maior do que a própria média, logo essa alternativa é eliminada. Se a tarefa 2 fosse transferida para a máquina 1, o tempo de processamento resultante seria de 40 segundos, inferior à média, e a diferença entre a média e esse novo tempo possível seria de 6 segundos, superior ao delta inicial, que é 0. Logo, o valor do delta é atualizado para 6, e a máquina 1 é definida como a máquinaEscolhida.

As mesmas comparações são feitas para a máquina 3, mas se ela receber a tarefa 1 ou 2, somado com o tempo gasto com as tarefas já atribuídas a ela, o tempo seria superior a média. Portanto, a única tarefa que pode ser redistribuída foi a tarefa 2 para a máquina 1.

Na iteração seguinte, ao examinar quais tarefas atribuídas à máquina 3 podem ser redistribuídas, observa-se que nenhuma tarefa satisfaz essa condição.

Resultado Max-Min

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, menor que o makespan gerado anteriormente pelo Min-Min (80), e o flowtime de 150 segundos (40+50+60).

**Min-Var**

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

Onde:

σ2 é a variância;

xi é o valor observado da i-ésima máquina;

é a média aritmética de todos os valores;

n é o número total de observações.

**Pseudocódigo Min-Var**

Execute o Min-Min

Para máquina em máquinas

tempoConclusãoTotal = mat[máquina]

média = tempoConclusãoTotal / qtdmáquinas;

Para máquina em máquinas

variancia = variancia + (mat[máquina]-média)²

variancia = variancia/qtdmáquinas;

Para máquina em máquinas {

Se (mat[máquina] < média)

continue para a próxima iteração

double difMatMedia = (mat[máquina]-média)²

Se (difMatMedia < variancia)

continue para a próxima iteração

Para tarefa em (tarefas atribuídas a máquina){

difMatMediaPossible = 0;

delta = 0;

Para máquinaCandidata em máquinas{

Se (máquinaCandidata == máquina)

continue para a próxima iteração

Se (mat[máquinaCandidata] < média){

difMatMediaPossible = ((mat[máquinaCandidata] + etc[tarefa][máquinaCandidata]) - média)²

}

Senão{

difMatMediaPossible = (mat[máquinaCandidata] - média)²

Se (difMatMediaPossible > variancia)

continue;

difMatMediaPossible = ( (mat[máquinaCandidata] + etc[tarefa][máquinaCandidata]) - média)²

//Pode ser maior que a média, mas o importante é que a diferença entre o mat e a média não seja maior que a variancia

}

Se(difMatMediaPossible > variancia)

continue;

Se (difMatMediaPossible <= variancia) {

Se ((variancia - difMatMediaPossible) >= delta){

delta = variancia - difMatMediaPossible;

máquinaEscolhida = máquinaCandidata;

}

}

}

Se (delta > 0){

remover tarefa da máquina

atribuir tarefa para máquinaEscolhida

}

}

}

**Exemplo simples 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

Resultado Min-Min

Máquina Tarefas Tempo total

1 nenhuma 0

2 2 e 1 80

3 3 e 4 60

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,

σ2 = ((0-46)² + (80-46)² + (60-46)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 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, (80-46)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 = 79² = 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 = 129² = 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 = 16² = 256

Por ser menor que a variância, segue para o calculo do delta, que é a diferença entre a variância e o difMatMediaPossible:

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.

Resultado Min-Var

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.

**Referências**

[1] – SINGH, S.; BAWA, R. **An efficient job scheduling algorithm for grid computing**. Proceedings of the International Conference on Advances in Computing and Artificial Intelligence. **Anais**... Em: ACAI ’11: INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING AND ARTIFICIAL INTELLIGENCE. Rajpura/Punjab India: ACM, 21 jul. 2011. Disponível em: <<https://dl.acm.org/doi/10.1145/2007052.2007097>>. Acesso em: 9 fev. 2023

[2] – BHATIA, M. **RR based grid scheduling algorithm**. Proceedings of the International Conference on Advances in Computing and Artificial Intelligence. **Anais**... Em: ACAI ’11: INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING AND ARTIFICIAL INTELLIGENCE. Rajpura/Punjab India: ACM, 21 jul. 2011. Disponível em: <<https://dl.acm.org/doi/10.1145/2007052.2007076>>. Acesso em: 10 fev. 2023

[3] – ZHANG, Y.; INOGUCHI, Y.; SHEN, H. A Dynamic Task Scheduling Algorithm forGrid Computing System. Em: CAO, J. et al. (Eds.). **Parallel and Distributed Processing and Applications**. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. v. 3358p. 578–583.

<https://sci-hub.hkvisa.net/10.1007/978-3-540-30566-8_69>

[4] – BRAUN, T. D. et al. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. **Journal of Parallel and Distributed Computing**, v. 61, n. 6, p. 810–837, jun. 2001.

<https://sci-hub.ru/https:/www.sciencedirect.com/science/article/abs/pii/S0743731500917143>

[5] – DONG, Fangpeng; AKL, Selim G. Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. *[S. l.]*, [s.d.].

<https://research.cs.queensu.ca/home/akl/techreports/GridComputing.pdf>

[6] – HEMAMALINI, M.; SRINATH, M. V. STATE OF THE ART: TASK SCHEDULING ALGORITHMS IN A HETEROGENEOUS GRID COMPUTING ENVIRONMENT. *[S. l.]*, v. 1, n. 1, 2014.

[State of the art: Task scheduling Algorithms in Grid Computing Environment (ejerm.com)](http://ejerm.com/vol1_aug2014/img/cs004%20%20CRC%20STATE%20OF%20THE%20ART%20TASK%20SCHEDULING%20ALGORITHMS%20IN%20A%20HETEROGENEOUS%20GRID%20COMPUTING%20ENVIRONMENT.pdf)

[7] –VERMA, Apurv. **dapurv5/distributedscheduling**. 2022. Disponível em: <https://github.com/dapurv5/distributedscheduling>. Acesso em: 10 fev. 2023.

[8] - BARRA, Marcos Victor de Aquino. **Comparação de algoritmos para o problema de escalonamento de tarefas em grades computacionais**. Orientador: Paulo Henrique Ribeiro Gabriel. 2022. 45 p. Trabalho de conclusão de curso (Bacharelado em Ciência da Computação) - Universidade Federal de Uberlândia, Uberlândia, 2022.