

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Carla de Mello Caixeta

**Implementação de modificações em um
simulador de escalonamento de tarefas em
sistemas distribuídos**

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Carla de Mello Caixeta

**Implementação de modificações em um simulador de
escalonamento de tarefas em sistemas distribuídos**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Sistemas de Informação.

Orientador: Paulo Henrique Ribeiro Gabriel

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2018

Resumo

Essa monografia tem como objetivo descrever as modificações realizadas no simulador desenvolvido por [Maheswaran *et al.* \(1999\)](#) e [Kamalam e Muralibhaskaran \(2010\)](#). O simulador, escrito em linguagem em Java, tem como função simular algoritmos de escalonamento de tarefas em sistemas computacionais distribuídos. O simulador original possui algumas limitações, como não permitir inserir independentemente características de máquinas e tarefas e apresentar apenas a média do tempo de execução de cada algoritmo para a análise estatística dos resultados. As modificações feitas foram realizadas alterando algumas classes existentes no sistema e criando novas classes para novas funcionalidades, na mesma linguagem de programação do simulador original. As modificações realizadas não alteraram os resultados da nova simulação, demonstrando que o simulador modificado é apenas uma extensão do original.

Palavras-chave: Escalonamento de tarefas, simulação, sistemas distribuídos.

Lista de ilustrações

Figura 1 – Exemplo de Matriz ETC	11
Figura 2 – Diagrama de Classes do Simulador	14
Figura 3 – Classe Computer	15
Figura 4 – Classe Task	16
Figura 5 – Classe Processor	16
Figura 6 – Classe DesvioPadrao	17
Figura 7 – Diagrama de Classes do Simulador Alterado	18
Figura 8 – Primeira execução do simulador original	20
Figura 9 – Segunda execução do simulador original	21
Figura 10 – Terceira execução do simulador original	21
Figura 11 – Primeira execução do simulador modificado	22
Figura 12 – Segunda execução do simulador modificado	22
Figura 13 – Terceira execução do simulador modificado	23

Sumário

1	INTRODUÇÃO	5
1.1	Contextualização e Motivação	5
1.2	Objetivo	6
1.3	Organização da Monografia	6
2	REFERENCIAL TEÓRICO	7
2.1	Escalonamento de Tarefas	7
2.2	Algoritmos de Escalonamento	8
3	DESENVOLVIMENTO	10
3.1	Apresentação	10
3.2	Simulador Original	10
3.3	Modificações Realizadas	13
3.4	Novo Diagrama de Classes	17
3.4.1	Eliminação de Classes	19
4	RESULTADOS	20
4.1	Simulação Original	20
4.2	Simulação Modificada	20
5	CONCLUSÃO	24
5.1	Trabalhos Futuros	24
	REFERÊNCIAS	25

1 Introdução

1.1 Contextualização e Motivação

Um tópico emergente na computação é a utilização de sistemas distribuídos heterogêneos na execução de aplicações paralelas (compostas por tarefas) em um conjunto determinado de máquinas. No sistema considerado nesse trabalho, as tarefas são independentes, não é necessário que elas se comuniquem entre si para serem executadas.

Os algoritmos de escalonamento determinam a ordem em que essas tarefas serão executadas e em qual máquina. Existem diversos algoritmos (XHAFA; ABRAHAM, 2010) para realizar tal escalonamento, cada um com determinadas características e limitações. Tais limitações têm motivado novos trabalhos a propor formas para lidar com o problema de escalonamento (KOŁODZIEJ, 2012).

Na avaliação desses algoritmos, duas abordagens têm sido comumente utilizadas: *i*) implementação em ambientes reais; e *ii*) simulação. A primeira, embora realista, tem como limitação o fato de que, muitas vezes, não se dispõe de sistemas computacionais dedicados apenas ao estudo de algoritmos de escalonamento. Além disso, mesmo quando disponíveis, tais sistemas podem sofrer variações em seu desempenho devido a fatores externos (queda de energia, latência de rede, etc.).

Por outro lado, simulações fornecem um ambiente controlado, onde todas as variáveis do sistema são definidas pelo usuário, além de permitirem a exata coleta das informações desejadas, de modo a conduzir testes estatísticos confiáveis.

No contexto de escalonamento de tarefas, diferentes simuladores têm sido projetados, alguns dos quais modelam ambientes de grande complexidade (CASANOVA *et al.*, 2014; KLUSÁČEK; RUDOVÁ, 2010).

O simulador desenvolvido por Maheswaran *et al.* (1999) não se caracteriza por ter grande complexidade, embora seja bastante adequado para simulações mais simples e provas de conceito. Esse simulador, estendido posteriormente por Kamalam e Muralibhas-karan (2010), é implementado em Java e contém vários algoritmos de escalonamento de tarefas em múltiplas máquinas, permitindo a comparação entre os métodos já existentes nele, além de ser flexível por permitir a inclusão de novos algoritmos.

No entanto, esse simulador é baseado em um conhecimento prévio dos custos de execução de tarefas nas diferentes máquinas do sistema; além disso, não permite avaliações estatísticas mais detalhadas, limitando-se à apresentação de médias.

1.2 Objetivo

O principal objetivo deste trabalho de conclusão de curso é modificar o simulador desenvolvido por [Maheswaran *et al.* \(1999\)](#) e [Kamalam e Muralibhaskaran \(2010\)](#), utilizado para simular algoritmos de escalonamento de tarefas em sistemas computacionais distribuídos ([XHAFa; ABRAHAM, 2010](#)). O simulador original não permite especificar, independentemente, características de tarefas e computadores. Além disso, apresenta como limitação o fato de não proporcionar meios para uma análise mais detalhada das soluções, uma vez que não traz funcionalidades como salvar dados relativos à carga de trabalho em cada computador do sistema.

Com esse trabalho, espera-se produzir uma versão do simulador, mais flexível, que possa ser utilizada em outros trabalhos do grupo de pesquisa.

Com a finalidade de atingir esse objetivo, foi proposta uma alteração no simulador original para proporcionar o cálculo do desvio-padrão e informar dados específicos a respeito de máquinas e tarefas. As alterações feitas se resumem em criação de novas classes, atributos e métodos, referentes às novas informações e dados estatísticos presentes no novo programa e alterações em classes já existentes para expandir funcionalidades já presentes no simulador original.

As mudanças realizadas não afetaram o funcionamento do simulador. Isso se justifica pelo fato de que os dados resultantes da simulação na nova versão são consistentes e proporcionais aos resultados do simulador original.

1.3 Organização da Monografia

O restante dessa monografia está organizado da seguinte maneira: no Capítulo [2](#) é apresentado o referencial teórico, referente ao problema de escalonamento de tarefas e alguns algoritmos utilizados na sua resolução. No Capítulo [3](#), detalha-se o simulador original e as modificações propostas. O novo simulador é validado em experimentos descritos no Capítulo [4](#). Finalmente, o Capítulo [5](#) traz as conclusões desse trabalho.

2 Referencial Teórico

2.1 Escalonamento de Tarefas

Na sociedade atual, a necessidade por computação de alto desempenho é algo crescente, assim como o tamanho de centros de armazenamento de dados (*datacenters*) e a energia que eles demandam, resultando em um aumento da emissão de carbono. Para lidar com esse problema, são estudados heurísticas eficientes de escalonamento em computação de alto desempenho, o que reduz o consumo de energia sem afetar o sistema (GUPTA; HO; STATE, 2000; XHAFA; ABRAHAM, 2009).

Conceitualmente, escalonamento de tarefas consiste na alocação eficiente de aplicações paralelas sobre os recursos disponíveis em um sistema distribuído (XHAFA; BAROLLI; DURRESI, 2007b). O escalonamento de processos discute o agendamento das tarefas na CPU em sistemas paralelos e distribuídos. O agendamento da CPU é parte de uma classe mais ampla de problemas de alocação de recursos, e é provavelmente o mais cuidadosamente estudado. A motivação principal para o estudo do escalonamento é o desejo de aumentar a velocidade na execução de uma carga de trabalho. Partes da carga de trabalho, chamadas tarefas, podem ser espalhadas por vários processadores e, portanto, serem executadas mais rapidamente do que em um único processador. O problema do escalonamento de processos em sistemas multiprocessados pode ser geralmente formulado pela questão: "Como podemos executar um conjunto de tarefas T em um conjunto de processadores P sujeito a algum conjunto de critérios de otimização C ?"

O objetivo mais comum do escalonamento é minimizar o tempo de execução esperado de um conjunto de tarefas. Exemplos de outros critérios de agendamento incluem a minimização do custo, diminuindo o atraso na comunicação, dando prioridade aos processos de determinados usuários, ou necessidade de dispositivos de hardware especializados. A política de escalonamento para um sistema multiprocessado geralmente incorpora uma mistura de vários desses critérios (XHAFA; BAROLLI; DURRESI, 2007b; XHAFA; BAROLLI; DURRESI, 2007a)

A computação em grade (STOCKINGER, 2007) possui fontes homogêneas em sua maioria, mas atualmente a tendência é utilizar fontes heterogêneas. Essas, possuem diferentes capacidades de processamento e arquiteturas que executam tarefas heterogêneas. O escalonamento de tarefas no entanto se torna mais complexo e essencial para o uso eficiente de recursos (XHAFA; BAROLLI; DURRESI, 2007b).

Devido à alta complexidade do problema de escalonamento não é possível determinar uma solução ideal porque o tempo gasto para explorar todo o espaço de pesquisa não

é razoável. Dessa forma, é necessário aproximar métodos para encontrar uma boa solução com alta eficiência computacional em vez de algoritmos que realizam buscas exaustivas e exigem um grande tempo computacional.

2.2 Algoritmos de Escalonamento

Diversas heurísticas têm sido propostas no contexto de escalonamento de processos em sistemas heterogêneos. Braunt *et al.* (2001) propôs um modelo de simulação, comparando onze diferentes heurísticas. Posteriormente, Kamalam e Muralibhaskaran (2010) estenderam esse modelo, produzindo um simulador no qual são implementados os seguintes algoritmos:

Minimum Execution Time (MET): designa cada tarefa para a máquina com o melhor tempo de execução esperado para essa tarefa, independente da disponibilidade da máquina. Seu objetivo é atribuir a cada tarefa sua melhor máquina porém isso pode causar um desequilíbrio e carga severa nas máquinas (MAHESWARAN *et al.*, 1999; BRAUNT *et al.*, 2001).

Minimum Completion Time (MCT): designa cada tarefa para a máquina com o menor tempo de conclusão esperado para essa tarefa, porém resulta em algumas tarefas que não possuem o mínimo tempo de execução para elas (MAHESWARAN *et al.*, 1999; BRAUNT *et al.*, 2001).

Min-Min: baseado no MCT, no entanto, Min-Min considera todas as tarefas não mapeadas durante cada decisão de mapeamento enquanto que no MCT é considerada apenas uma tarefa por vez (IBARRA; KIM, 1977; BRAUNT *et al.*, 2001).

Max-Min: é bastante similar a Min-Min porém esse algoritmo seleciona a tarefa com o tempo máximo de conclusão em determinada máquina e atribui essa tarefa a essa máquina. O Max-Min tenta minimizar as penalidades ocorridas ao executar tarefas com longos tempos de execução (IBARRA; KIM, 1977; BRAUNT *et al.*, 2001).

Sufferage: a ideia desse algoritmo é que os melhores mapeamentos devem ser gerados ao atribuir uma determinada máquina a uma tarefa que seria mais afetada em termos de conclusão de tempo esperada caso aquela máquina em particular não fosse designada a essa tarefa (MAHESWARAN *et al.*, 1999).

Min-Mean: esse algoritmo é definido em duas partes. A primeira executa o mapeamento baseado no Min-Min. A partir desse mapeamento, uma média de tempo de conclusão de todas as máquinas é calculada e as máquinas que possuírem o tempo de conclusão maior que o tempo médio são selecionadas. As tarefas designadas a essas máquinas

são redistribuídas entre as máquinas que possuírem tempo de conclusão menor que o médio calculado (KAMALAM; MURALIBHASKARAN, 2010).

Kamalam e Muralibhaskaran (2010) apresentam uma comparação desses algoritmos, utilizando uma versão estendida do simulador proposto por Maheswaran *et al.* (1999)¹. No entanto, a simulação proposta apresenta diversas limitações, como, por exemplo, o fato de não permitir uma análise estatística mais detalhada, limitando-se ao cálculo de uma média simples das soluções.

Além disso, o simulador não traz um modelo de computadores, o que o torna menos flexível (uma vez que não permite a simulação de diferentes sistemas computacionais). Sem esse modelo, não é possível avaliar os algoritmos em relação a outras métricas de escalonamento, como taxa de ocupação das máquinas, atraso na execução de tarefas, tempo médio de execução, etc. Tais informações podem ser obtidas quando se conhece a carga de trabalho em cada máquina.

No entanto, o simulador de (KAMALAM; MURALIBHASKARAN, 2010) é adequado na modelagem de tempo de resposta e de chegada das tarefas ao sistema, além de ser bem modularizado, facilitando a implementação de novos algoritmos. Por esse motivo, este trabalho de conclusão de curso visa estender esse simulador, deixando-o mais flexível e permitindo a extração de informações adequadas nos experimentos.

¹ O simulador utilizado está disponível, via licença GPL, no link: <<https://github.com/dapurv5/distributedscheduling>> (Acesso em 14 jul. 2018).

3 Desenvolvimento

3.1 Apresentação

Esta monografia tem como objetivo realizar alterações no simulador proposto por [Kamalam e Muralibhaskaran \(2010\)](#) de modo que seja possível especificar atributos de tarefas e computadores de maneira independente e prover um maior detalhamento dos resultados, tornando a comparação e a extração de dados entre os diversos algoritmos propostos na simulação mais versátil.

Como o paradigma de computação utilizando sistemas distribuídos heterogêneos está se tornando um tópico cada vez mais pesquisado e utilizado, mais detalhes serão necessários extrair para realizar uma análise completa e coerente dos dados dessas pesquisas. Um dos assuntos desse tema são os algoritmos de escalonamento por esse motivo é importante detalhar cada vez mais o simulador proposto para que contenha os detalhes necessários para a devida análise.

3.2 Simulador Original

O processo de simulação do programa original é feito utilizando um simulador de eventos discretos. A chegada de tarefas é calculada utilizando a distribuição de Poisson ([BUSSAB; MORETTIN, 2017](#)) que irá modelar o número de tarefas que chegam ao sistema por um intervalo de tempo.

Em seguida, o simulador considera uma matriz ETC (do inglês, *expected time to compute* – tempo de computação esperado) que contém o tempo de execução esperado para uma tarefa em um conjunto de máquinas, para todas as tarefas ([KOŁODZIEJ, 2012](#)).

A matriz ETC contém linhas que indicam as tarefas a serem executadas e colunas que indicam as máquinas a serem usadas. Cada dado armazenado nessa matriz irá representar um tempo de execução esperado para determinada tarefa executada em uma máquina.

A Figura 1 é um exemplo de matriz ETC que possui 3 máquinas representadas pelas colunas e 10 tarefas representadas pelas linhas. Nesse exemplo, para a tarefa 1, o tempo de execução esperado na máquina 1 é de 5 segundos; já na máquina 2 é de 29 segundos e, finalmente, para a máquina 3 é de 21 segundos. Essa figura foi obtida pela execução do simulador, após a geração da matriz.

A variação média ao longo das linhas da matriz é chamada de *heterogeneidade de*

Figura 1 – Exemplo de Matriz ETC

5	29	21
313	157	79
1	15	15
109	163	163
41	25	73
1	1	1
141	491	141
456	547	365
67	78	23
1	101	176

BUILD SUCCESS		

Fonte: Autoria própria (2018)

tarefas. De maneira similar, a variação média ao longo das colunas é chamada de *heterogeneidade de máquinas*. A heterogeneidade pode ser classificada entre alta heterogeneidade e baixa heterogeneidade que podem ser categorizados da seguinte maneira (BRAUNT *et al.*, 2001; KOŁODZIEJ, 2012):

- **HiHi – High Task Heterogeneity and High Machine Heterogeneity:** alta heterogeneidade de tarefas e alta heterogeneidade de máquinas.
- **HiLo – High Task Heterogeneity and Low Machine Heterogeneity:** alta heterogeneidade de tarefas e baixa heterogeneidade de máquinas.
- **LoHi – Low Task Heterogeneity and High Machine Heterogeneity:** baixa heterogeneidade de tarefas e alta heterogeneidade de máquinas.
- **LoLo – Low Task Heterogeneity and Low Machine Heterogeneity:** baixa heterogeneidade de tarefas e baixa heterogeneidade de máquinas.

Os valores de alta e baixa heterogeneidade são pré-definidos no programa e, a partir deles, é gerado um número aleatório para as heterogeneidades de tarefas e máquinas. Os valores a serem preenchidos na matriz ETC são aleatórios, mas não ultrapassam os valores definidos como heterogeneidade de tarefa e máquina.

Portanto, quando uma nova simulação é iniciada, o simulador gera os tempos de chegada de cada tarefa e constrói a matriz ETC para, assim, começar a simular as tarefas da fila de tarefas gerada. Todos os algoritmos de escalonamento presentes no programa

são executados e, ao final de toda a execução, é calculada a média do tempo de execução para cada um desses algoritmos.

Dentre as classes do simulador, as principais são as seguintes:

ArrivalGenerator: Calcula a chegada de cada uma das tarefas utilizando a distribuição de Poisson, armazenando esses dados em um vetor.

ETCGenerator: Cria a matriz ETC com base no número de máquinas e tarefas previamente informadas e preenche a mesma com o auxílio dos valores das classes **TaskHeterogeneity** e **MachineHeterogeneity**, que determinam os limites para valores referentes a máquinas e tarefas. Cada célula dessa matriz é resultado da multiplicação de dois números aleatórios, um baseado na heterogeneidade da tarefa e outro baseado na heterogeneidade na máquina.

Heuristic: Nomeia todos os algoritmos de escalonamento presentes nesse simulador.

MachineHeterogeneity: Possui informações sobre os valores altos e baixos de heterogeneidade da máquina.

Main: Determina os dados referentes ao número de máquinas e tarefas a serem utilizados e o valor a ser usado como parâmetro no cálculo da distribuição de Poisson. Cria um atributo do tipo **SimulatorEngine** que recebe os parâmetros determinados anteriormente e após a execução da simulação, calcula a média para cada algoritmo e exibe o resultado em tela.

RandomGenerator: Implementa o cálculo da distribuição de Poisson, que recebe apenas um parâmetro, referente à taxa média de chegada de tarefas, para calcular os outros valores da distribuição.

SimulatorEngine: Determina como é feita a simulação dos algoritmos, recebendo dados necessários para que cada um dos objetos criados possua seus determinados valores. A matriz ETC e os tempos de chegada são instanciados e criados nessa classe, de acordo com os dados informados na classe **Main**.

Task: Possui atributos simples que informam o tempo de chegada, fornecido pela classe **ArrivalGenerator**, o tempo de conclusão, usado na simulação e o tempo esperado para essa tarefa ser executada na máquina atribuída a ela.

TaskComparator: Compara os tempos de conclusão das tarefas, usada na simulação.

TaskHeterogeneity: Possui informações sobre os valores altos e baixos de heterogeneidade da tarefa.

SchedulingEngine: Implementa cada um dos algoritmos definidos na classe **Heuristic**. Também possui uma base para implementação de outros algoritmos de escalonamento. Os algoritmos utilizados nesse simulador, descritos no Capítulo 2, são:

- *Minimum Execution Time* (MET);
- *Minimum Completion Time* (MCT);
- *Min-Min*;
- *Max-Min*;
- *Sufferage*;
- *Min-Mean*.

A Figura 2 representa o diagrama de classes do simulador contendo todas as classes, atributos e métodos. O diagrama foi feito utilizando o *plugin* **easyUML** na IDE NetBeans ¹.

3.3 Modificações Realizadas

Na simulação original, os tempos de espera eram calculados pela matriz ETC, com base nas classes e valores de alta heterogeneidade e baixa heterogeneidade. Não existia nenhum atributo em específico que definia a quantidade de instruções de execução de cada tarefa, assim como não havia uma classe para modelar as máquinas do sistema. Portanto, também não existia um atributo que definisse a capacidade de cada máquina.

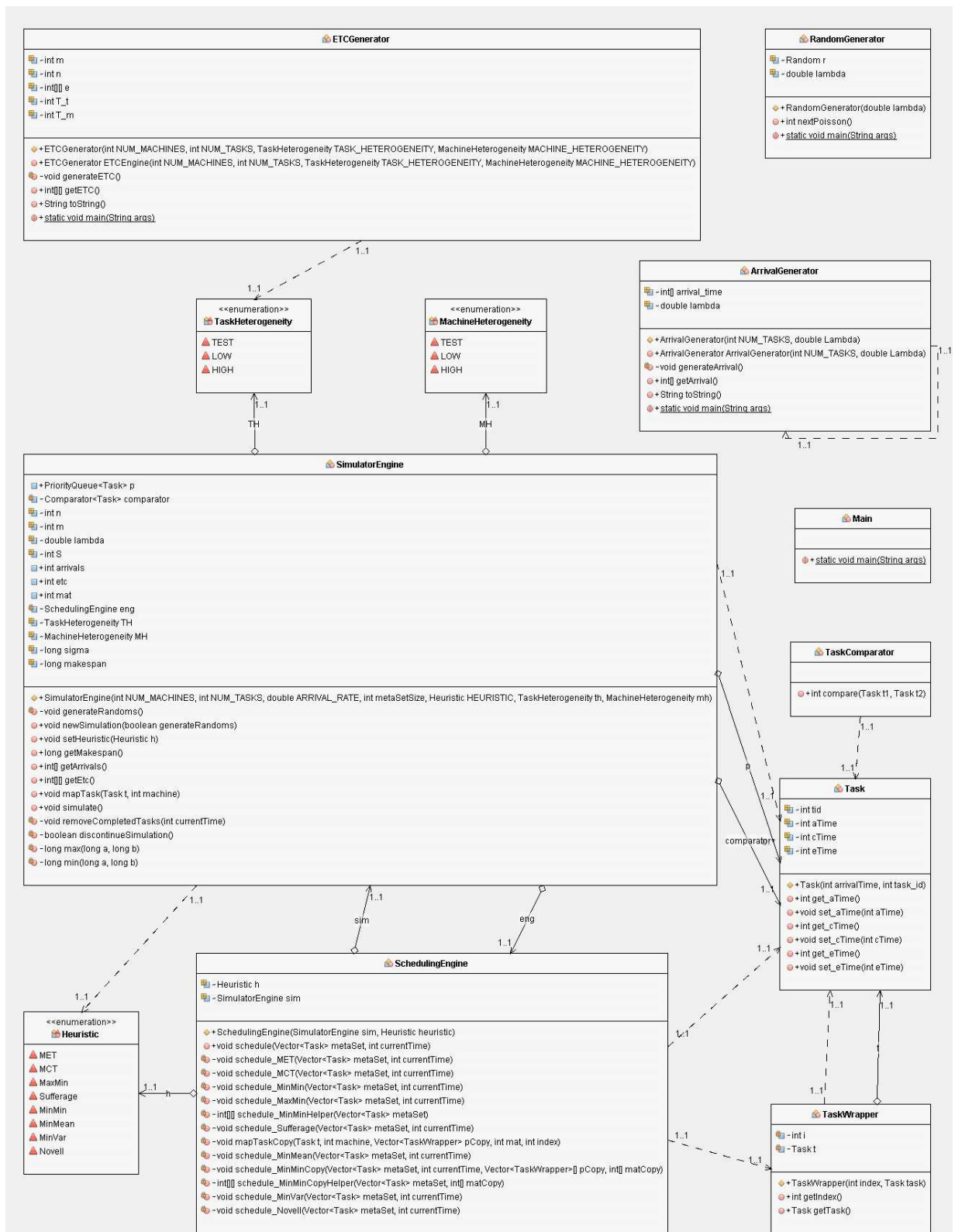
Para realizar essas alterações, a matriz ETC não precisa mais ser utilizada e deve ser substituída por uma outra matriz. A nova matriz foi implementada em uma nova classe, denominada **Computer**. Essa classe possui um atributo **time**, que é uma matriz de inteiros, que representa os tempos de espera, um atributo do tipo **Task**, um atributo do tipo **Processor** e dois inteiros para auxiliar na construção da matriz.

A classe **Computer** é uma classe básica e contém apenas dados sobre a matriz de tempos de espera e sua criação. O tamanho da matriz é definido pelos atributos do número de tarefas e número de máquinas e para preencher a matriz é feita uma divisão do valor da carga de trabalho da tarefa sobre o valor de capacidade da máquina. Esse resultado é atribuído a cada uma das posições da matriz. Os valores de carga de trabalho e de capacidade são gerados aleatoriamente dentro de uma faixa de valores definidos nas classes **Task**, para carga, e **Processor**, para capacidade. Na Figura 3 é possível ver a implementação dessa classe.

A classe **Task** originalmente não possuía a informação da quantidade de instruções que uma tarefa requer, esse dado não era calculado e nem fornecido. Foi necessário

¹ <<https://netbeans.org/>> (Acesso em 14 jul. 2018).

Figura 2 – Diagrama de Classes do Simulador



Fonte: Autoria própria (2018)

Figura 3 – Classe Computer

```

/*Gera a matriz que contem quantas tarefas por segundo de cada maquina*/
public class Computer {
    int [][]time;
    Task task = new Task(1,1);
    Processor processor = new Processor();
    int num_tasks, num_machine;

    public Computer(int NUM_TASKS, int NUM_MACHINE){
        num_tasks = NUM_TASKS;
        num_machine = NUM_MACHINE;
    }

    public void generate(){
        time = new int[num_tasks][num_machine];
        int i,j;
        for(i = 0; i < num_tasks; i++){
            for(j = 0; j < num_machine; j++){
                time[i][j] = task.get_workload()/processor.get_capacity();
            }
        }
    }

    public int[][] getComputer(){
        generate();
        return time;
    }
}

```

Fonte: Autoria própria (2018)

adicionar atributos e métodos a essa classe para que esse valor de carga de trabalho seja criado e gerado. Dentro dessa classe, o atributo **workload** foi criado, assim como atributos auxiliares para a geração de valores aleatórios para esse mesmo atributo. Foram definidos métodos básicos de **get** e **set** para esse atributo e o valor aleatório é gerado dentro do informado previamente na classe pela formula inserida na classe. Na Figura 4, é possível ver os novos atributos e novos métodos da classe **Task**.

No programa original, não existia um dado que informava a capacidade das máquinas a serem utilizadas nas simulações. Para inserir essa informação no simulador, uma nova classe foi criada, a classe **Processor**. Essa classe contém apenas um atributo que indica a capacidade da máquina do computador e é instanciada aleatoriamente dentro de um intervalo de valores pré determinado nessa mesma classe e possui métodos básicos de **set** e **get**. A Figura 5 possui a implementação dessa classe.

O único dado estatístico fornecido pelo simulador original é a média de tempo que as simulações demoravam para serem realizadas. Para aumentar esses dados e a precisão deles, foi adicionado o cálculo do desvio padrão para o tempo em que as simulações demoravam. Dessa maneira, dado o tempo de todas as simulações executadas, é calculado um desvio padrão entre elas para cada um dos algoritmos. Então, se o número de simulações foi definido como 4000, serão computados a média e o desvio padrão desses 4000 valores e para cada um dos algoritmos executados.

Figura 4 – Classe Task

```
/*Quantidade de instruções a tarefa requer*/
int workload = 0;
/*Quantidade de instruções a tarefa requer,
geradas aleatoriamente dentro dos valores máx e min*/
Random workl = new Random();
int min = 10000000;
int max = 30000000;

public int get_workload() {
    gera_workload();
    return this.workload;
}

public void set_workload(int workload) {
    this.workload = workload;
}

public void gera_workload() {
    this.workload = workl.nextInt((max - min) + 1) + min;
}
```

Fonte: Autoria própria (2018)

Figura 5 – Classe Processor

```
public class Processor {
    /*Quantidade de instruções que o processador processa por segundo*/
    int capacity = 0;
    /*Quantidade de instruções o processador processa por segundo,
    geradas aleatoriamente dentro dos valores max e min*/
    Random cap = new Random();
    int min = 100;
    int max = 300;

    public int get_capacity() {
        this.gera_capacity();
        return capacity;
    }

    public void set_capacity(int capacity) {
        this.capacity = capacity;
    }

    public void gera_capacity() {
        capacity = cap.nextInt((max - min) + 1) + min;
    }
}
```

Fonte: Autoria própria (2018)

Para calcular o desvio padrão é necessário que a média já esteja calculada e, para finalizar o cálculo da média, é necessário finalizar simulação. Portanto, para que o desvio padrão seja calculado de forma correta, é necessário armazenar os valores de todas as simulações para todos os algoritmos em uma matriz para que após a finalização das simulações e do cálculo da média, o desvio padrão comece a ser calculado.

Com finalidade de deixar o programa mais organizado, foi criada uma classe para realizar o cálculo do desvio padrão, chamada **DesvioPadrao**. Nessa classe, há a criação da matriz, baseado no número de simulações e no número de heurísticas implementadas no programa e um método que calcula o desvio padrão para cada uma das heurísticas implementadas no simulador. Para realizar esse calculo, a fórmula a ser utilizada é

$$dp = \sqrt{\frac{\sum_{i=1}^{nSim} (valores[i][nAlg] - avg)^2}{nSim}},$$

onde $nSim$ é o número de simulações definidas, $nAlg$ é o número de algoritmos de escalonamento implementados e avg é a média dos tempos de execução. A Figura 6 demonstra a implementação dessa nova classe.

Figura 6 – Classe DesvioPadrao

```
public class DesvioPadrao {
    long [][]valores;
    long desviop;
    int nSim; //número de simulações
    int htype; //número de heurísticas
    int i;

    public DesvioPadrao(int no_of_simulations, int htype){
        valores = new long[no_of_simulations][htype];
        this.nSim = no_of_simulations;
        this.htype = htype;
    }

    public long calculaDesvio(long avg, int nAlg){
        long cal = 0, sum = 0;
        for(i = 0; i < nSim; i++){
            cal = (valores[i][nAlg] - avg); //primeira parte da conta
            cal = (long)Math.pow(cal,2); //cal^2
            sum = sum + cal; //somatorio
        }
        desviop = (long) Math.sqrt((sum/nSim));
        return desviop;
    }
}
```

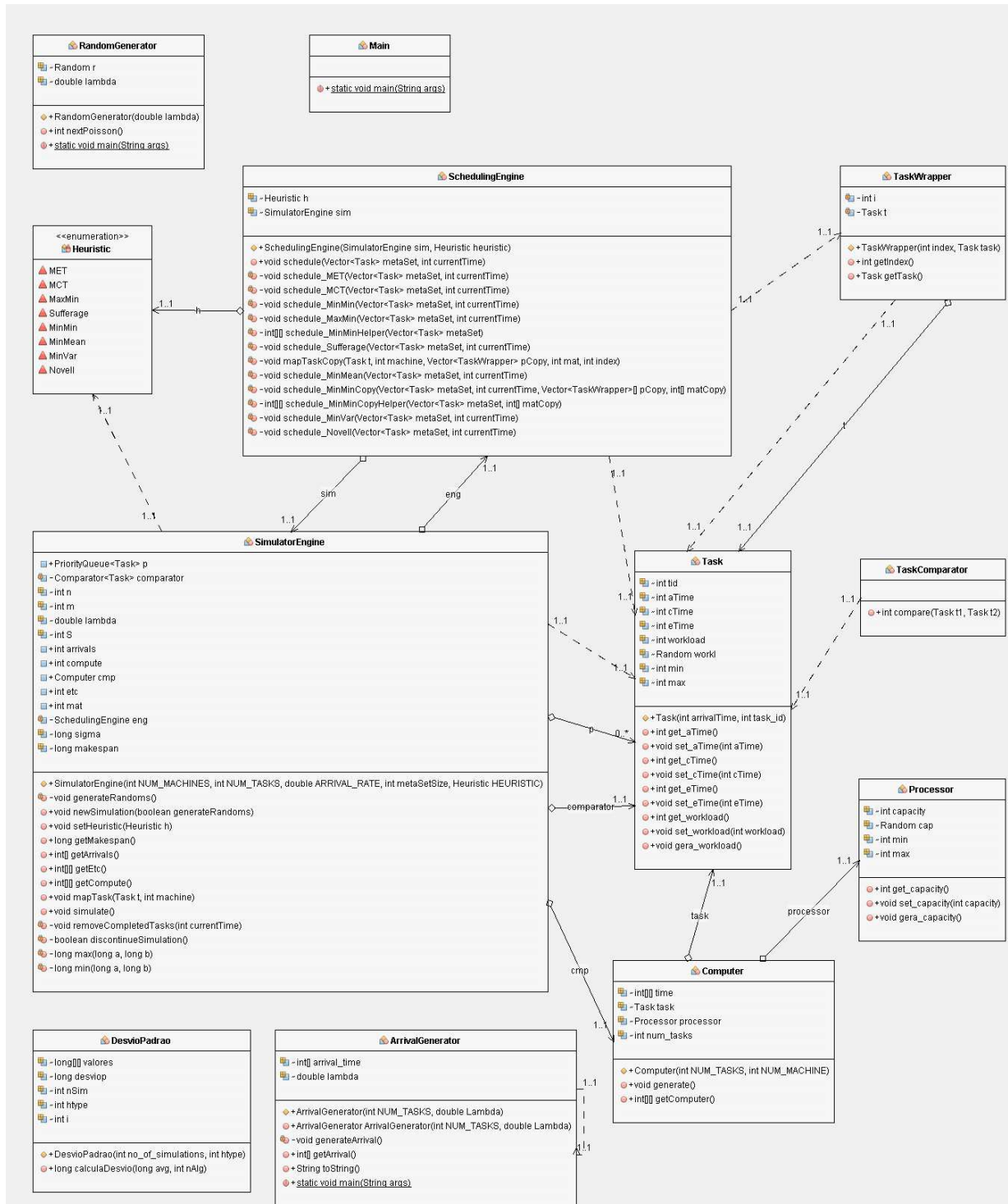
Fonte: Autoria própria (2018)

3.4 Novo Diagrama de Classes

A Figura 7 representa o novo diagrama de classes do simulador demonstrando todas as classes, os atributos e métodos criados, além das classes já existentes no simulador.

É possível analisar também quais classes não serão mais utilizadas, comparando esse diagrama com o da Figura 2. O diagrama foi feito utilizando o *plugin easyUML* na IDE NetBeans.

Figura 7 – Diagrama de Classes do Simulador Alterado



Fonte: Autoria própria (2018)

3.4.1 Eliminação de Classes

Com a implementação de uma nova classe que gera a matriz de tempos esperados, as classes **ETCGenerator**, **MachineHeterogeneity** e **TaskHeterogeneity** não precisam mais ser utilizadas, podendo ser eliminadas. As classes **MachineHeterogeneity** e **TaskHeterogeneity** são classes auxiliares para criação da matriz ETC, logo, se a matriz não é mais criada pela classe **ETCGenerator**, elas também deixaram de ser utilizadas.

4 Resultados

Para demonstrar que o simulador modificado se mantém coerente com o original, sendo apenas uma extensão daquele, é necessário realizar um conjunto de simulações e verificar se os dados se mantiveram consistentes. Assim, ambos os simuladores foram executados com os mesmos valores referentes a número de simulações, tarefas e máquinas. O número de simulações é 4000, o número de tarefas é 500 e o número de máquinas é 29.

4.1 Simulação Original

A simulação original foi executada 3 vezes, com os valores citados anteriormente. Os resultados obtidos em cada uma das simulações são apresentados nas Figuras 8, 9 e 10.

Como é possível observar nos resultados da execução do simulador, o algoritmo MET é o que demanda mais tempo para executar todas as tarefas, enquanto o Min-Mean ([AGRAWAL et al., 2010](#)) é o mais eficiente, possuindo uma média de tempo bem próxima dos algoritmos Min-Var, Min-Min e Sufferage.

Figura 8 – Primeira execução do simulador original

```
-----
Building distributed scheduling 1.0-SNAPSHOT
-----

--- exec-maven-plugin:1.2.1:exec (default-cli) @ distributed scheduling ---
Avg makespan for      MET heuristic for 4000 simulations is = 44185260
Avg makespan for      MCT heuristic for 4000 simulations is = 26208060
Avg makespan for      MaxMin heuristic for 4000 simulations is = 20605526
Avg makespan for      Sufferage heuristic for 4000 simulations is = 01428535
Avg makespan for      MinMin heuristic for 4000 simulations is = 01385310
Avg makespan for      MinMean heuristic for 4000 simulations is = 01377082
Avg makespan for      MinVar heuristic for 4000 simulations is = 01388351
Avg makespan for      NovelI heuristic for 4000 simulations is = 00000000
Total time taken in the simulation = 25 sec.

BUILD SUCCESS
-----
```

Fonte: Autoria própria (2018)

4.2 Simulação Modificada

A nova versão do simulador foi executada 3 vezes, com os valores citados anteriormente. Os resultados obtidos em cada uma das simulações são apresentados nas Figuras 11, 12 e 13.

Figura 9 – Segunda execução do simulador original

```

-----
Building distributed scheduling 1.0-SNAPSHOT
-----

--- exec-maven-plugin:1.2.1:exec (default-cli) @ distributed scheduling ---
Avg makespan for      MET heuristic for 4000 simulations is = 43814350
Avg makespan for      MCT heuristic for 4000 simulations is = 26337729
Avg makespan for      MaxMin heuristic for 4000 simulations is = 20592949
Avg makespan for      Sufferage heuristic for 4000 simulations is = 01428630
Avg makespan for      MinMin heuristic for 4000 simulations is = 01383715
Avg makespan for      MinMean heuristic for 4000 simulations is = 01376293
Avg makespan for      MinVar heuristic for 4000 simulations is = 01387814
Avg makespan for      NovellI heuristic for 4000 simulations is = 00000000
Total time taken in the simulation = 22 sec.

BUILD SUCCESS
-----

```

Fonte: Autoria própria (2018)

Figura 10 – Terceira execução do simulador original

```

-----
Building distributed scheduling 1.0-SNAPSHOT
-----

--- exec-maven-plugin:1.2.1:exec (default-cli) @ distributed scheduling ---
Avg makespan for      MET heuristic for 4000 simulations is = 43858551
Avg makespan for      MCT heuristic for 4000 simulations is = 26265115
Avg makespan for      MaxMin heuristic for 4000 simulations is = 20573666
Avg makespan for      Sufferage heuristic for 4000 simulations is = 01430115
Avg makespan for      MinMin heuristic for 4000 simulations is = 01383737
Avg makespan for      MinMean heuristic for 4000 simulations is = 01378435
Avg makespan for      MinVar heuristic for 4000 simulations is = 01388071
Avg makespan for      NovellI heuristic for 4000 simulations is = 00000000
Total time taken in the simulation = 22 sec.

BUILD SUCCESS
-----

```

Fonte: Autoria própria (2018)

Os resultados da execução do simulador alterado são consistentes com os resultados encontrados na execução do simulador original. Nos resultados obtidos pela execução, nota-se que o algoritmo menos eficiente continua sendo o MET, enquanto o algoritmo mais eficiente continua sendo o Min-Mean.

Como os dados da matriz ETC são gerados aleatoriamente, não é possível simular na nova matriz de tempos esperados com os mesmos dados existentes na matriz ETC. Dessa forma, alguns algoritmos possuem valores de média e desvio padrão muito próximos, porém, mantendo a ordem de resultados anteriormente exibidas para a simulação original.

Além de consistentes, nota-se que as novas simulações trazem o valor do desvio padrão. Assim, conclui-se que o simulador modificado cumpre com os objetivos deste trabalho de conclusão de curso.

Figura 11 – Primeira execução do simulador modificado

```

-----
Building distributed scheduling 1.0-SNAPSHOT
-----

--- exec-maven-plugin:1.2.1:exec (default-cli) @ distributed scheduling ---
Avg makespan for      MET heuristic for 4000 simulations is = 06211856
Desvio padrão para    MET para 4000 simulações é = 01303573
Avg makespan for      MCT heuristic for 4000 simulations is = 03600057
Desvio padrão para    MCT para 4000 simulações é = 00222330
Avg makespan for      MaxMin heuristic for 4000 simulations is = 02574754
Desvio padrão para    MaxMin para 4000 simulações é = 00249850
Avg makespan for      Sufferage heuristic for 4000 simulations is = 00944342
Desvio padrão para    Sufferage para 4000 simulações é = 00012096
Avg makespan for      MinMin heuristic for 4000 simulations is = 00866133
Desvio padrão para    MinMin para 4000 simulações é = 00010146
Avg makespan for      MinMean heuristic for 4000 simulations is = 00866003
Desvio padrão para    MinMean para 4000 simulações é = 00010159
Avg makespan for      MinVar heuristic for 4000 simulations is = 00866162
Desvio padrão para    MinVar para 4000 simulações é = 00010107
Avg makespan for      NovelI heuristic for 4000 simulations is = 00000000
Desvio padrão para    NovelI para 4000 simulações é = 00000000
Total time taken in the simulation = 25 sec.

BUILD SUCCESS
-----

```

Fonte: Autoria própria (2018)

Figura 12 – Segunda execução do simulador modificado

```

-----
Building distributed scheduling 1.0-SNAPSHOT
-----

--- exec-maven-plugin:1.2.1:exec (default-cli) @ distributed scheduling ---
Avg makespan for      MET heuristic for 4000 simulations is = 06263555
Desvio padrão para    MET para 4000 simulações é = 01340042
Avg makespan for      MCT heuristic for 4000 simulations is = 03599045
Desvio padrão para    MCT para 4000 simulações é = 00222819
Avg makespan for      MaxMin heuristic for 4000 simulations is = 02574198
Desvio padrão para    MaxMin para 4000 simulações é = 00245881
Avg makespan for      Sufferage heuristic for 4000 simulations is = 00944166
Desvio padrão para    Sufferage para 4000 simulações é = 00011622
Avg makespan for      MinMin heuristic for 4000 simulations is = 00865921
Desvio padrão para    MinMin para 4000 simulações é = 00009898
Avg makespan for      MinMean heuristic for 4000 simulations is = 00865665
Desvio padrão para    MinMean para 4000 simulações é = 00009967
Avg makespan for      MinVar heuristic for 4000 simulations is = 00865681
Desvio padrão para    MinVar para 4000 simulações é = 00009999
Avg makespan for      NovelI heuristic for 4000 simulations is = 00000000
Desvio padrão para    NovelI para 4000 simulações é = 00000000
Total time taken in the simulation = 24 sec.

BUILD SUCCESS
-----

```

Fonte: Autoria própria (2018)

Figura 13 – Terceira execução do simulador modificado

```
-----
Building distributed scheduling 1.0-SNAPSHOT
-----

--- exec-maven-plugin:1.2.1:exec (default-cli) @ distributed scheduling ---
Avg makespan for      MET heuristic for 4000 simulations is = 06249391
Desvio padrão para    MET para 4000 simulações é = 01341094
Avg makespan for      MCT heuristic for 4000 simulations is = 03602940
Desvio padrão para    MCT para 4000 simulações é = 00220018
Avg makespan for      MaxMin heuristic for 4000 simulations is = 02571779
Desvio padrão para    MaxMin para 4000 simulações é = 00245883
Avg makespan for      Sufferage heuristic for 4000 simulations is = 00944288
Desvio padrão para    Sufferage para 4000 simulações é = 00011824
Avg makespan for      MinMin heuristic for 4000 simulations is = 00866082
Desvio padrão para    MinMin para 4000 simulações é = 00010172
Avg makespan for      MinMean heuristic for 4000 simulations is = 00865972
Desvio padrão para    MinMean para 4000 simulações é = 00010080
Avg makespan for      MinVar heuristic for 4000 simulations is = 00865998
Desvio padrão para    MinVar para 4000 simulações é = 00010078
Avg makespan for      NovelI heuristic for 4000 simulations is = 00000000
Desvio padrão para    NovelI para 4000 simulações é = 00000000
Total time taken in the simulation = 25 sec.
-----
BUILD SUCCESS
-----
```

Fonte: Autoria própria (2018)

5 Conclusão

Este trabalho tem como objetivo principal realizar modificações no simulador de algoritmos de escalonamento, proposto por [Kamalam e Muralibhaskaran \(2010\)](#), de forma que o mesmo seja estendido para apresentar mais informações e detalhes estatísticos para análises mais precisas, como o cálculo do desvio padrão.

Os desafios desse trabalho se referem à construção de uma matriz que possua uma estrutura parecida com a matriz ETC usada no simulador original e, além do armazenamento dos dados para calcular o desvio padrão de forma correta. Como a matriz ETC era fundamentada nas classes **TaskHeterogeneity** e **MachineHeterogeneity**, a nova classe criada deveria possuir dados que se adequassem a essa estrutura, mesmo não utilizando essas classes, para manter o processo de simulação da forma original. Já o desvio padrão é uma fórmula que precisa de todos os valores do conjunto para conseguir realizar o cálculo de forma adequada, logo, pensar em uma estrutura que melhor se adapte ao programa para armazenar todos esses dados foi o outro desafio.

Os objetivos e desafios desse trabalho foram concluídos com sucesso. O simulador alterado possui informação do desvio padrão, que não era calculado anteriormente. Assim como permite a definição e alteração de informações da capacidade da máquina e quantidade de instruções cada tarefa requer. Dessa forma, o simulador alterado possui uma gama maior de dados a serem consultados, possibilitando uma análise maior.

5.1 Trabalhos Futuros

Para trabalhos futuros, recomenda-se implementar no simulador mais funções estatísticas para que a análise de resultados fique ainda mais detalhada e precisa, com a finalidade de facilitar e melhorar as análises dos algoritmos.

No quesito de melhorias, um bom tópico é centralizar e padronizar a inserção de dados em um só lugar, possibilitando informá-los em alguma tela, e não somente realizar essas alterações no código, em suas próprias classes.

Há também a possibilidade de estender o número de heurísticas implementadas no simulador, podendo acrescentar novas heurísticas e aumentar a quantidade de algoritmos a serem analisados, para ter um detalhamento mais preciso sobre qual a melhor heurística a ser utilizada diante dos dados informados.

Referências

- AGRAWAL, K.; BENOIT, A.; MAGNAN, L.; ROBERT, Y. Scheduling algorithms for linear workflow optimization. *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010*, 2010. ISSN 1530-2075. Citado na página 20.
- BRAUNT, T. D.; SIEGEL, H. J.; BECK, N.; BOLONI, L. L.; MAHESWARANS, M. A comparison study of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of parallel and distributed computing*, v. 22, p. 810–837, 2001. Citado 2 vezes nas páginas 8 e 11.
- BUSSAB, W. O.; MORETTIN, P. A. *Estatística Básica*. 9. ed. [S.l.: s.n.], 2017. 576 p. Citado na página 10.
- CASANOVA, H.; GIERSCHE, A.; LEGRAND, A.; QUINSON, M.; SUTER, F. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, Elsevier, v. 74, n. 10, p. 2899–2917, jun. 2014. Citado na página 5.
- GUPTA, J. N. D.; HO, J. C.; STATE, C. Minimizing flowtime subject to optimal makespan on two identical parallel machines. *Pesquisa Operacional*, v. 20, n. 1, p. 5–18, 2000. ISSN 0101-7438. Citado na página 7.
- IBARRA, O. H.; KIM, C. E. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, v. 24, n. 2, p. 280–289, 1977. Citado na página 8.
- KAMALAM, G. K.; MURALIBHASKARAN, V. A new heuristic approach: Min-mean algorithm for scheduling meta-tasks on heterogeneous computing systems. *IJCSNS International Journal of Computer Science and Network Security*, v. 10, n. 1, 2010. Citado 7 vezes nas páginas 2, 5, 6, 8, 9, 10 e 24.
- KLUSÁČEK, D.; RUDOVÁ, H. Alea 2: Job scheduling simulator. In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. [S.l.]: ICST, 2010. Citado na página 5.
- KOŁODZIEJ, J. *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. v. 419. 1–219 p. (Studies in Computational Intelligence, v. 419). ISSN 1860949X. ISBN 978-3-642-28970-5. Disponível em: <<http://link.springer.com/10.1007/978-3-642-28971-2>>. Citado 3 vezes nas páginas 5, 10 e 11.
- MAHESWARAN, M.; ALI, S.; SIEGEL, H. J.; HENSGEN, D.; FREUND, R. F. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, v. 59, n. 2, p. 107–131, 1999. ISSN 07437315. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0743731599915812>>. Citado 5 vezes nas páginas 2, 5, 6, 8 e 9.

STOCKINGER, H. Defining the grid: A snapshot on the current view. *Journal of Supercomputing*, v. 42, n. 1, p. 3–17, 2007. ISSN 09208542. Citado na página 7.

XHAFA, F.; ABRAHAM, A. A Compendium of Heuristic Methods for Scheduling in Computational Grids. In: CORCHADO, E.; YIN, H. (Ed.). *Intelligent Data Engineering and Automated Learning - IDEAL 2009*. [s.n.], 2009. v. 5788 LNCS, p. 751–758. ISBN 3642043933. Disponível em: <http://link.springer.com/10.1007/978-3-642-04394-9_92>. Citado na página 7.

_____. Computational models and heuristic methods for Grid scheduling problems. *Future Generation Computer Systems*, Elsevier B.V., v. 26, n. 4, p. 608–621, 4 2010. ISSN 0167739X. Disponível em: <<http://dx.doi.org/10.1016/j.future.2009.11.005><http://linkinghub.elsevier.com/retrieve/pii/S0167739X09001782>>. Citado 2 vezes nas páginas 5 e 6.

XHAFA, F.; BAROLLI, L.; DURRESI, A. An Experimental Study on Genetic Algorithms for Resource Allocation on Grid Systems. *Journal of Interconnection Networks*, v. 8, n. 4, p. 427–443, 2007. Citado na página 7.

_____. Batch mode scheduling in grid systems. *International Journal of Web and Grid Services*, v. 3, n. 1, p. 19, 2007. ISSN 1741-1106. Disponível em: <<http://www.inderscience.com/link.php?id=12635>>. Citado na página 7.