UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Johnata Ferreira Santos

Algoritmos Online para Escalonamento de Tarefas em Sistemas Multiprocessados

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Johnata Ferreira Santos

Algoritmos Online para Escalonamento de Tarefas em Sistemas Multiprocessados

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

Johnata Ferreira Santos

Algoritmos Online para Escalonamento de Tarefas em Sistemas Multiprocessados

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.

Trabalho aprovado. Uberlândia, Brasil, 11 de julho de 2018:

Paulo Henrique Ribeiro Gabriel
Orientador

Professor

Professor

Uberlândia, Brasil
2018

Agradecimentos

Primeiramente agradeço a Deus por finalmente estar conseguindo a obtenção do grau em Bacharel em Sistemas de Informação, profissão essa que eu escolhi para a vida inteira. Não foi fácil e muito menos rápido, porém a trajetória foi esplêndida. Agradeço aos meus pais, Mario e Maria, por me incentivarem a estudar, hoje reconheço que é algo fundamental. Agradeço ao meu irmão, Leandro, pelos conselhos, haja vista que ele obteve grau antes de mim. Agradeço a minha noiva, Ana Paula, pela paciência que ela demonstrou quando tive que deixá-la de lado em alguns momentos para estudar e realizar trabalhos. Agradeço ao meu orientador, Paulo Henrique, pelos ensinamentos, pelos conselhos, pela paciência e pelo auxílio, foi fundamental para realização deste trabalho. Por fim agradeço aos meus amigos de curso por essa grande experiência vivida em conjunto que foi a conclusão de um excelente curso.

Resumo

O escalonamento de processos é uma atividade de grande importância para garantir que os sistemas computacionais trabalhem de forma otimizada. Diversos algoritmos foram estabelecidos para resolver este problema de maneira a realizar uma distribuição equitativa das tarefas entre as máquinas disponíveis para executá-las. Existem diversas métricas de avaliação que auxiliam a determinar se a solução proposta por esses algoritmos podem ser aplicadas ao escopo do problema analisado, dentre as quais podemos citar o *Makespan* e a utilização média. Este trabalho possui o objetivo de implementar e avaliar três algoritmos heurísticos para solucionar o problema do escalonamento de processos: *List-scheduling*, *Random* e *Round-robin*. Resultados mostraram que o *List-scheduling* apresentou a melhor solução para este problema enquanto que o *Random* apresentou a pior solução. Um detalhe importante observado foi que os algoritmos *List-scheduling* e *Round-robin* produzem o mesmo resultado com tarefas ordenadas por sua carga em ordem crescente.

Palavras-chave: Escalonamento de Processos, Otimização Online, Makespan, Algoritmos Heurísticos.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CPU	*Central Process Unit* (Unidade Central de Processamento)
T	Conjunto de tarefas
P	Conjunto de processos ou processadores
C	Conjunto de critérios de otimização ou *completion time* (tempo gasto pelo computador para executar uma tarefa)
r	Sequência de solicitação finita da entrada na otimização online
ALG	Um algoritmo online
RALG	Um algoritmo online aleatório (“randomizado”)
t	Tempo de liberação da tarefa no algoritmo online que utiliza o modelo *timestamp*
V	Conjunto de computadores
l	Carga de um processo
TCC	Trabalho de Conclusão de Curso
ID	Identificador das máquinas que executam as tarefas
LS	Algoritmo List-scheduling
RND	Algoritmo Random
RR	Algoritmo Round-robin
um	Utilização média
cm	Carga média
IOT	*Internet of Things* (Internet das Coisas)
ITU	*International Telecommunication Union* (União Internacional de Telecomunicações)

Sumário

1 Introdução

1.1 Contextualização

Aplicações de alto desempenho, bem como a demanda por escalabilidade, têm motivado o desenvolvimento de sistemas computacionais multiprocessados (paralelos e/ou distribuídos). Nesses ambientes, um conjunto de tarefas, oriundas de aplicações paralelas, devem ser atribuídas a um conjunto de máquinas (processadores), objetivando a redução do tempo de execução. Essa atribuição recebe o nome de escalonamento de tarefas (PRUHS; TORNG; SGALL, 2004).

Uma variação bastante estudada deste problema é o chamado *escalonamento online*, no qual as tarefas a serem executadas não são conhecidas antecipadamente, ou seja, a escolha das máquinas que as receberão deve ocorrer apenas no momento em que a tarefa chega ao sistema (KRUMKE; THIELEN, 2014). Nesse contexto, diversos algoritmos têm sido propostos (GRAHAM, 1966; BARTAL *et al.*, 1995a; KARGER; PHILLIPS; TORNG, 1996; ALBERS, 1997; FLEISCHER; WAHL, 2000). Devido à desvantagem da entrada não ser conhecida a priori, os algoritmos existentes realizam cálculos na busca pelas melhores decisões. Em geral, soluções ótimas são quase impossíveis de serem atingidas devido ao fato de não termos o conhecimento de todos os processos a serem executados, daí a necessidade da busca por soluções mais próximas do ótimo possível.

Existem diversos fatores que influenciam na escolha do algoritmo adequado: os modelos de máquinas (se são idênticas ou não, se são relacionadas ou não), os formatos de processamento (se existe programação preemptiva ou não) e os tipos de função objetivo (*Makespan*, soma dos tempos de conclusão, soma dos tempos de fluxo). Cada problema possui características específicas inerentes ao contexto em que eles se encontram, portanto conhecer bem esses fatores é extremamente importante para definir qual solução será a mais adequada.

Neste trabalho de conclusão de curso, o problema de escalonamento considerado possui as seguintes características:

- Nenhuma informação sobre as tarefas (nem menos sua quantidade) é conhecida antecipadamente;
- As máquinas são idênticas, ou seja, o tempo de execução de uma tarefa não varia de uma máquina para a outra;
- Uma máquina processa apenas uma tarefa por vez;

- Uma tarefa pode ser executada somente em uma máquina por vez.

Algoritmos de escalonamento online de tarefas têm sido investigados desde a década de 1960, porém um estudo mais aprofundado dos mesmos teve inicio apenas na década de 1980, com o desenvolvimento formal do conceito de *análise competitiva* (ROBERT; VIVIEN, 2009). Um dos algoritmos online mais conhecido e utilizado para minimizar o tempo máximo de execução das tarefas (*makespan*) em máquinas paralelas idênticas é o *List-scheduling* proposto por Graham (1966). Até a década de 1990, esse algoritmo era considerado o mais competitivo, ou seja, aquele que apresenta maior aproximação em relação à solução ótima do problema (ALBERS, 2003).

Contudo, novos algoritmos foram desenvolvidos posteriormente e alcançaram razões competitivas diferentes (BARTAL *et al.*, 1995a; KARGER; PHILLIPS; TORNG, 1996; ALBERS, 1997; FLEISCHER; WAHL, 2000). Tais algoritmos apresentam detalhadas provas matemáticas sobre sua eficiência e efetividade, no entanto, poucos trabalhos os avaliam em termos práticos. De fato, um dos poucos estudos experimentais foi realizado por Albers e Schröder (2002), para identificar sob diferentes cargas de trabalho se os novos algoritmos são realmente melhores na prática, ou apenas teóricos. Nas cargas de trabalho geradas por distribuição de trabalho padrão, o algoritmo de Graham (1966) é claramente melhor; no entanto, nos empregos do mundo real, os novos algoritmos superam essa estratégia (ALBERS; SCHRÖDER, 2002).

1.2 Motivação

Os trabalhos correlatos apresentam provas matemáticas consistentes sobre os limites dos algoritmos apresentados. Poucos, porém, focam em detalhes de implementação e ambientes de execução reais (ALBERS, 2003). Por esse motivo, neste trabalho de conclusão de curso é feito o estudo e a implementação de alguns desses algoritmos. Diferente do trabalho de Albers e Schröder (2002), são avaliadas outras métricas de desempenho, além do *makespan*. Dessa maneira, busca-se observar melhor o comportamento de tais algoritmos, bem como motivar o desenvolvimento de novos métodos.

1.3 Objetivos

Este trabalho tem como objetivo a implementação de alguns algoritmos clássicos de escalonamento online, além de comparar seus resultados sob determinadas métricas de desempenho. Três dos algoritmos estudados são o *List-scheduling* (GRAHAM, 1966), o *Random* (BARTAL *et al.*, 1995a) e o *Round-robin* (RASMUSSEN; TRICK, 2008), os quais são frequentemente utilizados como base no desenvolvimento de outros algoritmos.

A implementação dos algoritmos permite a realização de diferentes simulações. Nessas simulações, são gerados conjuntos de tarefas utilizando diferentes distribuições de probabilidade; essas tarefas serão submetidas aos algoritmos, analisando-se o desempenho de cada um. Assim, é possível identificar qual possui melhor desempenho e, ainda, verificar se os resultados corroboram ou não as provas matemáticas existentes para cada algoritmo.

1.4 Organização

O restante deste trabalho de conclusão de curso está organizado como descrito a seguir. O Capítulo 2 apresenta os principais conceitos envolvidos neste trabalho, ou seja: escalonamento de processos e otimização online. O Capítulo 3 apresenta trabalhos relacionados a esta pesquisa. No Capítulo 4 é descrito o desenvolvimento dos algoritmos utilizados bem como as métricas de avaliação e o ambiente utilizado nos experimentos. Os resultados obtidos de cada algoritmo são apresentados no capítulo 5. Finalmente, são apresentadas as conclusões e descritos possíveis trabalhos futuros no Capítulo 6.

2 Conceitos Adotados

A fim de cumprir com o objetivo deste trabalho, deve-se utilizar conceitos de: *i*) escalonamento de processos, definido como o ato de realizar o mapeamento de processos ativos, de acordo com regras bem estabelecidas, de forma que todos os processos possam ser executados; e *ii*) escalonamento online, que é uma classe de problemas dentro da qual as decisões sobre o escalonamento devem ser tomadas sem o conhecimento das tarefas a serem executadas. Os principais conceitos dessas áreas são apresentados, resumidamente, neste capítulo.

2.1 Escalonamento de Processos

Conceitualmente, escalonamento de processos consiste na alocação eficiente de aplicações paralelas sobre os recursos disponíveis em um sistema distribuído (CASAVANT; KUHL, 1988). Algoritmos de escalonamento têm sido projetados levando-se em consideração os requisitos individuais de processos, como o número de instruções e o volume de dados transferido, e recursos, como capacidade de processamento e largura de banda. Mais recentemente, pesquisadores têm dado especial atenção à heterogeneidade dos recursos, devido ao crescimento no número de sistemas distribuídos que mantêm parte de seus recursos legados (STOCKINGER, 2007).

Todas essas variáveis influenciam na complexidade do escalonamento de processos. De fato, pode-se demonstrar que o problema de encontrar um escalonamento ótimo é NP-Completo para dois ou mais computadores idênticos mesmo na ausência de comunicação entre processos (GAREY; JOHNSON, 1979).

O escalonamento de processos discute o mapeamento de tarefas nos diferentes recursos computacionais em sistemas paralelos e distribuídos. Esse mapeamento é parte de uma classe mais ampla de problemas de alocação de recursos e, provavelmente, um dos mais cuidadosamente estudados. 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, ser executado mais rapidamente do que em um único processador. O problema do escalonamento de processos em sistemas multiprocessados pode ser, geralmente, descrito da seguinte maneira (CHAPIN; WEISSMAN, 2002): Como 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 espe-

rado de um conjunto de tarefas. Exemplos de outros critérios de escalonamento incluem a minimização do custo, minimização do 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 combinação de alguns desses critérios (CHAPIN; WEISSMAN, 2002).

2.2 Escalonamento Online

Devido à complexidade do problema de escalonamento, diversos estudos têm sido propostos em busca da melhor solução, seja ela em questão de qualidade ou eficiência. Esses estudos produziram algoritmos capazes de proporcionar resultados satisfatórios se aplicados no contexto correto.

Esses algoritmos foram projetados para considerar características de processos e/ou recursos. Tais características são, em geral, conhecidas a priori ou estimadas ao logo da execução do algoritmo. No entanto, muitos cenários do mundo-real têm que lidar com o aspecto online do problema de escalonamento. Nesse caso, processos chegam ao sistema em sequência ao longo do tempo e as decisões de escalonamento devem ser tomadas sem conhecimento prévio sobre o comportamento de processos futuros. Consequentemente, os algoritmos de escalonamento possuem apenas um conhecimento parcial sobre a entrada do problema; assim, suas decisões são baseadas apenas no estado atual do sistema (PRUHS; TORNG; SGALL, 2004).

Diversos problemas do mundo real necessitam de algoritmos online para resolvê-los. Em tese este tipo de algoritmo não atinge a solução ótima, porém eles podem se aproximar bastante dela. Segue abaixo alguns problemas do mundo real que podem ser resolvidos com este tipo de algoritmo:

- Paginação de memória virtual;
- Roteamento em redes;
- Balanceamento de carga;
- Seleção de links patrocinados.

Na otimização online, a entrada do problema é modelada como uma sequência de solicitação finita $r_1, r_2, \ldots$ que deve ser atendida e que é revelada passo a passo para um algoritmo online. Como isso é feito, depende do paradigma online especificado. Os dois modelos mais comuns são o modelo de sequência e o modelo de *timestamp*. Na sequência estão definidos resumidamente os dois modelos.

Seja ALG um algoritmo online. No modelo de sequência, os pedidos do modelo devem ser atendidos na ordem de sua ocorrência. Mais precisamente, ao atender um pedido r_j, o algoritmo online ALG não tem conhecimento de pedidos r_i com $i > j$ (ou mesmo em relação ao número total de pedidos). Quando o pedido r_j é apresentado deve ser atendido pelo ALG de acordo com as regras específicas do problema. O serviço de r_j incorre num "custo" e o objetivo geral é minimizar o custo total do serviço. A decisão do ALG de como servir r_j é irrevogável. Somente depois que o r_j for atendido, o próximo pedido r_{j+1} é conhecido para ALG.

Já no modelo *timestamp*, cada solicitação tem um horário de chegada ou liberação no qual ela fica disponível para o serviço. O tempo de liberação $t_j \geq 0$ é um número real não negativo e que especifica o momento em que a solicitação r_j é liberada (torna--se conhecido por um algoritmo online). Um algoritmo online ALG deve determinar seu comportamento em um certo momento t em tempo como uma função de todos os pedidos liberados até o momento $t-1$ e do tempo atual t. Mais uma vez, estamos na situação que um algoritmo online ALG é confrontado com uma sequência finita $r_1, r_2, \ldots$ de pedidos que são dados em ordem de tempos de liberação não decrescentes e o serviço de cada pedido incorre em um custo para ALG.

A diferença do modelo *timestamp* para o modelo de sequência é que o algoritmo online pode aguardar e revogar decisões e esses pedidos não precisam ser atendidos na ordem de sua ocorrência (KRUMKE, 2001).

3 Trabalhos Relacionados

Conforme já mencionado, o escalonamento de processos consiste em mapear os elementos de um conjunto de processos P sobre um conjunto de computadores V sujeitos a um conjunto de critérios de otimização (GAREY; JOHNSON, 1979; FEITELSON *et al.*, 1997). Assim que um processo p_i é atribuído a um computador v_j, v_j tem sua carga (denotada por ℓ_j) incrementada e o processo pode ser executado. Além disso, cada computador não pode executar mais de um processo ao mesmo tempo. O total de tempo gasto por v_j para executar sua carga é denominado *completion time* (FEITELSON *et al.*, 1997), denotado por C_j. O *completion time* máximo, $C_{\max}$, é chamado *makespan* (FEITELSON *et al.*, 1997) e determina o tempo esperado de execução do conjunto de processos.

Graham (1966) publicou um trabalho seminal que apresenta um algoritmo guloso denominado *List-scheduling* (LS). Um algoritmo guloso toma decisões com base nas informações da iteração corrente, sem se preocupar com as consequências dessa escolha no futuro. As decisões tomadas são definitivas, ou seja, essas escolhas não são desfeitas. Esse algoritmo considera a carga atual dos computadores e escalona cada novo processo sobre o computador de menos carga. Pode-se demonstrar que o LS é uma $(2 - \frac{1}{m})$-aproximação para m computadores idênticos (ou seja, um ambiente homogêneo). Isso significa que o LS produz soluções cujo *makespan* é, no máximo, $(2 - \frac{1}{m})$ vezes maior que o ótimo. Além disso, esse algoritmo não considera informações sobre o conjunto de processos; consequentemente, LS é também considerado um algoritmo $(2 - \frac{1}{m})$-competitivo (CHEN; POTTS; WOEGINGER, 1998).

Essa competitividade foi melhorada em diversos trabalhos (BARTAL *et al.*, 1995b; KARGER; PHILLIPS; TORNG, 1996; ALBERS, 1997; FLEISCHER; WAHL, 2000). O melhor resultado teórico conhecido é atribuído a Fleischer e Wahl (2000), que propuseram o algoritmo IMBAL, cuja competitividade é aproximadamente igual a $1,9201$.

O objetivo do IMBAL é manter um escalonamento em que $l \approx 0,36m + 1$ e computadores estão com pouca carga e $m - l$ estão com alta carga. O escalonamento ocorre da seguinte maneira: seja A_l^{t-1} a carga média dos l computadores menos carregados, no instante de tempo $t - 1$; no instante t, um processo p_t é escalonado no $l + 1$-ésimo computador menos carregado se $A_l^{t-1} > \alpha \ell_{2l-1}^{t-1}$, sendo $\alpha \approx 0.46$ e $\ell_{2l-1}^{t-1} + p_t < cL$, em que L é o limitante inferior estimado do *makespan* ótimo. Caso contrário, p_t é alocado sobre o computador menos carregado do sistema Fleischer e Wahl (2000).

Nesse mesmo contexto, Blake (1992) descreve alguns algoritmos heurísticos subótimos. Dentre eles está o escalonamento aleatório (*Random*), em que um processador é selecionado aleatoriamente e é forçado a executar determinada tarefa.

Um algoritmo online aleatório $RALG$ é uma distribuição de probabilidade sobre um conjunto $ALG(y)$ de algoritmos online determinísticos (y pode ser considerado como o lançamento de moeda do algoritmo *Random*). A saída $RALG(\sigma)$ e o custo $RALG(\sigma)$ em uma dada entrada σ são variáveis aleatórias.

Uma terceira técnica de escalonamento bastante conhecida e utilizada é através do algoritmo *Round-robin*. É um dos algoritmos de escalonamento mais simples e antigo, porém ainda é bastante utilizado. Foi projetado inicialmente para sistemas mono-processados do tipo *time-sharing*, onde o tempo ocioso entre os processos é compartilhado com outros processos para aumentar a eficiência do sistema.

Segundo Arruda (2001) a ideia do algoritmo é definir uma pequena unidade de tempo, denominada *timeslice* ou *quantum*. Todos os processos são armazenados em uma fila circular. O escalonador da CPU percorre a fila alocando a CPU para cada processo durante um *quantum*. O escalonador retira o primeiro processo da fila e dá início à sua execução. Se o processo não terminar durante um quantum, ocorre uma preempção e o processo é inserido de volta ao fim da fila. Se o processo terminar antes do fim de um *quantum*, a CPU é liberada para que os demais processos da fila possam ser executados. Em ambos os casos, após a liberação da CPU, um novo processo é escolhido na fila. Novos processos são sempre inseridos ao fim da fila. A Figura 1 demonstra toda esta estrutura.

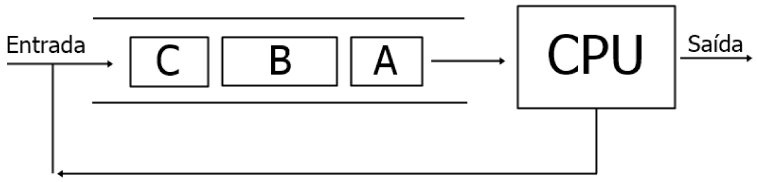


Figura 1 – Estrutura do Round-robin

Quando adaptado a sistemas multiprocessados, o *Round-robin* considera uma fila circular de processadores (ou seja, as diferentes CPUs). Cada processo que chega ao sistema é, então, atribuído a um dos processadores, em sequência; quando todos os processadores já foram considerados, o próximo processo é atribuído ao primeiro processador, novamente, e a fila volta a ser percorrida.

4 Desenvolvimento

Neste trabalho foram implementados três algoritmos de otimização online: *List-scheduling*, *Random* e *Round-robin*. Neste capítulo é descrito em que tipo de ambiente os experimentos foram realizados, como se deu a implementação destes algoritmos e quais as métricas de avaliação utilizadas.

4.1 O Ambiente dos Experimentos

Foi desenvolvido um programa em *Java* para simular a execução dos algoritmos. As cargas das tarefas utilizadas serão geradas de forma aleatória por meio de uma distribuição uniforme. Para isso o programa utiliza o método estático `Random` da classe `Math` do ambiente *Java*. Esse método gera um valor aleatório no intervalo $[0, 1)$. Neste trabalho, as cargas têm um valor inteiro entre 100 e 1000.

Para execução do programa, foram geradas um total de 128 tarefas a serem atribuídas a cada um dos algoritmos. A simulação é feita considerando alguns tipos de ambientes distintos.

O primeiro critério para execução do programa a ser considerado é a quantidade de máquinas disponíveis para as tarefas. O programa é executado com cada algoritmo considerando 8 máquinas na primeira execução, 16 máquinas na segunda e, por fim, 32 máquinas na última execução, todas elas idênticas.

Dentro desse primeiro critério, há uma segunda forma de execução dos algoritmos: a ordenação das cargas das tarefas. Cada algoritmo é executado considerando três tipos de ordenação:

1. Sem ordenação;
2. Ordenação crescente;
3. Ordenação decrescente.

Cada um desses cenários produz um resultado, que poderá ou não, sofrer variação de um ambiente para o outro. O objetivo é analisar se a lógica proposta pelo algoritmo produzirá resultados satisfatórios, e se isso vai depender da ordem em que os processos chegam ao sistema.

4.2 O Simulador

O simulador utilizado neste trabalho foi implementado utilizando a linguagem de programação *Java*. A ideia principal foi criar uma estrutura para simular a utilização dos algoritmos. À medida em que as tarefas chegam no sistema, cada algoritmo, de acordo com sua regra, irá definir em qual máquina essa tarefa deverá ser executada. A cada nova tarefa que chega, o algoritmo irá designá-la à próxima máquina até que todas as tarefas tenham sido concluídas. Ao fim da execução, os resultados serão apresentados.

Inicialmente, o programa gera um vetor com as cargas de todas as tarefas a serem executadas. Essas cargas são geradas de forma aleatória. Com o vetor preenchido, o programa percorre um *loop* em que cada iteração representa a quantidade de máquinas disponíveis para execução das tarefas.

Existe uma ordem de execução dos cenários a serem executados e essa ordem é seguida em todas as iterações. Na primeira iteração, o programa executa os algoritmos considerando 8 máquinas.

Na sequência, os algoritmos serão executados considerando a quantidade de máquinas e a ordenação das tarefas. Cada algoritmo utiliza sua lógica de implementação para designar as tarefas às máquinas que irão executá-las. Ao fim da execução de cada algoritmo, o programa exibe os resultados na tela considerando as métricas de desempenho utilizadas neste TCC.

Ao final dessa iteração o programa repete os passos citados anteriormente, considerando 16 máquinas e, em seguida, 32 máquinas, finalizando assim o ciclo de execução do programa.

4.2.1 List-scheduling

Para a implementação do algoritmo *List-scheduling* foi utilizada a classe *Java* `PriorityQueue` (Fila de Prioridade). De acordo com a própria documentação da linguagem, uma fila de prioridade ilimitada é baseada em uma árvore *heap*. Os elementos da fila de prioridade são ordenados de acordo com sua ordenação natural ou por um comparador fornecido no tempo de construção da fila, dependendo de qual construtor é utilizado.

No caso da implementação deste trabalho, o comparador é a carga da tarefa e o ID da máquina. A lógica do *List-scheduling* é que a próxima tarefa da fila seja executada na primeira máquina que estiver livre. O Algoritmo 1 ilustra, em pseudocódigo, o funcionamento da implementação do *List-scheduling*.

Algumas funções extras foram utilizadas para auxiliar na implementação do algoritmo. A seguir, são detalhadas as mais importantes:

Algoritmo 1 Pseudocódigo do *List-scheduling*

```
Entrada: qtdTarefas, qtdMaquinas
Saída: Vetor de inteiros com o índice sendo o ID da tarefa e o valor sendo
       o ID da máquina que a executará
início
    vt = inicializarVetorTarefas();
    para i = {8, 16, 32} faça
        vet = inicializarVetorExecucao(qtdTarefas);
        vct = inicializarVetorCargaTotalPorMaquina(qtdMaquinas);
        fp = inicializarFilaPrioridade();
        para j = 0 até qtdTarefas faça
            carga = vt[i]; //Passo 1
            par = fp.poll(); //Passo 2
            vct[par.ID] = par.CARGA + carga; //Passo 3
            vet[i] = par.ID; //Passo 4
            par = criaPar(vct[par.ID], par.ID); //Passo 5
            fp.add(par); //Passo 6
            j++;
        fim
        i = i * 2;
    fim
fim
retorna vet
```

- inicializarVetorTarefas: função utilizada para criar um vetor com as tarefas a serem processadas;
- inicializarVetorExecucao: função utilizada para criar um vetor de tamanho igual a quantidade de tarefas. O valor de cada posição será o ID da máquina no qual a tarefa será executada;
- inicializarVetorCargaTotalPorMaquina: função utilizada para criar um vetor de tamanho igual a quantidade de máquinas. Cada posição será o ID da máquina e o valor será a carga total executada por ela;
- inicializarFilaPrioridade: função utilizada para criar uma fila de prioridade de um objeto Par<int, int>. A posição 1 será a carga de execução da máquina e a posição 2 será o ID da máquina;
- Passo 1: pegando a carga da tarefa;
- Passo 2: pegando o primeiro elemento da fila e removendo-o;
- Passo 3: inserindo a carga na máquina;
- Passo 4: marcando em qual máquina a tarefa "i" será executada;

- Passo 5: cria-se um novo objeto "par" a ser inserido na fila com sua carga alterada;
- Passo 6: adiciona o objeto na fila.

4.2.2 Random

A implementação do algoritmo *Random* utilizou uma distribuição uniforme de números aleatórios (essa mesma distribuição foi utilizada para gerar as cargas aleatórias das tarefas deste simulador). Para isso, utilizou-se o método estático `random` da classe `Math` da linguagem *Java*, que determina em qual máquina a próxima tarefa será executada. O funcionamento do *Random* está ilustrado, como pseudocódigo, no Algoritmo 2.

Algoritmo 2 Pseudocódigo do *Random*.

```
Entrada: qtdTarefas, qtdMaquinas
Saída: Vetor de inteiros com o índice sendo o ID da tarefa e o valor sendo
       o ID da máquina que a executará
início
    vt = inicializarVetorTarefas();
    para i = {8, 16, 32} faça
        vet = inicializarVetorExecucao(qtdTarefas);
        para j = 0 até qtdTarefas faça
            idMaquina = getMaquinaAleatoria(qtdMaquina); //Passo 1
            vet[i] = idMaquina; //Passo 2
            j++;
        fim
        i = i * 2;
    fim
fim
retorna vet
```

Assim como o *List-scheduling*, o *Random* também fez o uso de algumas funções extras, detalhadas a seguir:

- inicializarVetorTarefas: função utilizada para criar um vetor com as tarefas a serem processadas;
- inicializarVetorExecucao: função utilizada para criar um vetor de tamanho igual a quantidade de tarefas. O valor de cada posição será o ID da máquina no qual a tarefa será executada;
- Passo 1: definindo de forma aleatória a próxima máquina de execução;
- Passo 2: marcando em qual máquina a tarefa "i" será executada.

4.2.3 Round-robin

O algoritmo *Round-robin* utiliza uma fila circular para determinar em qual máquina uma tarefa será executada. A lógica para isso é uma regra simples de matemática: se o ID da máquina de execução atual, acrescido de 1, for menor que a quantidade de máquinas utilizadas, então o ID acrescido de 1 será o ID da próxima máquina utilizada; caso contrário, a próxima máquina será a primeira da fila. Esse procedimento é apresentado no Algoritmo 3, como pseudocódigo.

Algoritmo 3 Pseudocódigo do *Round-robin*

```
Entrada: qtdTarefas, qtdMaquinas
Saída: Vetor de inteiros com o índice sendo o ID da tarefa e o valor sendo
       o ID da máquina que a executará
início
    vt = inicializarVetorTarefas();
    para i = {8, 16, 32} faça
        vet = inicializarVetorExecucao(qtdTarefas);
        idMaquina = 0;
        isPrimeiraExecucao = true;
        para j = 0 até qtdTarefas faça
            //Passo 1
            if isPrimeiraExecucao = false then
                if (idMaquina + 1) < qtdMaquinas then
                    idMaquina = idMaquina + 1;
                else
                    idMaquina = 0;
                end
            isPrimeiraExecuao = false; //Passo 2
            vet[i] = idMaquina; //Passo 3
            j++;
        fim
        i = i * 2;
    fim
fim
retorna vet
```

Da mesma forma que os demais algoritmos, o *Round-robin* também fez o uso de algumas funções auxiliares, descritas a seguir:

- inicializarVetorTarefas: função utilizada para criar um vetor com as tarefas a serem processadas;
- inicializarVetorExecucao: função utilizada para criar um vetor de tamanho igual a quantidade de tarefas. O valor de cada posição será o ID da máquina no qual a tarefa será executada;

- Passo 1: definindo a próxima máquina de execução. Se for a primeira execução a tarefa será designada à primeira máquina, caso contrário será seguida a lógica da fila circular;
- Passo 2: marcando a *flag* de controle de primeira execução fomo false, para que na segunda iteração seja executada a lógica da fila circular;
- Passo 3: marcando em qual máquina a tarefa "i" será executada.

4.3 Métricas de Avaliação

Otimização é uma área interdisciplinar com muitas aplicações no mundo real. Consiste em encontrar os mínimos ou máximos de uma função de várias variáveis, com valores dentro de uma determinada região do espaço multi-dimensional (MARTINEZ; SANTOS, 1995).

O principal objetivo dos algoritmos online é alcançar a solução ótima para determinado problema dentro de um domínio específico. Essa solução deve satisfazer os parâmetros de entrada bem como alcançar os objetivos propostos obtendo menor custo e/ou garantindo a melhor eficiência na execução.

Existem diversos algoritmos na literatura que podem ser utilizados para resolução desses problemas: *List-scheduling*, *Random*, *Round-robin*, *Imbal*. Contudo, um algoritmo que atinge a solução ótima para um determinado problema, pode ter um desempenho inadequado se executado em um domínio diferente. É importante estabelecer métricas de avaliação para a execução desses algoritmos.

Métricas são parâmetros e estatísticas que exibem, de forma quantitativa, o desempenho de uma determinada iniciativa colocada em prática. A ideia é mostrar os resultados obtidos pelos algoritmos em cada uma das métricas estabelecidas para este trabalho.

A seguir, são descritas as métricas utilizadas neste trabalho usando um caso de uso para exemplificá-las. Considere a execução do simulador proposto levando em consideração os seguintes parâmetros:

- Serão utilizadas 10 tarefas com cargas entre 100 e 200;
- Serão utilizados 3 computadores com configurações idênticas;
- O programa irá executar os 3 algoritmos considerando as cargas sem ordenação.

A Tabela 1 mostra as cargas geradas para exemplificação das métricas:

Tabela 1 – Exemplo de cargas de trabalho geradas

Máquina	Carga
1	153
2	147
3	131
4	172
5	179
6	192
7	199
8	184
9	181
10	106

4.3.1 Carga Total por Máquina

Esta métrica é utilizada para avaliar a carga de trabalho máxima executada em cada uma das máquinas. A partir dessa métrica, é possível identificar se as máquinas foram utilizadas com equidade ou se houve discrepância no uso.

Além da carga máxima, o simulador implementado exibe uma lista com as tarefas e cargas que foram executadas em cada uma das máquinas.

As tabelas 2, 3 e 4 mostram os resultados para essa primeira métrica, considerando os exemplos utilizados por cada um dos algoritmos.

Tabela 2 – Carga Total por Máquina: *List-scheduling*

Máquina	Carga Total	Tarefas	Cargas
1	526	1, 6, 9	153, 192, 181
2	510	2, 5, 8	147, 179, 184
3	608	3, 4, 7, 10	131, 172, 199, 106

Tabela 3 – Carga Total por Máquina: *Random*

Máquina	Carga Total	Tarefas	Cargas
1	562	5, 7, 8	179, 199, 184
2	287	9, 10	181, 106
3	795	1, 2, 3, 4, 6	153, 147, 131, 172, 192

4.3.2 Makespan

Outra métrica utilizada neste trabalho é o *makespan*, também conhecido como maior carga final. Quando se trabalha com algoritmos online, não se sabe a priori as cargas das tarefas a serem utilizadas. Um dos principais objetivos de um bom algoritmo

Tabela 4 – Carga Total por Máquina: *Round-robin*

Máquina	Carga Total	Tarefas	Cargas
1	630	1, 4, 7, 10	153, 172, 199, 106
2	510	2, 5, 8	147, 179, 184
3	504	3, 6, 9	131, 192, 181

online é tentar minimizar o *makespan*, ou seja, garantir que as máquinas disponíveis para executar as tarefas utilizem o menor tempo possível para isso (FEITELSON *et al.*, 1997; XHAFA; ABRAHAM, 2010).

O algoritmo deve propor uma lógica para que todas as máquinas finalizem a execução de todas as tarefas, garantindo um balanceamento dos esforços entre si. Uma máquina não pode ser mais utilizada que outra, caso contrário, no final da execução uma solução ótima não será encontrada.

A Tabela 5 mostra o resultado com cada um dos algoritmos, considerando novamente as cargas da Tabela 1.

Tabela 5 – Makespan

Algoritmo	Makespan
LS	608
RND	795
RR	630

Como dito anteriormente, essa é uma métrica de minimização: quanto menor for o *makespan*, melhor será considerado o algoritmo.

4.3.3 Soma de Todas as Cargas

Essa métrica é bastante simples: a ideia é apresentar apenas o total de cargas das tarefas disponíveis para execução. Com isso, é possível ter uma noção se a quantidade de máquinas disponíveis para o trabalho é suficiente, ou se serão necessárias novas estações (XHAFA; ABRAHAM, 2010) . Considerando o exemplo, o total de cargas para execução é 1644.

4.3.4 Utilização Média

Essa métrica é responsável por mostrar se o algoritmo distribui equitativamente as tarefas entre todas as máquinas. A fórmula matemática resultante para calcular tal métrica é:

$$f_{um} = \frac{cargaTotal}{maiorCargaFinal * qtdMaquinas}$$

Os valores de f_{um} estão sempre contidos no intervalo $[0, 1]$. Busca-se, aqui, a maximização: quanto mais próximo de 1, melhor o algoritmo (XHAFA; ABRAHAM, 2010). A Tabela 6 mostra a utilização média das máquinas para cada um dos algoritmos.

Tabela 6 – Utilização Média

Algoritmo	Utilização Média
LS	0.9013157895
RND	0.6893081761
RR	0.8698412698

Pelo resultado descrito na tabela é possível afirmar que a melhor execução realizada foi do algoritmo *List-scheduling*, visto que, no cenário utilizado, seus resultados estiveram mais próximos de 1.

4.3.5 Maior Carga por Máquina

Quanto maior a carga de uma tarefa, mais tempo em execução ela ficará na máquina definida pelo algoritmo. Essa métrica fornece uma visão das maiores cargas executadas em cada uma das máquinas. É um resultado simples, mas que pode mostrar se há uma grande discrepância entre as tarefas fornecidas para execução.

A Tabela 7 disponibiliza a maior carga executada em cada máquina por cada um dos algoritmos.

Tabela 7 – Maior Carga por Máquina

Máquina	LS	RND	RR
1	192	199	199
2	184	181	184
3	199	192	192

4.3.6 Carga Média

Como próprio nome sugere, essa métrica fornece a média de carga de execução de cada máquina. A fórmula para o cálculo é a seguinte:

$$f_{cm} = \frac{cargaTotal}{qtdMaquinas}$$

A ideia é garantir que todas as máquinas busquem executar uma quantidade de carga próxima a carga média, mostrando que os esforços estão balanceados (FEITELSON *et al.*, 1997). Considerando o cenário de exemplo, a carga média de execução para cada uma das máquinas é $f_{cm} = 548$.

4.3.7 Atraso por Tarefa

Como dito anteriormente, quando se trabalha com algoritmos online, não se sabe quais as cargas das tarefas à priori. Elas são enviadas para execução à medida em que chegam ao sistema.

A ideia desta métrica é medir o tempo de espera da tarefa ou o atraso que ela levou para ser executada. Neste trabalho, considerou-se que todas as tarefas chegam no sistema no mesmo período de tempo ($t = 0$). Sendo assim, o atraso de cada tarefa será igual a soma da execução de todas as tarefas executadas antes da tarefa em análise. Essa análise deve ser feita máquina a máquina (FEITELSON *et al.*, 1997).

As tabelas 8, 9 e 10 fornecem os atrasos de cada uma das tarefas considerando os seguintes aspectos: máquina em que foi executada e o algoritmo utilizado.

Tabela 8 – Atraso por Tarefa: *List-scheduling*

Máquina	Tarefa	Atraso	Observação
1	1	0	Primeira execução
	6	153	Esperou tarefa(s): 1
	9	345	Esperou tarefas(s): 1, 6
2	2	0	Primeira execução
	5	147	Esperou tarefa(s): 2
	8	326	Esperou tarefa(s): 2, 5
3	3	0	Primeira execução
	4	131	Esperou tarefa(s): 3
	7	303	Esperou tarefa(s): 3, 4
	10	502	Esperou tarefa(s): 3, 4, 7

Tabela 9 – Atraso por Tarefa: *Random*

Máquina	Tarefa	Atraso	Observação
1	5	0	Primeira execução
	7	179	Esperou tarefa(s): 5
	8	378	Esperou tarefas(s): 5, 7
2	9	0	Primeira execução
	10	147	Esperou tarefa(s): 2
3	1	0	Primeira execução
	2	153	Esperou tarefa(s): 1
	3	300	Esperou tarefa(s): 1, 2
	4	431	Esperou tarefa(s): 1, 2, 3
	6	603	Esperou tarefa(s): 1, 2, 3, 4

Tabela 10 – Atraso por Tarefa: *Round-robin*

Máquina	Tarefa	Atraso	Observação
1	1	0	Primeira execução
	4	153	Esperou tarefa(s): 1
	7	325	Esperou tarefa(s): 1, 4
	10	524	Esperou tarefas(s): 1, 4, 7
2	2	0	Primeira execução
	5	147	Esperou tarefa(s): 2
	8	326	Esperou tarefa(s): 2, 5
3	3	0	Primeira execução
	6	131	Esperou tarefa(s): 3
	9	323	Esperou tarefa(s): 3, 6

5 Resultados

Neste capítulo, são apresentados todos os resultados a partir da execução de cada algoritmo implementado. A ideia principal é analisar se tais resultados corroboram suas provas matemáticas.

5.1 Experimentos

Como citado anteriormente, para a criação dos experimentos deste trabalho, foi desenvolvido um simulador na linguagem de programação *Java*. Esse programa irá produzir uma quantidade predeterminada de tarefas com cargas geradas de forma aleatória que serão processadas por uma quantidade, também predeterminada, de computadores. Serão realizadas algumas iterações para simular ambientes com quantidade de computadores diferentes.

Os algoritmos implementados neste trabalho, *List-scheduling*, *Random* e *Round-robin*, serão responsáveis por determinar em quais máquinas cada tarefa será executada. A ideia é determinar qual algoritmo possui a melhor solução considerando os experimentos que foram realizados, a quantidade de máquinas utilizadas e a ordenação das tarefas a serem executadas. A premissa utilizada nestes experimentos é que a tarefa será executada na ordem em que ela chegar no sistema. Não foi considerado no escopo deste trabalho tarefas com prioridade.

Os três algoritmos foram executados em cada um dos ambientes simulados. O programa gerou o resultado de duas formas:

- No *log* da aplicação;
- Em um arquivo gerado no formato `.txt`.

Os resultados gerados são mostrados neste trabalho na forma de gráficos e tabelas. Assim, é possível comparar os algoritmos e determinar qual obteve a melhor solução dentro do escopo utilizado.

Todas as execuções foram realizadas em um notebook modelo *Acer Aspire 5742* que possui as seguintes especificações:

- Sistema Operacional: *Windows 7 Ultimate Service Pack 1*;
- Processador: *Intel(R) Core(TM) i5 CPU M 480 2.67GHz*;

- Memória principal (RAM): *6,00 GB*;
- Tipo de sistema: *Sistema Operacional de 64 bits.*

5.2 Resultados e Discussão

Para realização das análises desses experimentos foram definidas algumas métricas de avaliação úteis para determinar qual algoritmo será capaz de fornecer a melhor solução em termos de obter menor custo e/ou garantir a melhor eficiência na execução.

As métricas, como já descritas anteriormente, mostram a qualidade da solução proposta pelo algoritmo. Algumas são utilizadas prezando pela minimização do resultado, ou seja, quanto menor o valor obtido, melhor a solução, e outras prezam pela maximização do resultado, o oposto da primeira.

As métricas de *makespan* e utilização média são mostradas na forma de tabelas. Já as métricas de carga total por máquina e atraso por tarefa são dadas na forma de *boxplot*, que é utilizado para avaliar a distribuição empírica dos dados.

Makespan

A Tabela 11 apresenta o *makespan* obtido com cada um dos algoritmos em todos os cenários simulados: ambiente com 8, 16 e 32 máquinas. Dentro de cada um desses cenários, foi simulado, ainda, algumas formas de ordenação das tarefas: sem ordenação, ordenação das cargas na forma crescente e também na forma decrescente. É possível verificar na tabela que essa forma de ordenação produz resultados diferentes.

Tabela 11 – *Makespan* no ambiente completo

	Sem Ordenação			Ordenação Crescente			Ordenação Decrescente		
Máquinas	LS	RND	RR	LS	RND	RR	LS	RND	RR
8	9200	11864	10131	9361	11072	9361	8976	13820	9361
16	4741	8370	5604	4928	7089	4928	4501	8422	4928
32	2605	5114	3135	2695	5174	2695	2309	4939	2698

O objetivo do algoritmo é minimizar o *makespan*, ou seja, quanto menor seu valor melhor é a solução proposta pelo mesmo. Um baixo valor significa que as tarefas foram distribuídas de maneira equitativa; assim o esforço que as máquinas irão produzir tendem a ser próximos entre si.

Analisando a Tabela 11, com relação ao ambiente com as cargas sem ordenação, a melhor solução obtida foi com o algoritmo *List-scheduling*. Em todas as iterações, independente da quantidade de máquinas utilizadas, seu resultado se sobressaiu aos demais.

Considerando o ambiente com ordenação crescente das cargas tanto o *List-scheduling* quanto o *Round-robin* obtiveram o mesmo resultado. Neste cenário, com as cargas ordenadas, não há distinção com relação a implementação dos algoritmos. A premissa do *List-scheduling* é distribuir as tarefas à primeira máquina disponível, neste caso, como as tarefas estão ordenadas, essa distribuição funciona como uma fila circular, da mesma forma que o *Round-robin*.

No cenário de tarefas com ordenação decrescente, o melhor resultado foi apresentado pelo *List-scheduling*, que novamente se sobressaiu aos demais. Um detalhe importante é que nas três formas de ordenação, esta última foi a que o *List-scheduling* produziu o menor *makespan*. Em contrapartida, foi também neste cenário que o algoritmo *Random* produziu seu pior resultado: o maior *makespan* obtido.

Utilização Média

A Tabela 12 apresenta os resultados considerando a métrica da utilização média. O objetivo dessa métrica é produzir um valor próximo a 1, o que significa que as tarefas foram bem distribuídas entre as máquinas disponíveis para executá-las.

Tabela 12 – Utilização média no ambiente completo

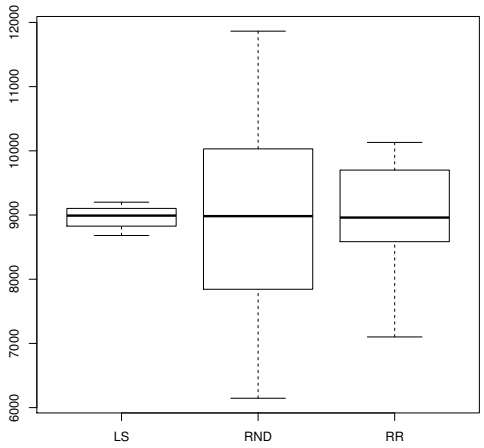
	Sem Ordenação			Ordenação Crescente			Ordenação Decrescente		
Máquinas	LS	RND	RR	LS	RND	RR	LS	RND	RR
8	0.9745	0.7557	0.8849	0.9577	0.8097	0.9577	0.9988	0.6487	0.9577
16	0.9455	0.5356	0.7999	0.9096	0.6323	0.9096	0.9959	0.5323	0.9096
32	0.8604	0.4383	0.7149	0.8317	0.4332	0.8317	0.9707	0.4538	0.8317

A utilização média seguiu, aproximadamente, os mesmos resultados do *makespan*. No cenário de cargas sem ordenação, o melhor resultado foi alcançado pelo algoritmo *List-scheduling* que produziu a melhor solução em todas as iterações (8, 16 e 32 máquinas). No cenário de cargas ordenadas de forma crescente tanto o *List-scheduling* quanto o *Round-robin* produziram o mesmo resultado. Como explicado anteriormente, nesse cenário o *List-scheduling* é executado como se fosse uma fila circular. No último cenário, de cargas ordenadas na forma decrescente, o melhor resultado obtido foi também pelo *List-scheduling*.

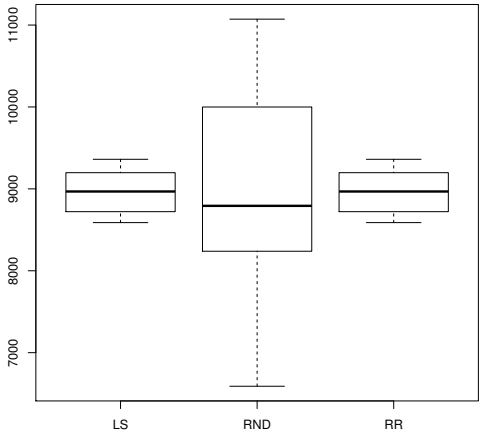
Carga Total

Primeiramente apresenta-se a carga total obtida em cada uma das máquinas. A distribuição das cargas deve ser realizada de maneira que, ao final da execução, as máquinas tenham tido esforços próximos entre si, ou seja, sua carga total dever estar próxima da carga média de execução por máquina.

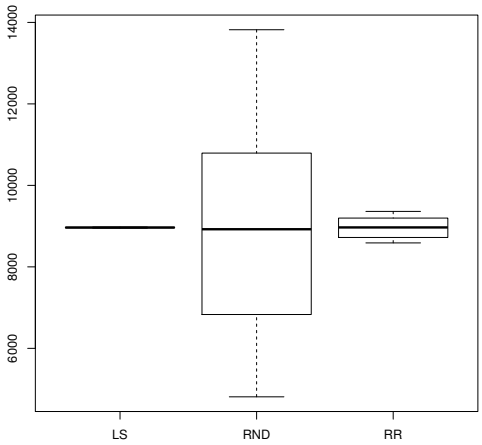
As figuras 2, 3 e 4 mostram os resultados para 8, 16 e 32 máquinas, respectivamente, em cada uma das formas de ordenação. O resultado dessas simulações apresentaram uma mesma tendência. O *List-scheduling* (LS) produziu seu melhor resultado na ordenação decrescente. É possível visualizar no gráfico que todas as máquinas tiveram suas cargas totais bem próximas da média. O *Round-robin* (RR) também demonstrou um resultado bom, não tão satisfatório quanto o *List-scheduling*, embora muito melhor que o algoritmo *Random* (RND) que, de todas as formas, produziu resultados muito distantes da média.



(a) Sem ordenação



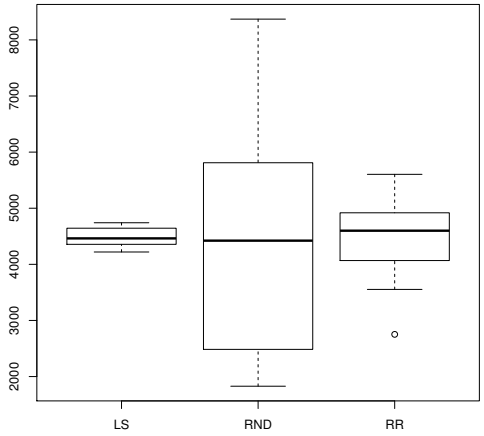
(b) Ordem crescente



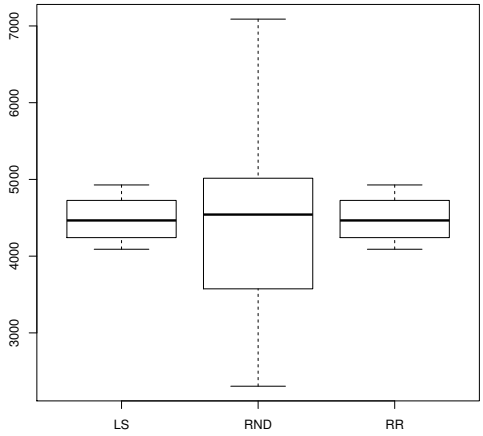
(c) Ordem decrescente

Figura 2 – Carga total para 8 máquinas

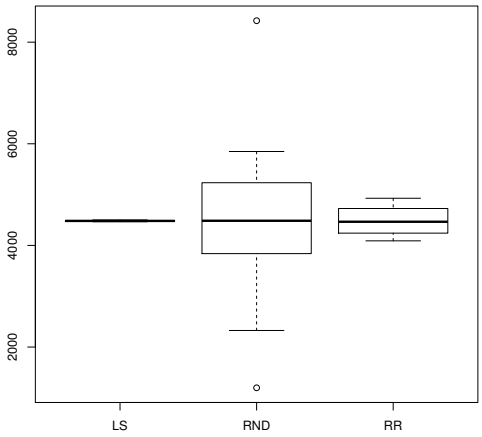
Um detalhe a ser levado em consideração é que, quando as tarefas estão em ordem crescente, os algoritmos *List-scheduling* e *Round-robin* produziram o mesmo resultado para todas as iterações.



(a) Sem ordenação



(b) Ordem crescente



(c) Ordem decrescente

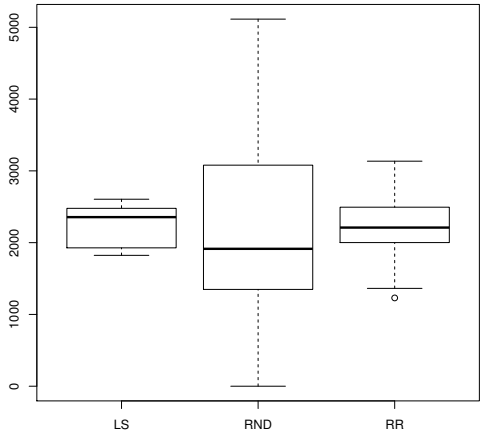
Figura 3 – Carga total para 16 máquinas

Atraso por Tarefa

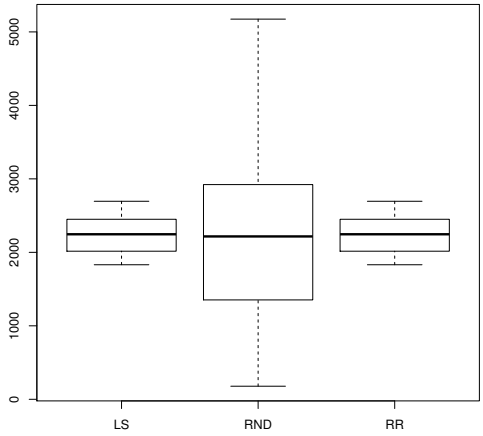
A solução proposta pelos algoritmos deve prezar pela minimização do atraso por tarefa. Quanto menor for o atraso de cada tarefa, menor será o tempo de espera para sua execução e, consequentemente, melhor será o algoritmo.

A Figura 5 apresenta o atraso das tarefas na simulação que considera 8 máquinas disponíveis para processar as cargas. Os três algoritmos mantiverem as médias próximas entre si, porém, na ordenação crescente, essa média obteve o menor valor. Considerando uma certa discrepância com relação a média, o algoritmo *Random* foi o que mais se destoou da mesma.

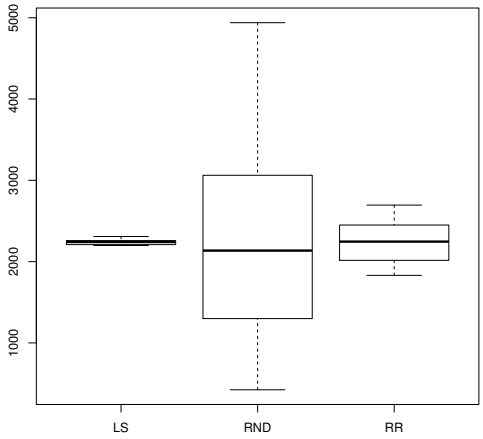
De acordo com a Figura 6, é possível verificar que, na simulação sem ordenação



(a) Sem ordenação



(b) Ordem crescente

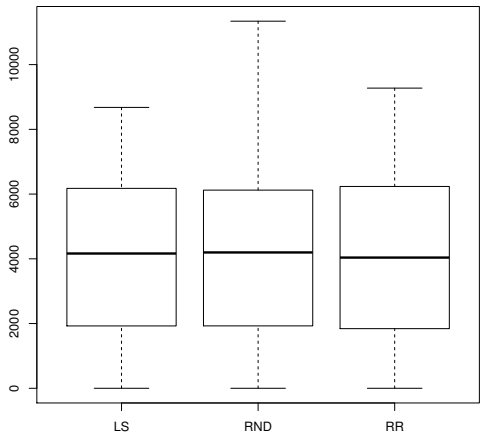


(c) Ordem decrescente

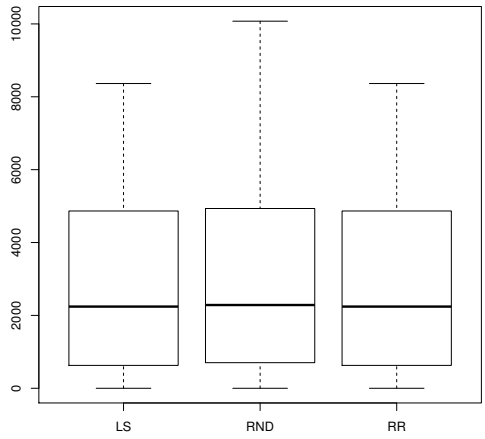
Figura 4 – Carga total para 32 máquinas

e com ordenação crescente, todos os algoritmos se destoaram um pouco da média em algum momento. Apenas na ordenação crescente que o *List-scheduling* e *Round-robin* se mantiveram próximos dela. Contudo, o algoritmo *Random* produziu um resultado ruim em todas as execuções considerando 16 máquinas disponíveis para executar as tarefas.

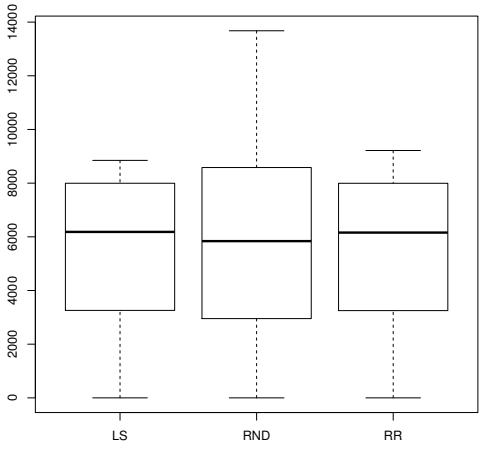
Comparando a Figura 6 com a Figura 7, os resultados foram similares. Na simulação sem ordenação e com ordenação crescente todos os algoritmos também se destoaram da média, enquanto que, na ordenação decrescente, apenas o *Random* o fez. Sua média também ficou um pouco diferente dos demais algoritmos, considerando 32 máquinas para execução, ela ficou menor.



(a) Sem ordenação

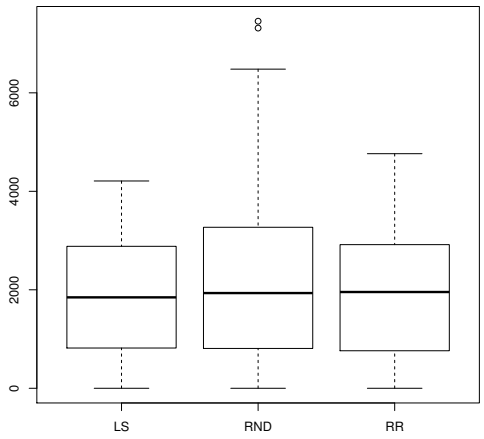


(b) Ordem crescente

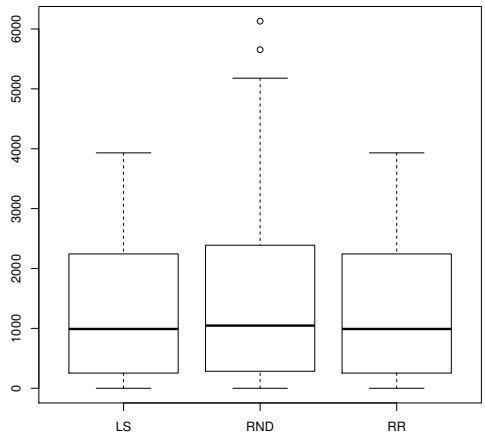


(c) Ordem decrescente

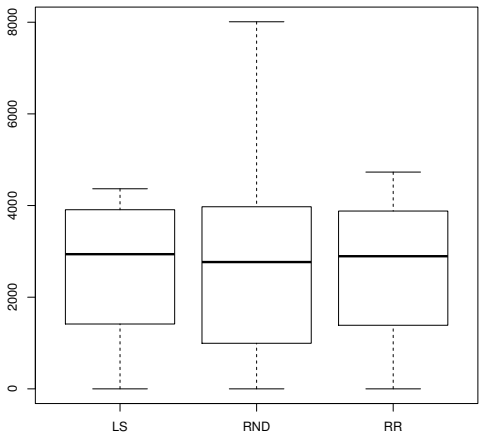
Figura 5 – Atraso para 8 máquinas



(a) Sem ordenação

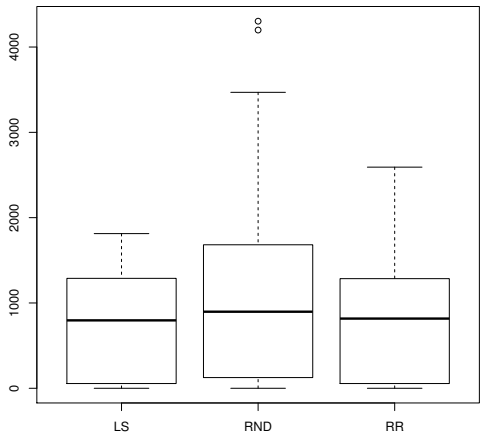


(b) Ordem crescente

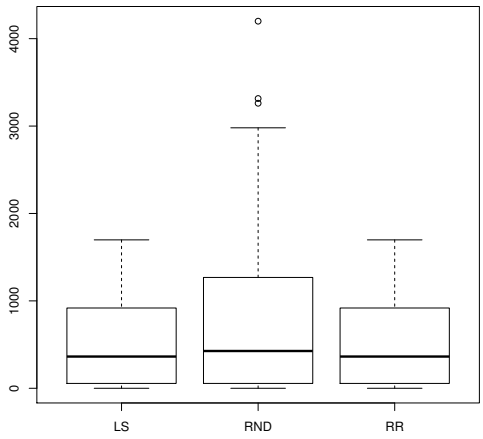


(c) Ordem decrescente

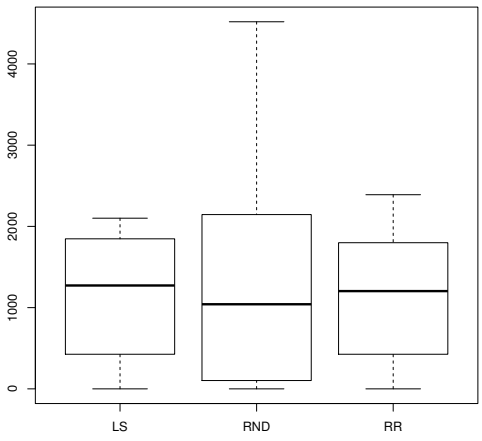
Figura 6 – Atraso para 16 máquinas



(a) Sem ordenação



(b) Ordem crescente



(c) Ordem decrescente

Figura 7 – Atraso para 32 máquinas

6 Conclusões

Atualmente, a utilização de sistemas computacionais pela sociedade aumentou significativamente. Praticamente toda pessoa tem pelo menos um *smartphone*. Existem ainda pessoas apaixonadas por tecnologia que possuem *smartphones*, *tablets*, *notebooks*, *smartwatches*, entre diversos outros dispositivos que fazem o uso de um sistema computacional.

Há de se falar ainda no conceito de *Internet of Things* (IoT). A *International Telecommunication Union* (ITU) define a IoT como uma infra-estrutura global para a Sociedade da Informação, permitindo serviços avançados através da interconexão (física e virtual) de coisas com base em tecnologias de informação e comunicação interoperáveis existentes e em evolução.

Com esse grande número de novos dispositivos e novos sistemas computacionais é extremamente importante utilizar um escalonador de tarefas eficiente que seja capaz de distribuir essas tarefas de forma inteligente buscando o balanceamento dos recursos utilizados.

A literatura oferece diversos algoritmos heurísticos para tratar o problema do escalonamento de tarefas. Uma análise criteriosa deve ser realizada sobre o domínio do problema para definir qual será a melhor solução. Para auxiliar nessa questão e otimizar o resultado é necessário adotar algumas métricas de desempenho, que por sinal estão sendo bastante utilizadas também pela literatura.

Este trabalho de conclusão de curso teve como finalidade a apresentação e implementação de três desses algoritmos: *List-scheduling*, *Random* e *Round-robin*. Foram criados alguns experimentos para que as métricas fornecidas pela literatura pudessem ser aplicadas, dentre elas podemos citar o *makespan* e a utilização média.

Observou-se que, na grande maioria das simulações, o algoritmo *List-scheduling* obteve os melhores resultados para as diversas métricas avaliadas. Um detalhe importante que deve ser levado em consideração também, é que, se as tarefas estiverem ordenadas por ordem crescente de carga, o algoritmo *Round-robin* produz o mesmo resultado que o algoritmo *List-scheduling*. Assim, dependendo da complexidade das implementações dos mesmos, o *Round-robin* poderá ser escolhido.

6.1 Trabalhos Futuros

O foco deste trabalho foi analisar alguns algoritmos heurísticos oferecidos pela literatura dentro de cenários específicos. As máquinas utilizadas nas simulações eram

idênticas, sendo assim o tempo de execução de uma tarefa não varia de uma máquina para outra. Além disso, o paradigma online utilizado foi o modelo de sequência, no qual não há a preempção das máquinas.

Para trabalhos futuros sugere-se a implementação de outros algoritmos da literatura considerando outros cenários. Por exemplo, a utilização do paradigma *timestamp*, onde cada tarefa tem um horário de chegada que é utilizado para definir a ordem de execução. Este paradigma, diferente do estudado neste trabalho, pode revogar suas decisões e as tarefas não precisam ser executadas na ordem que chegam no sistema.

Referências

ALBERS, S. Better bounds for online scheduling. In: LEIGHTON, F. T.; SHOR, P. (Ed.). *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1997. p. 130–139. Citado 3 vezes nas páginas 9, 10 e 15.

_____. Online algorithms: a survey. *Mathematical Programming*, Springer, v. 97, n. 1–2, p. 3–26, 2003. Citado na página 10.

ALBERS, S.; SCHRÖDER, B. An experimental study of online scheduling algorithms. *Journal of Experimental Algorithmics (JEA)*, ACM, v. 7, p. 3, 2002. Citado na página 10.

ARRUDA, F. R. de. *Simulador de escalonamento distribuído de processos baseado no algoritmo Round-robin*. 2001. Disponível em: <https://www.ime.usp.br/~kon/MAC5755/trabalhos/software/FlavioArruda/>. Acesso em: 28 jun. 2018. Citado na página 16.

BARTAL, Y.; FIAT, A.; KARLOFF, H.; VOHRA, R. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, v. 51, n. 1–2, p. 359–366, 1995. Citado 2 vezes nas páginas 9 e 10.

_____. New algorithms for an ancient scheduling problem. Journal of Computer and System Sciences, v. 51, p. 359–366, 1995. Citado na página 15.

BLAKE, B. A. Assignment of independent tasks to minimize completion time. *Software: Practice and Experience*, Wiley Online Library, v. 22, n. 9, p. 723–734, 1992. Citado na página 15.

CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transactions on Software Engineering, v. 14, p. 141–154, 1988. Citado na página 12.

CHAPIN, S. J.; WEISSMAN, J. B. An experimental study of online scheduling algorithms. *Electrical Engineering and Computer Science*, v. 7, p. 40, 2002. Citado 2 vezes nas páginas 12 e 13.

CHEN, B.; POTTS, C. N.; WOEGINGER, G. J. A review of machine scheduling: Complexity, algorithms and approximability. In: DU, D.-Z.; PARDALOS, P. M. (Ed.). *Handbook of Combinatorial Optimization*. EUA: Springer-Verlag, 1998. v. 3, p. 21–169. Citado na página 15.

FEITELSON, D. G.; RUDOLPH, L.; SCHIEGELSHOHN, U.; SEVCIK, K. C.; WONG, P. Theory and practice in parallel job scheduling. In: FEITELSON, D. G.; RUDOLPH, L. (Ed.). *Job Scheduling Strategies for Parallel Processing*. Switzerland: Springer, 1997. (Lecture Notes in Computer Science, v. 1291), p. 1–34. Citado 3 vezes nas páginas 15, 24 e 26.

FLEISCHER, R.; WAHL, M. Online scheduling revisited. In: PATERSON, M. S. (Ed.). *Algorithms – ESA 2000*. Switzerland: Springer, 2000. (Lecture Notes in Computer Science, v. 1878), p. 202–210. Citado 3 vezes nas páginas 9, 10 e 15.

GAREY, M. R.; JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. EUA: W. H. Freeman and Company, 1979. Citado 2 vezes nas páginas 12 e 15.

GRAHAM, R. L. Bounds for certain multiprocessing anomalies. *Bell Labs Technical Journal*, Wiley Online Library, v. 45, n. 9, p. 1563–1581, 1966. Citado 3 vezes nas páginas 9, 10 e 15.

KARGER, D. R.; PHILLIPS, S. J.; TORNG, E. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, v. 20, p. 400–430, 1996. Citado 3 vezes nas páginas 9, 10 e 15.

KRUMKE, S. O. *Online Optimization Competitive Analysis and Beyond*. Tese (Doutorado) — Techniche Universität, Berlim, 2001. Citado na página 14.

KRUMKE, S. O.; THIELEN, C. *Introduction to Online Optimization*. University of Kaiserslautern, 2014. Notas de Aula. Citado na página 9.

MARTINEZ, J. M.; SANTOS, S. A. *Métodos computacionais de otimização*. Rio de Janeiro: IMPA, 1995. 256 p. (Apostilas do Colóquio Brasileiro de Matemática). Citado na página 22.

PRUHS, K.; TORNG, E.; SGALL, J. Online scheduling. In: LEUNG, J. Y.-T. (Ed.). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. EUA: CRC Press, 2004. v. 42, p. 3–17. Citado 2 vezes nas páginas 9 e 13.

RASMUSSEN, R. V.; TRICK, M. A. Round-robin scheduling: A survey. *European Journal of Operational Research*, Elsevier, v. 188, n. 3, p. 617–636, 2008. Citado na página 10.

ROBERT, Y.; VIVIEN, F. *Introduction to scheduling*. EUA: CRC Press, 2009. Citado na página 10.

STOCKINGER, H. Defining the grid: a snapshot on the current view. The Journal of Supercomputing, v. 42, p. 3–17, 2007. Citado na página 12.

XHAFA, F.; ABRAHAM, A. Computational models and heuristic methods for grid scheduling problems. *Future Generation Computer Systems*, v. 26, p. 608–621, 2010. Citado 2 vezes nas páginas 24 e 25.