



UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”
Câmpus de São José do Rio Preto

Pedro Simões Catali

**Comparação De Algoritmos De Escalonamento De Tarefas Em Ambientes
De Edge-Computing**

São José do Rio Preto
2023

Pedro Simões Catali

**Comparação De Algoritmos De Escalonamento De Tarefas Em Ambientes
De Edge-Computing**

Trabalho de Conclusão de Curso (TCC) apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, junto ao Conselho de Curso de Bacharelado em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Câmpus de São José do Rio Preto.

Orientador: Prof. Dr. Aleardo Manacero Junior

São José do Rio Preto
2023

Pedro Simões Catali

Comparação De Algoritmos De Escalonamento De Tarefas Em Ambientes De Edge-Computing

Trabalho de Conclusão de Curso (TCC) apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, junto ao Conselho de Curso de Bacharelado em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Câmpus de São José do Rio Preto.

Comissão Examinadora

Prof. Dr. Aleardo Manacero Junior
UNESP – Câmpus de São José do Rio Preto
Orientador

Prof. Dr. Geraldo Francisco Donega Zafalon
UNESP – Câmpus de São José do Rio Preto

Prof. Dr. Rodrigo Capobianco Guido
UNESP – Câmpus de São José do Rio Preto

São José do Rio Preto
2023

C357c

Catali, Pedro Simões

Comparação de Algoritmos de Escalonamento de Tarefas em Edge Computing / Pedro Simões Catali. -- São José do Rio Preto, 2023
40 f. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto

Orientador: Aleardo Manacero Junio

1. Ciência da computação. 2. Computação em Grade (Sistemas de computador). 3. Computação em nuvem. 4. Simulação (Computadores digitais). I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Dedico este trabalho a todos que sempre estiveram ao meu lado e sempre acreditaram em mim.

Agradecimentos

Agradeço ao meu orientador Prof. Dr. Aleardo Manacero Júnior por aceitar conduzir o meu trabalho de pesquisa, pelo sua dedicação e paciência durante o projeto e por me auxiliar durante meu período de intercâmbio. E também agradeço disponibilidade e a atenção dos professores que aceitaram fazer parte da banca avaliadora.

À Universidade UNESP e todos os seus professores que sempre proporcionaram um ensino de alta qualidade e sempre me mantiveram motivado durante meus estudos.

Aos meus pais e família que sempre apoiaram as minhas decisões, me guiaram durante a vida e nunca deixaram de acreditar em mim. Em especial, agradeço aos meus tios, que estiveram ao meu lado, ao meu primo, ao meu pai e à minha mãe pelo seu auxílio neste projeto.

À Camila, que durante todo meu período de estudo e produção deste trabalho, esteve comigo, as noites e dias que passei escrevendo este projeto.

Por último, mas não menos importante, aos meus amigos, amigas, amigues e camaradas que me estimularam, me escutaram, me inspiraram. Todos, muito importantes!

A teoria sem a prática de nada vale, a prática sem a teoria é cega.

Lenin

Resumo

O desenvolvimento dos aparelhos de IoT (*Internet of Things*) tornou possível o estabelecimento de um setor no qual a automatização de tarefas e extração de informações são facilitadas, permitindo a existência dos ambientes inteligentes. Entretanto, nessa situação, grandes quantidades de dados são gerados, causando um excessivo consumo de banda, assim como lentidão na resposta dos dispositivos. Com o objetivo de solucionar tal problema, redes de *Edge Computing* são utilizadas na transferência da carga de processamento de dados da nuvem para as bordas do sistema, ou seja, faz-se um pré-processamento dos dados nas proximidades dos aparelhos que os geram. A *Edge* é constituída por aparelhos de capacidade de computação limitada e variável, sendo necessária uma boa maneira para distribuir as tarefas que devem ser realizadas. Algoritmos de escalonamento são uma forma de distribuir essas tarefas, a fim de minimizar o tempo necessário para a realização das mesmas. Para validar a qualidade de tais códigos, realizam-se testes com o uso de simuladores. Neste trabalho, são observados e comparados três algoritmos de escalonamento em ambientes *Edge Computing*: o AaSA (*Application-aware Scheduling Algorithm*); o IBGWO (*Improved Binary Grey Wolf Optimizer*); e o MMCTS (*Modified Monte Carlo Tree Search*), por meio da ferramenta de simulação iSPD (*iconic Simulator of Parallel and Distributed Systems*) com o objetivo de determinar o melhor dentre os três.

Palavras-chave: *Edge Computing*. Escalonamento. Computação em grade. Simulação.

Abstract

The development of IoT (Internet of Things) devices has made it possible to establish an industry in which task automation and information extraction are facilitated, enabling the existence of smart environments. However, in this situation, large amounts of data are generated, causing excessive bandwidth consumption as well as slow response times for devices. In order to solve this problem, Edge Computing networks are used to transfer the data processing load from the cloud to the edges of the system, i.e., data is pre-processed in the vicinity of the devices that generate it. The Edge consists of appliances of limited and variable computing capacity, and a good way is needed to distribute the tasks that must be performed. Scheduling algorithms are one way to distribute these tasks in order to minimize the time required to perform them. To validate the quality of such codes, tests are performed using simulators. In this work, three scheduling algorithms in Edge Computing environments are observed and compared: the AaSA (Application-aware Scheduling Algorithm); the IBGWO (Improved Binary Grey Wolf Optimizer); and the MMCTS (Modified Monte Carlo Tree Search), through the simulation tool iSPD (iconic Simulator of Parallel and Distributed Systems) with the objective of determining the best among the three.

Keywords: Edge Computing. Scheduling. Grid computing. Simulation.

Lista de figuras

Figura 1 – Arquitetura de um sistema <i>Edge Computing</i>	17
Figura 2 – Modelo de grade criado no iSPD.	30
Figura 3 – Gráfico do processamento total em cada recurso em MFLOPS por recurso após 1.000 tarefas no IBGWO.	32
Figura 4 – Gráfico do processamento total em cada recurso em MFLOPS por recurso após 1.000 tarefas no AASA.	33
Figura 5 – Gráfico do processamento total em cada recurso em MFLOPS por recurso após 1.000 tarefas no MMCTS.	33
Figura 6 – Gráfico dos tempos de simulação em relação ao número de tarefas realizadas.	35
Figura 7 – Gráfico das velocidades de processamento em relação a quantidade de tarefas realizadas.	35
Figura 8 – Gráfico da eficiência de cada algoritmo em relação a quantidade de tarefas realizadas.	36

Lista de tabelas

Tabela 1 – Definições das máquinas escolhidas.	30
Tabela 2 – Resultado das simulações, com 100 tarefas, dos algoritmos de escalonamento.	34
Tabela 3 – Resultado das simulações, com 1.000 tarefas, dos algoritmos de escalonamento.	34
Tabela 4 – Resultado das simulações, com 10.000 tarefas, dos algoritmos de escalonamento.	34
Tabela 5 – Resultado das simulações, com 100.000 tarefas, dos algoritmos de escalonamento.	34

Lista de abreviaturas e siglas

AaSA	Application-aware Scheduling Algorithm
BGWO	Binary Grey Wolf Optimizer
EFESTO	Efficient parEto optimality-baSed Task scheduling for vehicular edge cOmputing
GAD	Grafo Acíclico Direcionado
GSPD	Grupo de Sistemas Paralelos e Distribuído
GWO	Grey Wolf Optimizer
IBGWO	Improved Binary Grey Wolf Optimizer
IoT	Internet of Things
iSPD	iconic Simulator of Parallel and Distributed Systems
MCTS	Monte Carlo Tree Search
MMCTS	Modified Monte Carlo Tree Search
UCB	Upper Confidence Bound
UCP	Unidade Central de Processamento
QoS	Quality of Services

Sumário

1	INTRODUÇÃO	13
1.1	Motivação	13
1.2	Objetivos	14
1.3	Organização do Trabalho	14
2	REVISÃO BIBLIOGRÁFICA	15
2.1	Escalonamento	15
2.1.1	Escalonamento em Computação	15
2.1.1.1	Algoritmos de Escalonamento	16
2.2	Computação em nuvem	16
2.2.1	<i>Edge Computing</i>	16
2.2.2	Objetivos do <i>Edge Computing</i>	17
2.2.3	Escalonamento de Tarefas em <i>Edge Computing</i>	18
2.2.4	Sistemas de <i>Edge Computing</i> Conhecidos	19
2.3	iSPD	19
2.4	Trabalhos Relacionados	20
3	ALGORITMOS E METODOLOGIA	22
3.1	<i>Modified Monte Carlo Tree Search</i>	22
3.1.1	Modelamento do Sistema	22
3.1.2	Descrição Matemática	23
3.2	<i>Improved Binary Grey Wolf Optimizer</i>	25
3.2.1	Modelamento do Sistema	25
3.2.2	Descrição Matemática	26
3.3	<i>Application-aware Scheduling Algorithm</i>	28
3.3.1	Modelamento do Sistema	28
3.4	Metodologia	29
3.4.1	Aplicação do Sistema no iSPD	29
4	RESULTADOS	32
4.1	Análise dos Resultados	32
5	CONCLUSÃO	37
5.1	Trabalhos Futuros	37
	REFERÊNCIAS	39

1 Introdução

O desenvolvimento e progresso das tecnologias relacionadas à IoT (*Internet of Things*) permite um grande avanço em dispositivos inteligentes capazes de acessar a rede e realizar serviços de maneira mais eficiente, como exposto por Sittón e Canedo ([SITTÓN-CANDANEDO et al., 2019](#)). Em consequência do crescimento do número de dispositivos conectados simultaneamente, ocorre um grande aumento no consumo de banda e produção de dados nas bordas da rede, fazendo com que o processamento de dados na nuvem sofra com a demora na resposta e com graves congestionamentos.

Em razão disso, o *Edge Computing* foi desenvolvido, segundo Panchali ([PANCHALI, 2018](#)), como um dos paradigmas para solucionar esse problema, oferecendo recursos computacionais para os dispositivos mais distantes. E, para solução do conflito de distribuição de tarefas nesses dispositivos de capacidade limitada, surgem os algoritmos de escalonamento que determinam a melhor maneira de realizar a distribuição de recursos.

Esses algoritmos precisam ser observados e analisados, por meio de testes que garantam sua eficiência. Existem duas possíveis abordagens para análise dos mesmos: realizar testes fisicamente, com grandes redes de computadores, ou usando simuladores que, embora não sejam tão precisos, imitam as redes físicas, mas com o custo diminuído, pois não é necessário possuir grande número de computadores. A desvantagem do uso simuladores em relação ao ambiente físico é que eles não são tão exatos.

Neste trabalho foi realizada a comparação entre três algoritmos, AaSA (*Application Aware Scheduling Algorithm*), IBGWO (*Improved Binary Grey Wolf Optimizer*) e MMCTS (*Modified Monte Carlo Tree Search*), por meio da ferramenta de simulação iSPD (*iconic Simulator of Parallel and Distributed Systems*). A análise foi feita comparando os gráficos e os resultados gerados pela ferramenta escolhida.

1.1 Motivação

A necessidade de uma constante evolução no manuseio de sistemas de IoT e o crescimento da complexidade das tarefas relacionadas ao tratamento de dados, devido às grandes quantidades de dados produzidos, impõem a demanda por sistemas cada vez mais rápidos e eficientes. Sistemas *Edge Computing* são recursos para enfrentar esses problemas de maneira viável e crítica, com diversos algoritmos e modelos capazes de amenizar e melhorar a situação dos sistemas. Este trabalho é motivado pela necessidade de análise do desempenho e da eficiência de algoritmos de escalonamento utilizados em sistemas de *Edge Computing*.

1.2 Objetivos

O objetivo deste trabalho é realizar a comparação entre os algoritmos AATS (*Application-aware Scheduling Algorithm*), IBGWO(*Improved Binary Grey Wolf Optimizer*) e MMCTS (*Modified Monte Carlo Tree Search*), por meio de testes e análises gráficas, para melhor visualização de como cada algoritmo se comporta. Utiliza-se como ferramenta de simulação o iSPD, empregando como métricas de desempenho desses códigos o tempo de realização das tarefas e a eficiência dos algoritmos.

1.3 Organização do Trabalho

As etapas desse trabalho estão organizadas de modo a expor as teorias e processos que o norteiam. Desse modo, após estas considerações iniciais, no capítulo 2 são apresentados os conceitos que permitem uma melhor compreensão do contexto em que se insere o projeto, sendo estes: escalonamento; computação em nuvem e *Edge Computing*; o simulador iSPD. Por fim, são apresentados outros trabalhos relacionados à área de escalonadores.

No capítulo 3, são descritos os algoritmos, brevemente contextualizando-os, mostrando os modelos criados pelos autores dos algoritmos, pseudocódigos propostos e, quando necessário, suas descrições matemáticas. Em seguida, trata-se da metodologia usada, aborda-se a geração das políticas de escalonamento, o modelamento do sistemas no iSPD, as métricas de geração de tarefas e os testes que serão realizados nos algoritmos.

No capítulo 4 analisa-se, tendo em vista os testes realizados no iSPD, o comportamento gráfico dos algoritmos relacionados aos seus funcionamentos. Em seguida, verifica-se os valores de tempo e eficiência gerados pelo simulador, para assim determinar qual código mostrou o melhor escalonador. Finalmente, no capítulo 5 realiza-se a conclusão do trabalho e são indicados os possíveis caminhos futuros de pesquisa.

2 Revisão Bibliográfica

2.1 Escalonamento

Escalonamento é um processo de alocação de recursos para realização de tarefas, em um determinado período de tempo, com a objetivo de otimizar uma operação. É conhecido em indústrias de serviço e de manufatura. Um exemplo de escalonamento foi proposto por Pinedo ([PINEDO, 2016](#)). Trata-se de uma indústria que produz sacolas de papel. O material para as operações são rolos de papel e a produção da sacola possui três estágios: impressão do logo, colagem do mesmo na lateral da sacola e a costura. Cada um desses estágios possui um número diferente de máquinas que realizam diferentes tarefas, em diferentes quantidades de tempo. Algumas delas, por exemplo, podem realizar a impressão em apenas algumas cores; outras podem diferir no tamanho da sacola que costuram.

Cada pedido da indústria indica uma quantidade diferente de sacola que deve ser produzida e enviada em uma determinada data que, caso não seja cumprida, pode gerar uma penalidade. O tempo de processamento de cada operação é diretamente proporcional ao tamanho do pedido e às penalidades geradas pelo mau gerenciamento dos pedidos. Em razão dessas características, o objetivo do escalonamento do sistema é minimizar a soma das penalidades e o tempo de produção.

2.1.1 Escalonamento em Computação

A definição de problemas de escalonamento em computadores, proposta por Blaewicz ([BLAZEWICZ et al., 2001](#)), consiste em um problema de gerenciamento de um grupo de processadores, com seus recursos previamente determinados, para a realização tarefas que possuem seus próprios custos de recurso.

É exemplificada, em Pinedo ([PINEDO, 2016](#)), uma situação de distribuição do tempo que uma UCP (Unidade Central de Processamento) dedica por tarefa. Apesar de não ser possível determinar o tempo de processamento de cada programa previamente, a distribuição desses tempos de processamento aleatórios pode ser descoberta, levando em consideração fatores como a prioridade de cada tarefa. A solução do problema determinado seria a minimização da soma esperada dos tempos de conclusão ponderados de todas as tarefas.

Por meio do sistema operacional essas tarefas são divididas em fatias menores, para evitar que as mais curtas permaneçam muito tempo na fila, em razão de haver outras mais longas e de maior prioridade. A UCP alterna essas fatias, de modo que, em qualquer intervalo de tempo, alguma tarefa esteja sendo processada, assim, tarefas curtas saem mais rapidamente do sistema.

2.1.1.1 Algoritmos de Escalonamento

De acordo com Blazewicz ([BAEWICZ et al., 2001](#)), algoritmos de escalonamento são a abstração da solução de um problema de escalonamento. Ou seja, é um conjunto de métricas, funções e restrições criadas para resolver o escalonamento, sendo que não existe um algoritmo universal, mas diversos que abordam situações diferentes de um problema.

O algoritmo *Round Robin*, como apresentado por Arpaci-Dusseau ([ARPACI-DUSSEAU; ARPACI-DUSSEAU, 2018](#)), é um exemplo de escalonador que, ao invés de executar as tarefas até a conclusão, executa uma tarefa por uma fatia de tempo e depois muda para a próxima na fila de execução. Ele o faz repetidamente até que os trabalhos sejam concluídos.

Outro exemplo de uma política de escalonamento mais simples é a STJ (*Shortest Job First*), ([ARPACI-DUSSEAU; ARPACI-DUSSEAU, 2018](#)), que executa primeiro o trabalho mais curto, depois o próximo mais curto, e assim por diante.

2.2 Computação em nuvem

A ideia de computação na nuvem existe desde 1961, ano em que o professor John McCarthy sugeriu que a tecnologia de partilha de tempo do computador poderia levar a um futuro em que o poder computacional, e até mesmo aplicações específicas, pudesse ser vendido a partir de um modelo de negócios utilitário. Porém, os sistemas de informação ainda não eram capazes de sustentar tal tecnologia, como explica Rittinghouse([RITTINGHOUSE; RANSOME, 2009](#)). O termo Nuvem tem sua origem ligada à metáfora para a Internet, originalmente derivado de sua representação em diagramas de rede, em formato de um esboço de nuvem ([RITTINGHOUSE; RANSOME, 2009](#)).

O conceito atual de Computação em Nuvem, de acordo com Wang ([WANG et al., 2010](#)), é conjunto de serviços habilitados para a rede, que fornece infraestrutura computacional escalável, com garantia de QoS (*Quality of Services*), normalmente personalizada e barata, podendo ser acessada de forma simples e abrangente.

2.2.1 *Edge Computing*

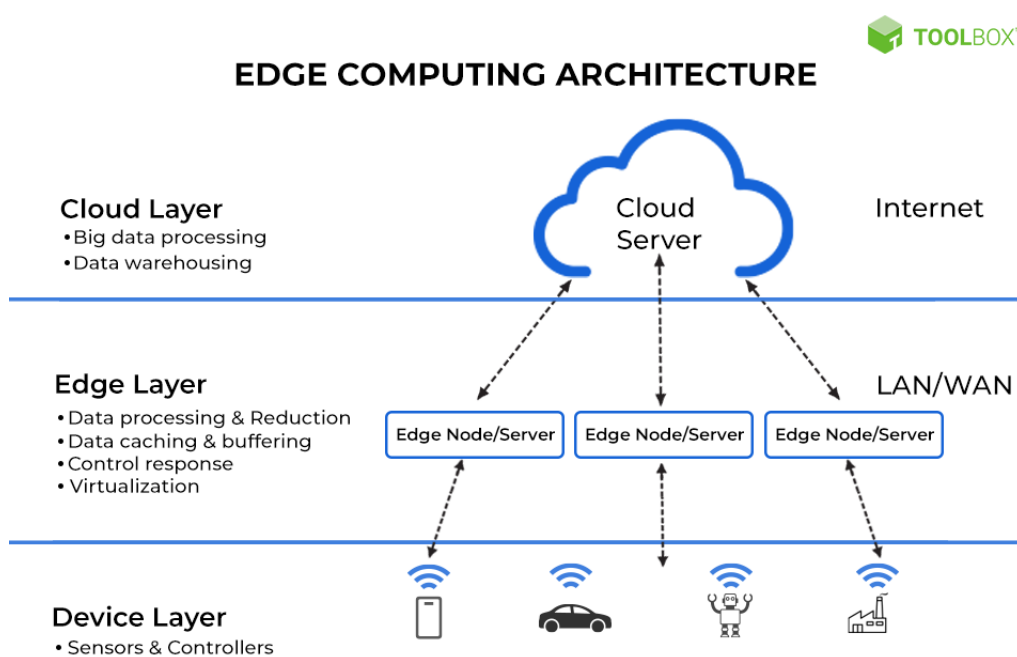
Em 2016, a *Cisco Global Cloud Index*, como citado por Panchali ([PANCHALI, 2018](#)), previu que, até o ano 2019, os dados produzidos por humanos e máquinas atingiriam 500 zettabytes e, em 2020, haveria 50 bilhões de dispositivos IoT conectados à Internet. Em razão dessa grande quantidade de dados, poderia haver demora no tempo de resposta da nuvem. Assim, para otimizar processos computacionais, surgiu a *Edge Computing* que, ao invés de processar os dados diretamente na nuvem, pré-processa os dados coletados por dispositivos IoT, para, então, enviá-los a servidores na nuvem, como os sistemas distribuídos ou os servidores remotos centralizados. Trata-se, portanto, de um método de otimização de sistemas de

computação na nuvem, nas bordas do sistema, próxima à fonte de dados, como descrito por Panchali (PANCHALI, 2018)

De acordo com Sittón e Canedo (SITTÓN-CANDANEDO et al., 2019), o termo *Edge*, borda em inglês, é utilizado porque o processamento dos dados passa a ser realizado em outros extremos, não mais apenas em nós centralizados ou distribuídos. Ou seja, transfere-se a carga de informações processadas para os dispositivos de IoT, reduzindo o tempo de resposta e a sobrecarga de tráfego na rede, sendo, por isso, denominada *Edge Computing* (Computação nas Bordas).

Pode-se observar, na figura 1, a arquitetura típica de um sistema *Edge*, em que a segunda camada realiza o processamento de dados com o auxílio dos dispositivos conectados a ela.

Figura 1 – Arquitetura de um sistema *Edge Computing*.



Fonte: Spiceworks.

2.2.2 Objetivos do *Edge Computing*

Segundo Panchali (PANCHALI, 2018), *Edge Computing* é utilizado para solucionar quatro situações distintas:

- **Latência:** Algumas aplicações, como jogos de realidade aumentada e *streaming* de vídeos, possuem requisitos de latência e desempenho em tempo real muito rigorosos. Colocar

os sistemas de *Edge* mais perto dos usuários e dos serviços pode evitar saltos de rede e propagação;

- **Conservação da largura de banda:** Certas regiões têm largura de banda limitada. Há casos em que os custos de envio de dados para nuvens ou *data centers* podem se tornar impossíveis em razão da escala ou do volume dos dados. Muitas operadoras têm limites de dados ou programas de cobrança com base no uso. Na *Edge Computing*, pode-se resolver melhor esse problema, filtrando, armazenando em cache e compactando os dados, técnicas usadas para maximizar efetivamente a largura de banda disponível;

- **Processamento eficiente:** Útil em algumas situações sem conexão confiável. Um exemplo pode ser um aplicativo de transporte ou logística que rastreia uma frota de veículos de carga, em tempo real, bem como dados críticos de temperatura da carga. Às vezes, um veículo em movimento pode perder sua conexão ao passar por túneis, áreas rurais e passagens subterrâneas. Nessas situações, é necessário evitar perda de dados. Assim, no sistema de *Edge Computing*, monitora-se o cache local e armazena-se dados até que a comunicação seja restabelecida. O sistema também pode ter técnicas de *failover* ou roteamento de retransmissão, para alternar para outra operadora, caso a principal seja perdida;

- **Segurança e privacidade:** Algumas situações devem assegurar ou remover dados, antes de enviá-los para a nuvem ou outras bordas. Isto é especialmente verdadeiro para situações envolvendo dados de saúde, de imagem ou de privacidade pessoal em vigilância. Em muitos casos, a segurança dos dados é definida pelos regulamentos da lei. Como exemplo, pode-se citar um sistema de vigilância por vídeo que precisa censurar imagens contendo crianças, em caso de transmissão pública.

2.2.3 Escalonamento de Tarefas em *Edge Computing*

Ao contrário de sistemas convencionais, que muitas vezes dispõem de recursos como UCPs, memória, armazenamento e placas gráficas, sistemas *Edge Computing* contam com recursos heterogêneos desequilibrados, isto é, com pequena capacidade computacional dos dispositivos de IoT nas bordas da rede. Esses aparelhos são, muitas vezes, programados para servidores inadequados, causando uma má distribuição dos recursos para a realização de tarefas.

Esses problemas de distribuição de recursos para realização de tarefas, relacionados a *Edge Computing*, são conhecidos como NP - Completos e são impossíveis de solucionar de uma maneira perfeita, devido ao tempo que seria exigido para tal tarefa. Por esse motivo, buscando mitigar essa situação, políticas de escalonamento de tarefas são implementadas, como uma maneira de encontrar a melhor forma de dividir recursos computacionais e de selecionar quais tarefas devem ser executadas primeiro, a fim de diminuir os tempos de resposta dos sistemas

2.2.4 Sistemas de *Edge Computing* Conhecidos

No mundo atual, possuímos alguns exemplos de aplicações em larga escala de sistemas que utilizam *Edge Computing*. O *Azure IoT Edge* transfere a análise de dados dos computadores na nuvem e os coloca em dispositivos de borda. Os três componentes que compõem o *Azure IoT Edge* são os módulos *IoT Edge*, o tempo de execução e a interface para a nuvem. O *runtime* do *IoT Edge* permite que a comunicação e outras operações sejam gerenciadas pela lógica, baseada em nuvem dos dispositivos de borda, enquanto a interface conecta-se à lógica baseada em nuvem. A interface de nuvem *IoT Edge* monitora e controla cargas de trabalho em dispositivos, usando módulos *IoT Edge*. Esses módulos podem executar contêineres compatíveis com *Docker* de serviços do *Azure*, serviços de terceiros ou até mesmo código personalizado.

Já o *Amazon AWS Greengrass* é um software que permite ao computador local ter comunicação, cache de dados, sincronia e análise no dispositivo *Edge*. O núcleo *Greengrass* permite a execução local do *AWS Lambda*, o envio de mensagens, a criação de sombras do dispositivo, com uma maior segurança e com tempo de execução baixo.

O *Apache Edgent* é um modelo de programação que pode processar fluxos de dados localmente nos dispositivos *Edge* ou *gateways*, em tempo real. Os dados determinados pelo *Edgent* são armazenados ou analisados, no dispositivo *Edge* ou nos sistemas *back-end*. O *Edgent* permite, também, que a aplicação seja transformada, para enviar ao servidor apenas dados essenciais e úteis, por meio do contínuo do fluxo de dados brutos. Esse processo permite significativa redução de dados transmitidos e armazenados para o servidor.

2.3 iSPD

Simuladores são ferramentas que imitam e reproduzem o comportamento de sistemas reais, sem a necessidade da estrutura física do sistema simulado, sendo, assim, uma maneira de se realizar testes que possuem resultados fiéis à realidade, mas com um custo monetário inferior.

O simulador escolhido para a comparação e o teste dos algoritmos é o iSPD (*iconic Simulator of Parallel and Distributed Systems*), proposto em (MANACERO et al., 2012) e produzido pelo GSPD (Grupo de Sistemas Paralelos e Distribuídos) da UNESP. Trata-se de um simulador de sistemas paralelos e distribuídos, criado em linguagem Java e com uma interface gráfica intuitiva, permitindo que até usuários inexperientes acomodem-se rapidamente, para configurar e realizar simulações por meio dos ícones gráficos. O serviço do iSPD pode ser dividido em cinco módulos:

- **Interface icônica:** A interface gráfica de usuário que fornece as funções para modelar o sistema e inserir seus parâmetros (carga, política de escalonamento, velocidade de

processamento, etc);

- **Interpretador de modelos Internos:** Interpreta o modelo icônico e o transforma em um modelo simulável, que será utilizado pelo motor de simulação;

- **Interpretador de modelos externos:** Responsável pela interpretação de modelos que foram gerados por outros simuladores. Atualmente é compatível com os modelos criados no SimGrid ([CASANOVA, 2001](#)) e no GridSim ([BUYYA; MURSHED, 2002](#));

- **Gerador e Gerenciador de Escalonadores:** Módulo utilizado para o gerenciamento de políticas de escalonamento disponíveis e para a criação de novas políticas de escalonamento, por intermédio de formulações matemáticas ou códigos em Java;

- **Motor de simulação:** Realiza a simulação dos eventos discretos, a partir dos modelos simuláveis fornecidos pelo usuário. Também é utilizado para apresentar as métricas e resultados da simulação realizada.

2.4 Trabalhos Relacionados

Devido ao fato de *Edge Computing* ser um assunto muito popular no mundo da computação, existem diversos trabalhos que discutem novas metodologias de escalonamento de tarefas e recursos. O artigo *Collaborative Scheduling of Computing Tasks for Edge Computing*, escrito por Liu ([LIU et al., 2021](#)), propõe um modelo de escalonamento cooperativo, usando como base a árvore de Monte Carlo. Ele toma o tempo médio de atraso dos dispositivos finais, com o objetivo de otimização, e utiliza o simulador EdgeCloudSim para gerar o modelo e o escalonador.

Wu Liao, por sua vez, aborda o problema de escalonamento em *Edge Computing* no artigo *Resource Allocation and Task Scheduling Scheme in Priority-Based Hierarchical Edge Computing System* ([LIAO; WU, 2020](#)), propondo um escalonador de recursos e tarefas baseado na prioridade de emergência do serviço, para minimizar o atraso total do sistema e garantir um atraso mínimo nos serviços de alta prioridade.

A questão do escalonamento pode ser observada, ainda, no artigo escrito por Jayasena ([JAYASENA; THISARASINGHE, 2019](#)), em que é introduzido o algoritmo meta-heurístico *Whale Optimization Algorithm*, que imita o comportamento de baleias. Em seguida, simula-se o código, utilizando o iFogSim, e compara-se seu desempenho com o de outros algoritmos de escalonamento meta-heurísticos.

O artigo *Improving Container Deployment in Edge Computing Using the Infrastructure Aware Scheduling Algorithm* ([KNOB; KAYSER; FERRETO, 2021](#)) atende à demanda de políticas de escalonamento que levam em consideração tanto a largura de banda de rede disponível, quanto a fila de download em cada nó do sistema de *Edge Computing*. Para isso, os autores desenvolveram um escalonador que procura reduzir a latência de implementação,

por meio de uma melhor colocação do recipiente, utilizando a fila de download e a largura de banda disponível na rede como prioridades.

Ao considerar a *Edge Computing* veicular como importante, para reduzir a latência em serviços de carros inteligentes, os autores de *Efficient Pareto Optimality-based Task Scheduling for Vehicular Edge Computing* (COSTA et al., 2022) propõem o EFESTO (*Efficient parEto optimality-baSed Task scheduling for vehicular edge cOmputing*). Trata-se de uma política de escalonamento que considera os aspectos contextuais, relacionados à mobilidade de veículos, em seu processo de decisão, minimizando o tempo de processamento em nuvens veiculares e reduzindo o tempo de utilização de recursos.

No campo de *Edge Computing* em *Smart Homes*, Xia desenvolve, em *Edge-based Energy Management for Smart Homes* (XIA et al., 2018), uma estrutura de gerenciamento de energia, baseada em *Edge Computing*, que permite diminuir o gasto de eletricidade. Isso ocorre a partir do escalonamento do tempo de operação de cada aparelho para alcançar o mínimo custo de eletricidade, empregando um algoritmo de monitoramento de carga não-intrusivo, para derivar o consumo de energia dos eletrodomésticos, com base no consumo.

Em *Hierarchical Scheduling Mechanisms in Multi-Level Fog Computing* (PEIXOTO; GENEZ; BITTENCOURT, 2022), Peixoto propõe a satisfação dos requisitos de latência de aplicações, decidindo quais componentes de serviços devem ser movidos ao topo da hierarquia *Fog-Cloud*, para aliviar a carga de trabalho de computação na borda da rede. Ou seja, o algoritmo de escalonamento proposto, em ambientes *Fog-Computing*, examina cada aplicação como uma composição de módulos que se comunicam uns com os outros, por meio de transferências de dados. Quando a nuvem é sobrecarregada, o escalonador decide qual módulo de aplicação deve ser movido para cima na hierarquia, para satisfazer os requisitos de latência, reduzindo ao mesmo tempo o atraso experimentado pelos usuários da aplicação.

Há muitos trabalhos a respeito de escalonamento em *Edge Computing* e em áreas correlatas, como *Fog Computing*. Desenvolvidos a partir de necessidades e processos distintos, todos contribuem para o avanço tecnológico. Aqui foram expostos apenas alguns exemplos, a título de demonstrar o interesse e a importância desse campo que compreende os sistemas paralelos e distribuídos.

3 Algoritmos e Metodologia

3.1 *Modified Monte Carlo Tree Search*

O algoritmo MMCTS (*Modified Monte Carlo Tree Search*) é uma variação do MCTS (*Monte Carlo Tree Search*), criado para adaptar o algoritmo original ao problema de escalonamento de tarefas. Ele foi proposto em 2006, por Kocsis e Szepesvári ([KOC SIS; SZEPE SVÁRI, 2006](#)), como ferramenta de tomada de decisões no jogo de estratégia GO. Trata-se de um algoritmo de busca em espaços combinatórios representados por árvores, cujas raízes correspondem ao início do jogo, os nós representam estados e as ligações entre os nós evidenciam a transição entre estados. As árvores são geradas, simulando diversos jogos cujos lances são escolhidos aleatoriamente. A partir dos resultados, a melhor seleção de jogadas é estabelecida, tendo em vista as que levam à vitória mais vezes. É, portanto, um algoritmo de aprendizado de máquina e, para realizar a tarefa, o programa é dividido em quatro etapas: Seleção, Expansão, Simulação e Retropropagação.

Na fase de seleção, escolhe-se qual nó deve seguir, de acordo com a política de seleção. Primeiramente, busca-se a porção da árvore que já foi representada na memória. Em seguida, seleciona-se, a partir da raiz, os próximos nós, terminando quando uma folha é visitada e o próximo nó ainda não foi representado na memória, ou quando um estado terminal é atingido, como explicam Kocsis e Szepesvári ([KOC SIS; SZEPE SVÁRI, 2006](#)).

A etapa de expansão tem como objetivo aumentar o espaço de solução do problema, caso a seleção não tenha atingido um estado terminal. Para isso, são adicionados novos nós que representam jogadas correspondentes ao estado alcançado, após a última ação realizada na seleção. Caso um estado terminal seja atingido durante essa fase, a simulação passa diretamente para a fase seguinte, a retropropagação.

Na fase de simulação, de forma aleatória ou seguindo uma estratégia heurística, realiza-se uma simulação completa do problema, para alcançar um estado terminal e criar os *payoffs*, um valor no nó final que representa os prós e contras, como apontam Kocsis e Szepesvári ([KOC SIS; SZEPE SVÁRI, 2006](#)). Por fim, na fase de retropropagação, utiliza-se os *payoffs* e propaga-se esse valor desde o nó final até a raiz, realizando todo o caminho percorrido até o último nó visitado.

3.1.1 Modelamento do Sistema

O modelo utilizado para o MMCTS é baseado no fornecido por Wang ([WANG et al., 2021](#)). Supõe-se que em uma nuvem *Edge* existem N_s servidores *Edge* heterogêneos S_i , $1 \leq i \leq N_s$, cada um com R^k atributos de recursos, em que k representa um tipo de recurso

diferente, e N_k o número de recursos. $\overline{N}_{R_{S_i}^k}$ representa a quantidade total de k recursos de computação do servidor S_i e $N_{R_{S_i}^k}$, a quantidade k de recursos que o servidor possui no momento.

Tomando $J = \{J_1, J_2, \dots, J_n\}$ como as tarefas independentes e N_j como o número total de tarefas que podem ser hospedadas na nuvem *Edge*, cada um delas só poderá ser processado por um servidor. É proposto que O_j^i mostre que J_j é hospedado por S_i . As tarefas subsequentes só podem ser recebidas por S_i se $N_{R_{S_i}^k} \leq N_{R_{J_j}^k}$, $k \in (1, N_k)$, com $N_{R_{J_j}^k}$ representando a quantidade k de recursos computacionais que J_j necessita para funcionar normalmente. Desse modo a tarefa do escalonador é maximizar N_j , escalonando-o para o S_i correto, com o problema de otimização sendo descrito por $\max_{O_j^{\text{melhor}}} (N_j)$.

3.1.2 Descrição Matemática

O nó raiz representa o estado atual de nuvem *Edge*, com N_s servidores contendo $\{N_{R_{S_i}^1}, N_{R_{S_i}^2}, \dots, N_{R_{S_i}^{N_k}}\}$, $1 \leq i \leq N_s$, e para a tarefa J_1 , $\{N_{R_{J_1}^1}, N_{R_{J_1}^2}, \dots, N_{R_{J_1}^{N_k}}\}$ é assumido como necessário. Quando o nó raiz é um nó folha, a raiz detecta se os recursos dos servidores de nuvem *Edge* são capazes de hospedar a tarefa. Caso nenhum possa fazer isso, a hospedagem falha. Caso contrário, se existem $\tilde{N}_s$ servidores que podem realizar tal alojamento, são criados $\tilde{N}_s$ nós que são transformados em filhos da raiz. Caso um nó raiz for um nó folha, constitui-se a etapa de expansão do algoritmo.

Por outro lado, um nó é selecionado como o seguinte, quando o nó raiz não é uma folha, ou, ainda, após a fase de expansão. Para determinar qual experiência anterior leva a um caminho melhor para resolução do problema, é utilizada a equação 3.1:

$$P = \frac{(N_{R_{S_i}^1}, N_{R_{S_i}^2}, \dots, N_{R_{S_i}^{N_k}}) \cdot (N_{R_{J_1}^1}, N_{R_{J_1}^2}, \dots, N_{R_{J_1}^{N_k}})}{\sqrt{(N_{R_{S_i}^1}, N_{R_{S_i}^2}, \dots, N_{R_{S_i}^{N_k}})^2} \times \sqrt{(N_{R_{J_1}^1}, N_{R_{J_1}^2}, \dots, N_{R_{J_1}^{N_k}})^2}} \quad (3.1)$$

na qual P representa a experiência anterior.

Assumindo C como o coeficiente de exploração, $D(cn_i)$ como o número do i -ésimo nó selecionado e $D(pn_i)$ como o número do nó pai do i -ésimo nó selecionado, é possível determinar o valor U de exploração, baseado nas experiências anteriores e no número de nós selecionados por intermédio da equação 3.2:

$$U = C \times P_i \times \frac{\sqrt{D(cn_i)}}{1 + D(pn_i)} \quad (3.2)$$

Para selecionar o melhor nó seguinte possível, é preciso determinar um limite superior de confiança, UCB (*Upper Confidence Bound*), e, para isso, é utilizado o valor de U e de Q , que representa o valor da exploração por meio da função 3.3:

$$UCB = Q + U \quad (3.3)$$

A etapa de simulação do algoritmo consiste na geração de diversas tarefas e seleção de nós para as hospedarem, até que isso não seja mais possível. O último nó escolhido é chamado de nó final e o processo da simulação até o nó final é chamado de *playout*.

Para refletir os prós e os contras do MMCTS, é necessário determinar um valor final V , com número de tarefas hospedadas pelo MMCTS, como L_M , e o número de tarefas simuladas por outro algoritmo, denominado L_A , a partir da equação 3.4.

$$V = L_m - L_a \quad (3.4)$$

É necessário também realizar a etapa da retropropagação, a passagem da influência de V sobre todos os Q dos nós selecionados durante o *playout* com a equação 3.5.

$$\hat{Q} = \frac{V + (D(cn_i) - 1) \times Q}{D(cn_i)} \quad (3.5)$$

Na qual $\hat{Q}$ é o valor de exploração atualizado. Por fim, o servidor hóspede será selecionado, sendo este o nó mais visitado após $N_{playout}$ vezes. E recorrendo a estas funções e definições é possível desenvolver o Algoritmo 1.

Algoritmo 1 – Modified Monte Carlo Tree Search

ENTRADA: Inicial: $\{N_{R_{S_i}^1}, N_{R_{S_i}^2}, \dots, N_{R_{S_i}^{N_k}}\}, 1 \leq i \leq N_s, \{N_{R_{J_j}^1}, N_{R_{J_j}^2}, \dots, N_{R_{J_j}^{N_k}}\}, N_{playout}$

SAÍDA: O_j^{melhor}

1. **Para** $n \leftarrow 1, N_{playout}$ **faça**
2. Nó raiz inicial a , colocar nó atual no nó raiz
3. **Enquanto** nó atual não é nó folha **faça**
4. Seleção, colocar nó atual no nó selecionado
5. Simulação
6. **Se** nó atual é um nó final **então**
7. Calcular V
8. Retropropagação
9. **Se não**
10. Expansão
- 11.
12. Selecionar nó filho mais visitado como O_j^{melhor}

3.2 Improved Binary Grey Wolf Optimizer

O algoritmo IBGWO (*Improved Binary Grey Wolf Optimizer*), proposto por Jiang (JIANG et al., 2019), tem sua origem ligada ao algoritmo de otimização GWO (*Grey Wolf Optimizer*), apresentado por Mirjalili (MIRJALILI; MIRJALILI; LEWIS, 2014), e o BGWO (*Binary Grey Wolf Optimizer*), estabelecido por Emary (EMARY; ZAWBAA; HASSANIEN, 2016). O GWO é um algoritmo meta-heurístico baseado na estrutura social de uma alcateia de lobos cinzentos.

No topo da hierarquia dos lobos cinzentos estão os alfas (ou dominantes). Eles são seguidos pelos membros da matilha e são os responsáveis por tomar decisões importantes na alcateia. Em seguida, estão os betas, subordinados responsáveis por auxiliar o alfa nas tomadas de decisão e em outras atividades do grupo. Seu papel é de segundo em comando, reforçando as ordens do alfa para o restante da alcateia e devolvendo *feedback* para o mesmo.

Na base da pirâmide social encontram-se os ômegas, submissos a todos os outros, desempenhando o papel de bodes expiatórios, ou seja, sua importância está em satisfazer o restante da matilha e manter a estrutura de dominância. Os demais dos lobos que não se encaixam nas categorias acima são considerados deltas. São submissos aos alfas e betas, mas dominam os ômegas, e preenchem as lacunas de tarefas na alcateia, sendo escoteiros, sentinelas, caçadores, anciãos e cuidadores. Outra característica importante dessa sociedade é o funcionamento da caça, que é dividida em três fases: rastrear, perseguir e aproximar da presa; cercar e assediar até que ela pare de se mover e, por fim, atacar.

O *Grey Wolf Optimizer* propõe, a partir das características da matilha, realizar um modelo matemático que as represente essas especificidades, com o objetivo de criar um algoritmo de otimização.

3.2.1 Modelamento do Sistema

Como Jiang indica, em (JIANG et al., 2019), é possível decompor escalonamento em subtarefas dependentes, que podem ser abstraídas na forma de um grafo acíclico direcionado (GAD), com um conjunto de vértices V e um conjunto de bordas E .

Em um GAD $G = \{V, E\}$, cada vértice $v_i \in V$ representa uma subtarefa. As bordas direcionadas $e_{i,j} \in E$ indicam uma restrição de precedência, mostrando que a tarefa v_i depende de v_j . Enquanto cada subtarefa pode ser executada apenas em um servidor, esses podem executar várias tarefas em sequência. Cada uma das dependências entre subtarefas pode ser abstraída na forma de uma matriz adjacente $n \times n$, com n sendo o número de vértices no GAD.

3.2.2 Descrição Matemática

Considerando a solução mais adequada como α , a segunda e terceira como β e δ , respectivamente, e o restante como ω , assim como os lobos, é possível representar o comportamento de cercamento da presa fazendo uso das equações 3.6 e 3.7, que retratam a posição de um lobo no espaço ao redor de sua presa em qualquer lugar aleatório (JIANG et al., 2019):

$$\vec{D}_i = |\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)| \quad (3.6)$$

$$\vec{X}_i(t+1) = |\vec{X}_p(t) - \vec{A} \cdot \vec{D}_i| \quad (3.7)$$

nas quais t indica a iteração atual, $\vec{X}_p$, o vetor de posição da presa e $\vec{X}$, a posição do vetor de um lobo, enquanto que $\vec{A}$ e $\vec{C}$ são vetores de coeficiente. Estes podem ser calculados pelas equações 3.8 e 3.9:

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a} \quad (3.8)$$

$$\vec{C} = 2 \cdot \vec{r}_2 \quad (3.9)$$

onde $\vec{a}$ decresce linearmente de 2 a 0, ao longo das iterações, e os vetores $\vec{r}_1$ e $\vec{r}_2$ são vetores aleatórios em $[0, 1]$.

Uma diferença entre o GWO e o IBGWO é no parâmetro $\vec{A}$, utilizado para controlar a convergência entre a aproximação de ω com α , β e δ , ao invés de $\vec{a}$ decrescer linearmente de 2 a 0 ele decresce seguindo a equação 3.10:

$$\vec{a} = 2 - 2 \times [(e^{t/m} - 1)/(e - 1)] \quad (3.10)$$

onde e é o Número de Euler, m denota o número máximo de iterações enquanto que t a iteração atual. Já a posição de cada lobo é atualizada por meio da equação 3.11.

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \quad (3.11)$$

na qual $\vec{X}_1$, $\vec{X}_2$ e $\vec{X}_3$ são calculados separadamente com α , β e δ como a presa cercada.

A modificação feita pelo IBGWO no BGWO é nas funções 3.12 e 3.13 que convertem os valores contínuos da solução para valores binários.

$$S(v_i^k(t)) = \frac{1}{1 + e^{-10(v_i^k(t) - 0.5)}} \quad (3.12)$$

$$x_i^k(t+1) = \begin{cases} 0 & r < S(v_i^k(t)) \\ 1 & r > S(v_i^k(t)) \end{cases} \quad (3.13)$$

em que $v_i^k(t)$ representa a velocidade da partícula i , na k -ésima dimensão, durante a iteração t , e $x_i^k(t+1)$ indica sua posição. Essas equações, além de forçarem os lobos a terem valores 0 ou 1, e gerarem um gráfico em formato de S, fazem com que a posição dos ω não se modifique enquanto a velocidade se mantiver crescente, levando à perda do salto estocástico e à permanência nas posições ótimas locais. Por essa razão, Jiang (JIANG et al., 2019) propõe a utilização das equações 3.14 e 3.15 que resultam em uma função que tenha o formato V e em uma mudança de posição mais eficiente:

$$V(x_i^k(t)) = \left| \frac{2}{\pi} \arctan \left(8\pi \frac{x_1 + x_2 + x_3}{3} - x_i^k(t) \right) \right| \quad (3.14)$$

$$x_i^k(t+1) = \begin{cases} 1 - x_i^k(t) & r < V(x_i^k(t)) \\ x_i^k(t) & r > V(x_i^k(t)) \end{cases} \quad (3.15)$$

Assim, enquanto $V(x_i^k(t))$ for maior que o valor aleatório na função de transferência em forma de V, existe maior probabilidade de mudar seus vetores de posição, fazendo com que os lobos ω afastem-se dos lobos α .

Tendo em vista as funções e os comportamentos dos lobos, Jiang (JIANG et al., 2019) propõe o Algoritmo 2 que, por meio de parâmetros pré definidos, possibilita a busca e escolha do servidor mais apto para hospedar a tarefa.

Algoritmo 2 – *Improved Binary Grey Wolf Optimizer*

1. *Inicializar parâmetros do cenário*
2. *Inicializar população de lobos cinzas $X_i(i = 1, 2, \dots, n)$*
3. *Inicializar a , A e C*
4. *Calcular aptidão de cada agente de busca*
5. *Escolher melhor lobo como α , segundo como β e terceiro como δ*
6. **Enquanto** ($t < \text{número máximo de iterações}$) **faça**
7. **Para** cada agente de busca
8. | Atualizar posição do agente de busca de acordo com as equações 3.14 e 3.15
9. Atualizar a , A e C
10. Calcular aptidão de cada agente de busca
11. Atualizar α , β e δ
12. $t = t + 1$
13. *Retorna α*

3.3 *Application-aware Scheduling Algorithm*

Proposto por Thanda(00; KO, 2019) o AaSA (*Application-aware scheduling algorithm*) consiste em um algoritmo de escalonamento que leva em consideração os tipos de aplicação e seus requerimentos de QoS (*Quality of Services*), do inglês "qualidade de serviços".

Ao contrário dos algoritmos discutidos anteriormente, o AaSA não é uma adaptação e aplicação de modelos matemáticos já consolidados para um problema de escalonamento, mas um código cuja prioridade, no algoritmo de escalonamento, é verificar se as aplicações escalonadas são sensíveis à latência.

No exemplo dado por Thanda (00; KO, 2019), observa-se que um robô cirúrgico e uma câmera de segurança poderiam competir por recursos computacionais em uma rede de hospital. Entretanto, em uma cirurgia, poderia haver morte, caso a latência, entre os comandos dados ao robô e a execução dos mesmos, seja muito grande. Já a segurança do hospital não seria afetada pelo atraso de alguns segundos. Em situações semelhantes, por meio do algoritmo, é possível haver um gerenciamento flexível da inserção dos módulos de aplicação sobre nós *Edge* de diferentes capacidades.

3.3.1 Modelamento do Sistema

Pode-se escolher uma arquitetura de mestres e servos para a rede. Em cada rede *Edge*, um nó mestre é escolhido e este coleta informações e status dos seus nós servos. O nó mestre tem o papel de controlar todas as tarefas recebidas e atribui-las a seus nós *Edge* disponíveis, dependendo das informações coletadas.

Entre as informações coletadas, estão as exigências de QoS e, se alguma delas for latência mínima, a tarefa é atribuída ao nó de borda com a menor latência. Por sua vez, caso a tarefa seja tolerante a latência, com uma latência específica vinculada e com recursos de computação requeridos, ela será atribuída ao nó de acordo com a largura de banda, com a RAM e com o limite de latência.

Oo (00; KO, 2019) desenvolveu o Algoritmo 3 considerando a possibilidade de resistência a latência. Ele é executado e escolhe a atribuição mais eficiente de módulos de uma aplicação, para uma infraestrutura de rede, sempre que uma tarefa chega ao nó Mestre.

Algoritmo 3 – Application-aware Scheduling Algorithm

ENTRADA: Conjunto de nós Edge $N = \{n_1, n_2, \dots, n_i\}$ e módulos de aplicação $M = \{m_1, m_2, \dots, m_j\}$

SAÍDA: Conjunto de alocação do módulo de aplicação para o nó Edge

```

1. Inicializar  $n = 1, k = 1$ 
2. enquanto  $n \leq j$  faça
3.     se  $m_n.QoS = \text{"Sensitivo à Latência"}$  então
4.         para todo  $n_k$  em  $N$ 
5.             se  $n_k.latência$  é a menor então
6.                 Alocar( $m_n, n_k$ )
7.                 Atualizar recursos disponíveis nos nós Edge
8.         fim para
9.     fim se
10.    se  $m_n.QoS = \text{"Tolerante a latência"}$ 
11.        para todo  $n_k$  em  $N$ 
12.            se  $(n_k.CPU \leq m_n.CPU \ \& \$ 
13.                 $n_k.RAM \leq m_n.CPU \ \& \$ 
14.                 $n_k.larguradebanda \leq m_n.larguradebanda \ \& \$ 
15.                 $n_k.latencia \leq m_n.latencia$ 
16.                Alocar( $m_n, n_k$ )
17.                Atualizar recursos disponíveis nos nós Edge
18.        fim para
19.        se não
20.            Alocar( $m_i, nuvem$ )

```

3.4 Metodologia

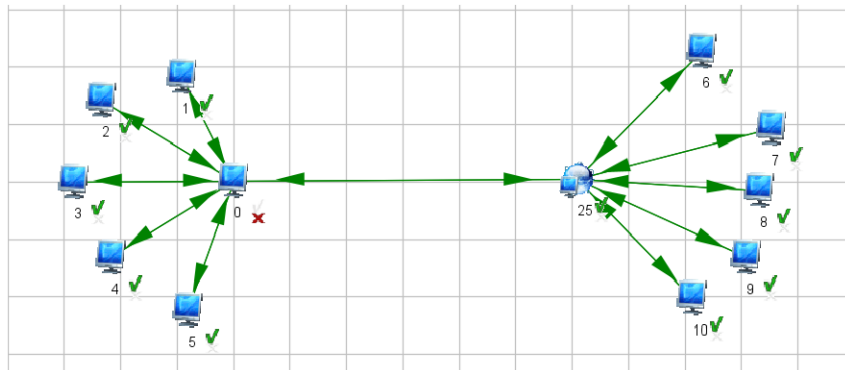
3.4.1 Aplicação do Sistema no iSPD

A primeira etapa, realizada no desenvolvimento da metodologia, é a geração dos escalonadores recorrendo ao iSPD. Para isso, utiliza-se a ferramenta de criação de políticas de escalonamento do programa e, dentro dessa aba, foi escolhida a função de produzir o escalonador por meio da classe . Essas classes foram, então, adaptadas de forma a seguir os pseudocódigos propostos para os algoritmos.

Em seguida, com base na definição de Sitton e Canedo ([SITTÓN-CANDANEDO et al., 2019](#)), foi desenvolvido o modelo de grade do sistema, a partir de dispositivos IoT homogêneos, que realizam o escalonamento. A determinação do número de máquinas do sistema segue a média de dez aparelhos IoT por residência, dado fornecido pelo vice-presidente executivo da *Intel Client Computing Group* em 2016 ([PHADINIS, 2016](#)). Na figura 2, é possível observar

o modelo de grade desenvolvido, com suas máquinas e conexões.

Figura 2 – Modelo de grade criado no iSPD.



Fonte: Elaborada pelo autor.

Os valores escolhidos como poder de processamento seguem uma progressão que aumenta em dez vezes por máquina, até o máximo de 100.000 MFLOPS. Esses valores representam aparelhos de IoT médios, desde câmeras de segurança até computadores e *laptops*. A lista das máquina pode ser observada na tabela 1, na qual *M* representa o mestre e *E* os computadores escravos.

Tabela 1 – Definições das máquinas escolhidas.

Rótulo	Poder de Processamento (MFLOPS)	Categoria
0	0	M
1	10	E
2	100	E
3	1000	E
4	10000	E
5	100000	E
6	10	E
7	100	E
8	1000	E
9	10000	E
10	100000	E

Quando os máquinas escravos estão ligadas diretamente ao mestre, não conectadas ao ícone da internet, recebem uma capacidade de banda de 200 mbps e uma latência de 0.05s. Por outro lado, computadores associados ao mestre por meio da internet possuem um valor de 100 mbps de banda e latência de 0.1s. Esses parâmetros são utilizados para auxiliar a visualização gráfica do AaSA e mostrar que dispositivos conectados diretamente ao mestre possuem uma latência teoricamente menor.

No iSPD, opta-se pelo modelo de inserção de tarefas por nó, com seu custo computacional entre 100 MFOLPS e 100.000 MFLOPS e valor de comunicação entre 10 mbps e 200

mbps. Optou-se por simulações com 100, 1.000, 10.000 e 100.000 tarefas e, com cada uma dessas quantidades são realizadas dez simulações.

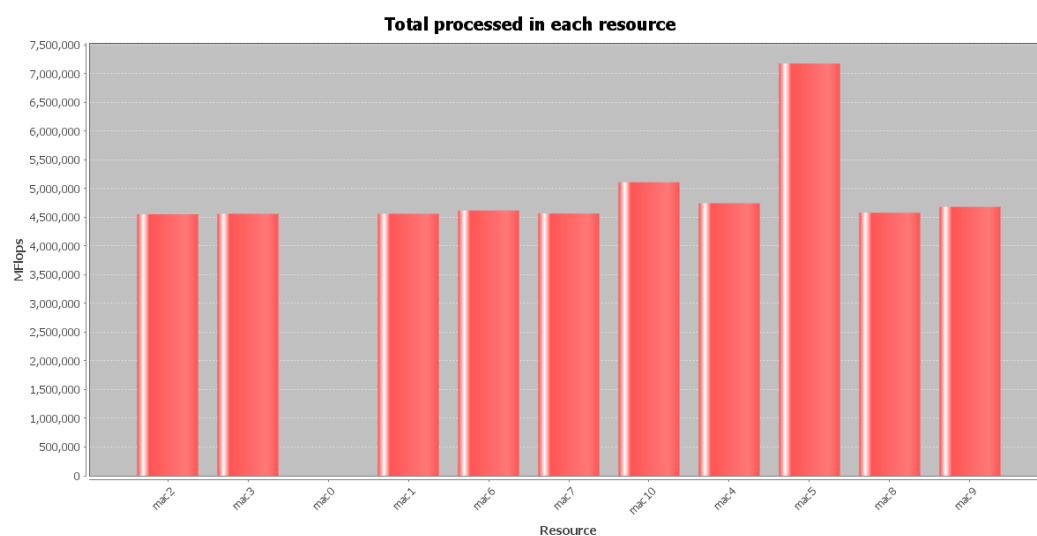
4 Resultados

4.1 Análise dos Resultados

Para facilitar a compreensão de como diferentes algoritmos de escalonamento funcionam e distribuem recursos para realização de tarefas, é preciso analisar e comparar seus comportamentos gráficos, observando, por exemplo, o gráfico da quantidade de recursos processados por cada máquina.

Na figura 3, é possível observar o gráfico gerado a partir do escalonamento de 1000 tarefas pelo IBGWO. Nesta representação, mostra-se uma clara tendência à seleção de máquinas com a maior quantidade de recursos. Isso acontece porque, para hospedar a tarefa, o algoritmo realiza sua seleção de nó *Edge*, atualizando os parâmetros de busca, com o objetivo de encontrar os três melhores para a hospedagem. Contudo, como o modelo utilizado não cria tarefas dependentes, a máquina com maior poder computacional é quase sempre escolhida.

Figura 3 – Gráfico do processamento total em cada recurso em MFLOPS por recurso após 1.000 tarefas no IBGWO.

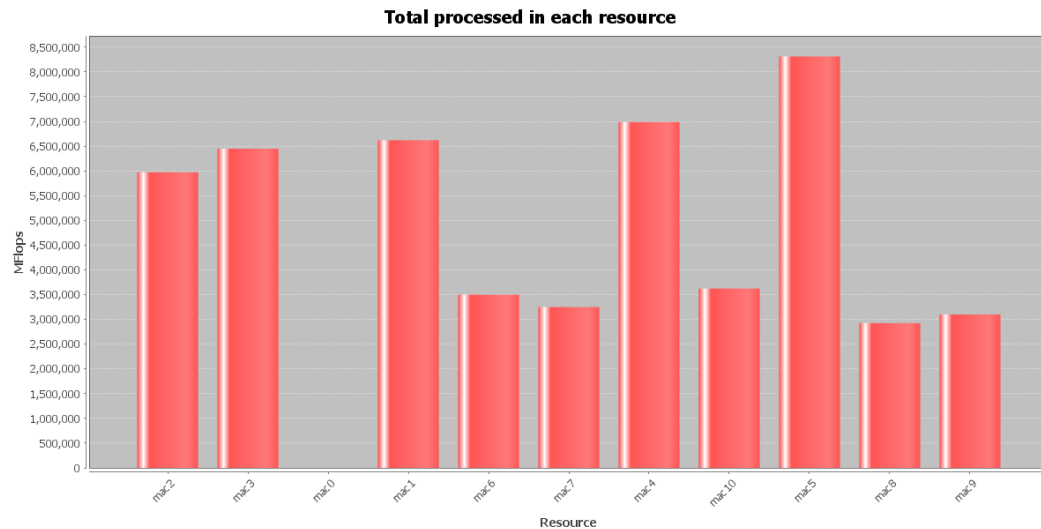


Fonte: Elaborada pelo autor.

As tarefas para computadores que possuem uma latência baixa e uma capacidade de trabalhar com uma banda de rede maior foram alocadas pelo algoritmo AaSA. É o que acontece quando notamos o comportamento de dez máquinas durante a realização de mil tarefas. As que mais trabalham, como se nota na figura 4, são os computadores 1, 2, 3, 4, e 5, ou seja, aqueles que apresentam as características mencionadas: baixa latência e condições para funcionar com banda de rede maior. Isso ocorre porque o objetivo do algoritmo é permitir que tarefas que necessitam de menor latência sejam realizadas primeiro, para evitar grandes

dessincronizações entre as ações que o usuário realiza e o que ele observa, como em ambientes de realidade virtual.

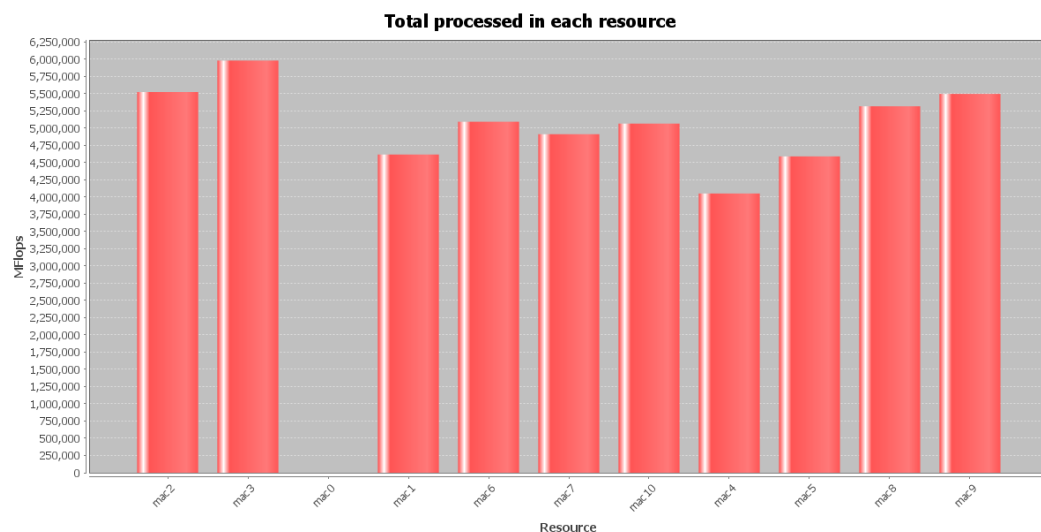
Figura 4 – Gráfico do processamento total em cada recurso em MFLOPS por recurso após 1.000 tarefas no AASA.



Fonte: Elaborada pelo autor.

No terceiro escalonador, o MMCTS, do total processado em cada nó, em relação aos anteriores, observa-se uma distribuição relativamente mais uniforme, como pode ser visto na figura 5. Tal característica é uma consequência direta da pontuação que cada de cada nó recebe a cada iteração do algoritmo, até selecionar o nó com maiores chances de ser o mais apto a alocar a tarefa. No gráfico da imagem 5, pode-se perceber que o servidor que mais pontuou na capacidade de hospedar tarefas foi o de número 3, seguido pelo 2.

Figura 5 – Gráfico do processamento total em cada recurso em MFLOPS por recurso após 1.000 tarefas no MMCTS.



Fonte: Elaborada pelo autor.

Com o intuito de facilitar a comparação entre as velocidades e eficiências dos escalonadores, desenvolveu-se os gráficos das figuras 7, 6 e 8, com base nos dados das tabelas 2, 3, 4 e 5, obtidos após a realização das simulações.

Tabela 2 – Resultado das simulações, com 100 tarefas, dos algoritmos de escalonamento.

Algoritmo	Média de tempo simulado	Eficiência média
IBGWO	29.424 s	41 %
MMCTS	62.977 s	47%
AaSA	47.017 s	45%

Tabela 3 – Resultado das simulações, com 1.000 tarefas, dos algoritmos de escalonamento.

Algoritmo	Média de tempo simulado	Eficiência média
IBGWO	469.530 s	43%
MMCTS	538.449 s	47%
AaSA	633.410 s	38%

Tabela 4 – Resultado das simulações, com 10.000 tarefas, dos algoritmos de escalonamento.

Algoritmo	Média de tempo simulado	Eficiência média
IBGWO	4.960.534 s	41%
MMCTS	5.212.047 s	46%
AaSA	6.646.536	39%

Tabela 5 – Resultado das simulações, com 100.000 tarefas, dos algoritmos de escalonamento.

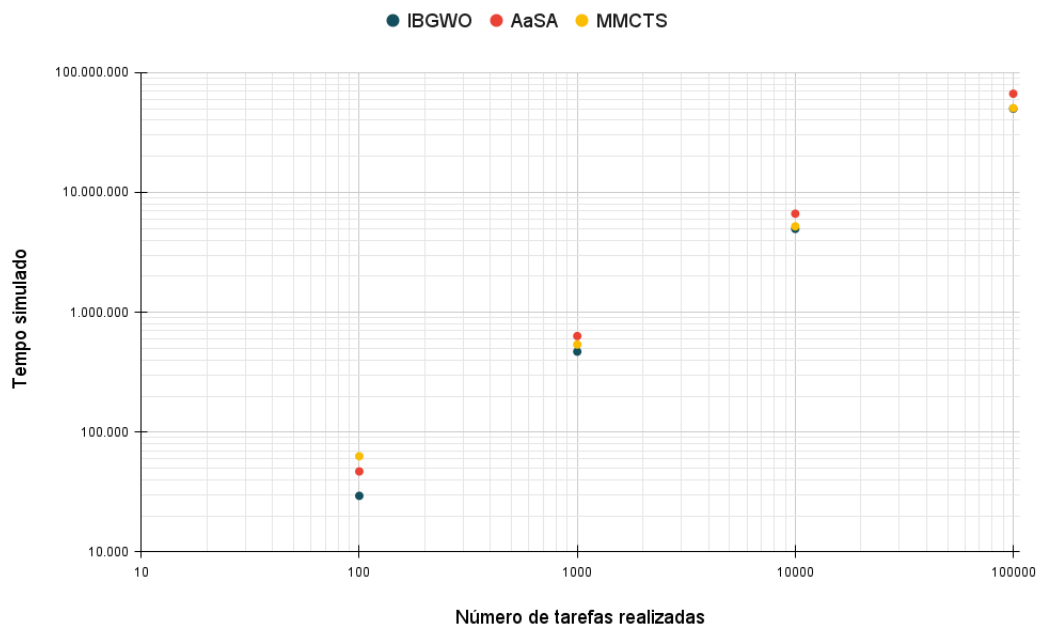
Algoritmo	Média de tempo simulado	Eficiência média
IBGWO	49.903.063 s	42%
MMCTS	50.504.732 s	47%
AaSA	66.601.104 s	29%

Quando observados os resultados dessas tabelas, é visível que existe uma tendência de crescimento linear do tempo, sempre aumentando em aproximadamente 10 vezes. Porém, quando se verificam os resultados para 100 tarefas, essa regra é quebrada. Uma possível explicação para essa diferença é o fato de as tarefas serem geradas de maneira aleatória e em número relativamente pequeno, por isso, apenas trabalhos com baixo ou alto custo computacional seriam criados, ao invés de ser uma distribuição uniforme de custos. Outra possível explicação para esse comportamento pode ser que, devido ao baixo número de tarefas, algum tipo de erro ocorre no simulador, fazendo com que elas sejam realizadas, desconsiderando os escalonadores escolhidos.

No gráfico logarítmico dos tempos de simulação dos algoritmos, na figura 6, perceb-se que o IBGWO levou a menor quantidade de tempo para realizar as simulações em todos os cenários estudados. Com o auxílio da figura 7, pode-se observar a velocidade do algoritmo

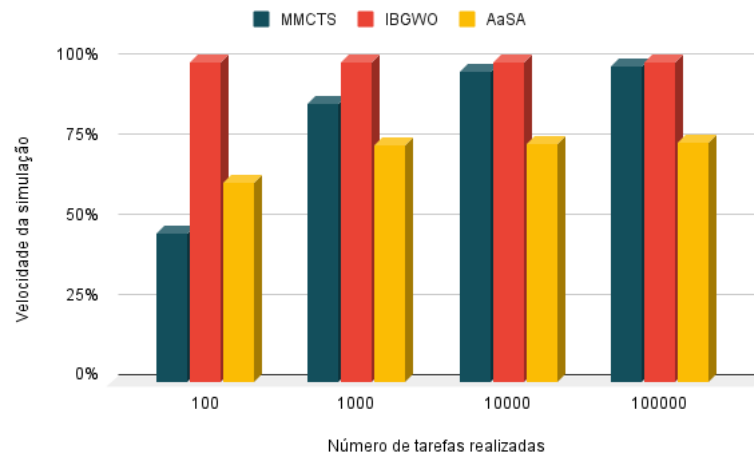
mais rápido como 100%. A velocidade dos restantes apresenta-se em proporções a essa máxima, ilustrando a superioridade do IBGWO. Além disso, observa-se algumas características dos outros códigos: o AaSA se mostrou relativamente constante em sua velocidade, sendo aproximadamente 72% tão veloz quanto o IBGWO, enquanto a velocidade do algoritmo MMCTS cresce de 47% para 99% da velocidade do IBGWO. Tal comportamento é explicado pelo fato de o *Modified Monte Carlo Tree Search* ser um algoritmo de aprendizado de máquina, ou seja, a cada iteração ele se torna mais capaz de selecionar o melhor servidor para realizar a tarefa.

Figura 6 – Gráfico dos tempos de simulação em relação ao número de tarefas realizadas.



Fonte: Elaborada pelo autor.

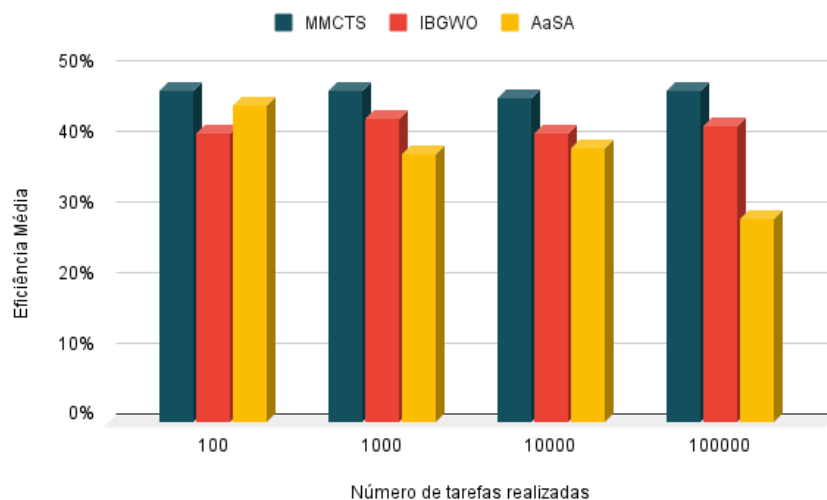
Figura 7 – Gráfico das velocidades de processamento em relação a quantidade de tarefas realizadas.



Fonte: Elaborada pelo autor.

Quando comparados os dados de eficiência, é perceptível que nenhum dos algoritmos apresenta uma eficiência particularmente alta, variando entre 29% e 47%. Na figura 8, é possível observar que o algoritmo mais eficiente é o MMCTS, apresentando uma eficiência, quase constante, de 47%. O segundo colocado é o IBGWO, com uma eficiência média aproximada de 42% e, por último, o AaSA, com uma média em torno de 38%. O baixo desempenho do *Application-aware Scheduling Algorithm* é explicado por sua política de escalonamento priorizar a sensibilidade à latência ao invés de uma melhor eficiência.

Figura 8 – Gráfico da eficiência de cada algoritmo em relação a quantidade de tarefas realizadas.



Fonte: Elaborada pelo autor.

Considerando as métricas propostas para avaliação dos algoritmos de escalonamento estudados, pode-se concluir que o melhor, entre os três, é o *Improved Binary Grey Wolf Optimizer*, porque demonstra a maior velocidade de processamento e a segunda melhor eficiência, respectivamente, 493 segundos por tarefa e 42% de eficiência. O *Modified Monte Carlo Tree Search*, por sua vez, é o segundo colocado, com uma eficiência de 47%. Percebe-se, ainda, que a diferença de velocidade é mínima, em relação ao primeiro colocado, quando realizadas simulações com 10.000 tarefas ou mais.

5 Conclusão

Neste trabalho, desenvolveu-se uma comparação entre três algoritmos de escalonamento, o AaSA (*Application Aware Scheduling Algorithm*), IBGWO (*Improved Binary Grey Wolf Optimizer*) e MMCTS (*Modfied Monte Carlo Tree Search*). Esses algoritmos foram testados com o objetivo de determinar o gerenciamento de recursos de processadores, para a realização de tarefas que possuem seus próprios custos. O processo é possível, por exemplo, por meio da *Edge Computing*, uma maneira de otimizar de sistemas de computação na nuvem, transferindo o processamento para as bordas do sistema, próxima à origem dos dados, conectando essa ideia à problemática de escalonar tarefas. Para realização da comparação, utilizou-se o iSPD (*iconic Simulator of Parallel and Distributed Systems*).

Assim, a partir dos modelos de sistema dos algoritmos e de suas descrições matemáticas, em conjunto com pesquisas que discorrem sobre o número de aparelhos IoT em residências, utilizou-se o iSPD para desenvolver o modelo de grade e o código dos escalonadores. Também foram escolhidos os valores de recurso dos sistemas e a metodologia de inserção de tarefas, com seus custos de recurso.

Com base no sistema criado, realizou-se testes de escalonamento com 100, 1.000, 10.000 e 100.000 tarefas. A observação dos resultados possibilitou uma análise gráfica dos códigos, assim como uma comparação entre seus tempos de simulação e suas eficiências médias. Determinou-se, assim, que o *Improved Binary Grey Wolf Optimizer* foi o melhor dentre os três algoritmos, pois levou o menor tempo para realizar as simulações, com a segunda melhor eficiência. Mas também foi possível observar que, a partir de 10.000 tarefas, o *Modied Monte Carlo Tree Search* começa a ter um desempenho similar, com uma eficiência maior. Esses resultados permitem concluir que não existe um algoritmo de escalonamento universal, perfeito para todas as situações. O que se verifica é que diferentes escalonadores são construídos para enfrentar diferentes situações-problema.

5.1 Trabalhos Futuros

A análise de algoritmos de escalonamento, em ambientes de computação de rede, pode ser aplicada em diferentes trabalhos futuros. Como exemplo, pode-se citar o desenvolvimento de outras políticas de escalonamento para ambientes *Edge Computing*, baseando-se em distintos algoritmos meta-heurísticos, como inteligência de enxame.

É possível, também, o surgimento de trabalhos que desenvolvam a comparação de políticas de escalonamento, utilizando paradigmas de computação distribuída diferentes, abordando o funcionamento desses códigos em ambientes de *Fog Computing* ou até em *Cloud*

Computing.

Outra possibilidade de estudo é a implementação, testagem e comparação de escalonadores e alocadores de máquinas virtuais interdependentes ou de tarefas que dependam uma das outras. Pode-se, ainda, utilizar outras ferramentas de simulação para a criação e testagem de sistemas de grade, que possam oferecer opções de configuração que não são encontradas no iSPD, como simuladores GridSim, SimGrid, FogSim e EdgeCloudSim.

Referências

- ARPACI-DUSSEAU, R. H.; ARPACI-DUSSEAU, A. C. *Operating Systems: Three Easy Pieces*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2018. ISBN 198508659X.
- BLAZEWICZ, J.; ECKER, K.; PESCH, E.; SCHMIDT, G.; WGLARZ, J. *Scheduling Computer and Manufacturing Processes*. [S.l.: s.n.], 2001. ISBN 978-3-642-07545-2.
- BUYYA, R.; MURSHED, M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. arXiv, 2002. Disponível em: <https://arxiv.org/abs/cs/0203019>.
- BAEWICZ, J.; ECKER, K. H.; PESCH, E.; SCHMIDT, G.; WGLARZ, J. *Scheduling Computer and Manufacturing Processes*. Springer Berlin Heidelberg, 2001. Disponível em: <http://dx.doi.org/10.1007/978-3-662-04363-9>.
- CASANOVA, H. Simgrid: a toolkit for the simulation of application scheduling. In: *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. [S.l.: s.n.], 2001. p. 430–437.
- COSTA, J. B. D. D.; SOUZA, A. M. de; ROSÁRIO, D.; SOMMER, C.; VILLAS, L. A. Efficient pareto optimality-based task scheduling for vehicular edge computing. In: *2022 IEEE 96th Vehicular Technology Conference (VTC2022-Fall)*. [S.l.: s.n.], 2022. p. 1–6.
- EMARY, E.; ZAWBAA, H. M.; HASSANIEN, A. E. Binary grey wolf optimization approaches for feature selection. *Neurocomputing*, v. 172, p. 371–381, 2016. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0925231215010504>.
- JAYASENA, K. N.; THISARASINGHE, B. Optimized task scheduling on fog computing environment using meta heuristic algorithms. In: *2019 IEEE International Conference on Smart Cloud (SmartCloud)*. [S.l.: s.n.], 2019. p. 53–58.
- JIANG, K.; NI, H.; SUN, P.; HAN, R. An improved binary grey wolf optimizer for dependent task scheduling in edge computing. In: *2019 21st International Conference on Advanced Communication Technology (ICACT)*. [S.l.: s.n.], 2019. p. 182–186.
- KNOB, L. A. D.; KAYSER, C. H.; FERRETO, T. Improving container deployment in edge computing using the infrastructure aware scheduling algorithm. In: *2021 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.: s.n.], 2021. p. 1–6.
- KOCSIS, L.; SZEPESVÁRI, C. Bandit based monte-carlo planning. In: FÜRNKRANZ, J.; SCHEFFER, T.; SPILIOPOULOU, M. (Ed.). *Machine Learning: ECML 2006*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 282–293. ISBN 978-3-540-46056-5.
- LIAO, J. X.; WU, X. W. Resource allocation and task scheduling scheme in priority-based hierarchical edge computing system. In: *2020 19th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*. [S.l.: s.n.], 2020. p. 46–49.

LIU, Y.; XIONG, G.; ZHU, F.; CHEN, S.; ZHANG, L.; LIU, X. Collaborative scheduling of computing tasks for edge computing. In: *2021 China Automation Congress (CAC)*. [S.l.: s.n.], 2021. p. 7824–7831.

MANACERO, A.; LOBATO, R.; OLIVEIRA, P.; GARCIA, M.; GUERRA, A.; AOQUI, V.; MENEZES, D.; DA, D.; UNIV, S.; ESTADUAL, P.-U. ispd: An iconic-based modeling simulator for distributed grids. *Simulation Series*, v. 44, 03 2012.

MIRJALILI, S.; MIRJALILI, S. M.; LEWIS, A. Grey wolf optimizer. *Advances in Engineering Software*, v. 69, p. 46–61, 2014. ISSN 0965-9978. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0965997813001853>.

OO, T.; KO, Y.-B. Application-aware task scheduling in heterogeneous edge cloud. In: *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. [S.l.: s.n.], 2019. p. 1316–1320.

PANCHALI, B. Edge computing- background and overview. In: *2018 International Conference on Smart Systems and Inventive Technology (ICSSIT)*. [S.l.: s.n.], 2018. p. 580–582.

PEIXOTO, M. L. M.; GENEZ, T. A. L.; BITTENCOURT, L. F. Hierarchical scheduling mechanisms in multi-level fog computing. *IEEE Transactions on Services Computing*, v. 15, n. 5, p. 2824–2837, 2022.

PHADINIS, S. Households have 10 connected devices now, will rise to 50 by 2020. 2016. Disponível em: <https://cio.economictimes.indiatimes.com/news/internet-of-things/households-have-10-connected-devices-now-will-rise-to-50-by-2020/53765773>.

PINEDO, M. L. *Scheduling: Theory, Algorithms, and Systems*. 5. ed. [S.l.]: Springer International Publishing, 2016. ISBN 978-3-319-26578-0, 978-3-319-26580-3.

RITTINGHOUSE, J.; RANSOME, J. *Cloud Computing: Implementation, Management, and Security*. 1st. ed. USA: CRC Press, Inc., 2009. ISBN 1439806802.

SITTÓN-CANDANEDO, I.; ALONSO, R. S.; CORCHADO, J. M.; RODRÍGUEZ-GONZÁLEZ, S.; CASADO-VARA, R. A review of edge computing reference architectures and a new global edge proposal. *Future Generation Computer Systems*, v. 99, p. 278–294, 2019. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X1930264X>.

WANG, H.; LI, C.; WANG, T.; HUANG, F. A job scheduling algorithm for edge computing based on modified monte carlo tree search. In: *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*. [S.l.: s.n.], 2021. p. 5–9.

WANG, L.; LASZEWSKI, G. von; YOUNGE, A.; HE, X.; KUNZE, M.; TAO, J.; FU, C. Cloud computing: a perspective study. *New Generation Computing*, v. 28, n. 2, p. 137–146, Apr 2010. ISSN 1882-7055. Disponível em: <https://doi.org/10.1007/s00354-008-0081-5>.

XIA, C.; LI, W.; CHANG, X.; DELICATO, F. C.; YANG, T.; ZOMAYA, A. Y. Edge-based energy management for smart homes. In: *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. [S.l.: s.n.], 2018. p. 849–856.