UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO – BCC

EDUARDO PANDINI

COMPARAÇÃO DE POLÍTICAS DE ESCALONAMENTO DE TAREFAS EM *CLOUD-EDGE CONTINUUM*

JOINVILLE
2024

EDUARDO PANDINI

COMPARAÇÃO DE POLÍTICAS DE ESCALONAMENTO DE TAREFAS EM *CLOUD-EDGE CONTINUUM*

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Guilherme Piêgas Koslovski

JOINVILLE
2024

EDUARDO PANDINI

COMPARAÇÃO DE POLÍTICAS DE ESCALONAMENTO DE TAREFAS EM *CLOUD-EDGE CONTINUUM*

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Guilherme Piêgas Koslovski

BANCA EXAMINADORA:

Orientador:

Dr. Guilherme Piêgas Koslovski
Universidade do Estado de Santa Catarina

Membros:

Dr. Charles Christian Miers
Universidade do Estado de Santa Catarina

Dr. Mauricio Aronne Pillon
Universidade do Estado de Santa Catarina

Joinville, Junho de 2024

Aos amigos dentro e fora da universidade, que acreditaram em mim quando eu deixei de acreditar.

“Até que durou” (Faria, [2018])

RESUMO

Edge Computing é um novo paradigma, no qual recursos de computação e armazenamento são posicionados nas bordas da Internet, próximos a sensores, dispositivos móveis e fontes de dados. Este paradigma é utilizado para prover serviços e processamento com menor latência e menor consumo de energia quando comparados aos processamentos realizados diretamente em *Cloud Computing*. Contudo, se tratando de serviços de borda, os recursos computacionais não possuem a mesma capacidade ou disponibilidade de recursos oferecidos pelas nuvens. Assim, é latente o desafio de como propriamente escalonar as tarefas dos usuários, selecionando adequadamente recursos de *Edge* e *Cloud Computing*. Para isto, são utilizadas políticas de escalonamento, as quais comparam diferentes aspectos do problema (tamanho do processamento, tempo de latência, possibilidade da divisão do processamento, etc) e tomam as decisões de como alocar as tarefas considerando os objetivos do ambiente (reduzir custos, melhorar qualidade do serviço, entre outros). Este trabalho realizou a comparação entre algumas destas políticas, e determinou, dados diversos aspectos, quais são as vantagens e desvantagens de cada uma em diferentes situações. A comparação foi realizada utilizando a simulação de um cenário composto por recursos de borda e de nuvem, denominado *Cloud-Edge Continuum*. Esta simulação foi feita utilizando o software EdgeSimPy, um simulador de redes *Edge* com as abstrações necessárias para realizar as comparações

Palavras-chave: *Cloud Computing, Edge Computing, Escalonamento, Continuum*

ABSTRACT

Edge Computing is a new paradigm, in which computing and storage resources are positioned on the physical borders of the internet, next to sensors, mobile devices and data sources. Such a paradigm is utilized to provide services and processing with lower latency and smaller energy consumption when compared to processing directly using Cloud Computing. However, being a border service, the computational resources do not posses the same capability or availability as resources offered by the Cloud. Thus, the challenge of how to properly schedule user tasks, appropriately selecting Edge and Cloud Computing resources, is latent. To that end, scheduling policies that compare different aspects of the problem (processing size, time, latency, possibility of division of provisioning, etc) are utilized, and take decisions on how to allocate tasks considering the objectives of the environment (reduce costs, better the quality of service, among others). This paper compared some of these policies, and determined, given diverse aspects, what are the advantages and disadvantages of each one in different situations. The comparison was be made using the simulation of a scenario composed by cloud and edge resources, denominated Cloud-Edge Continuum. This simulation was made using the software EdgeSimPy, an Edge network simulator with the necessary abstractions to accomplish these comparisons.

Keywords: *Cloud Computing, Edge Computing, Escalonamento, Continuum*.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AWS	*Amazon Web Services*
BF	*Best-Fit*
CE	*Cloud-Edge*
CEC	*Cloud-Edge Continuum*
CDF	*Cumulative Distribution Function*
DC	*Data Center*
DAG	*Directed Acyclic Graph*
EASY	*Easy Backfilling*
FCFS	*First Come First Served*
FF	*First-Fit*
FIFO	*First in First Out*
GVT	*Global Virtual Time*
JSON	*JavaScrypt Object Notation*
QOE	*Quality of Experience*
QoS	*Quality of Service*
RSU	*Roadside Unit*
RR	*Round-Robin*
SAF	*Shortest Area First*
SJF	*Shortest Job First*
SQF	*Shortest Number of Processors First*
SPA	*Shortest Path Algorithm*
SPT	*Shortest Processing Time first*
VANET	*Vehicular Ad-Hoc Network*
VANT	Veículo Aéreo Não Tripulado
WF	*Worst Fit*

SUMÁRIO

1 INTRODUÇÃO

Originalmente, uma nuvem computacional definiu o acesso a recursos geograficamente distribuídos, realizado através de uma rede de comunicação (MELL; GRANCE et al., 2011). Eventualmente, a necessidade de tempos de resposta eficientes, e menores custos de processamento levou ao desenvolvimento de centros de processamento mais próximos as fontes de dados e usuários, de modo a reduzir o tempo de resposta por conta da distância física, dando início a redes *edge*. A evolução no acesso aos dados, atualmente realizado de forma onipresente e ubíqua levou ao surgimento do paradigma de *cloud-edge Continuum*. Atualmente, por conta desta cooperação das redes *edge e cloud*, surge a latente necessidade de como realizar o escalonamento dos recursos disponíveis entre servidores *cloud e edge*. Este trabalho busca simular e avaliar técnicas para alocação de recursos e tarefas na colaboração *cloud-edge continuum*.

O escalonamento de tarefas é o processo de determinar a ordem de execução de diferentes tarefas ou processos em um sistema computacional, ou neste caso, em uma rede. O objetivo é otimizar o desempenho do sistema, minimizar o tempo de resposta e maximizar a utilização dos recursos disponíveis. Existem diversos algoritmos de escalonamento que podem ser empregados, cada um com suas características específicas. A realização deste escalonamento de tarefas e recursos é feita por políticas de escalonamento, uma função que, utilizando diferentes métricas e parâmetros previamente escolhidos, determina quais e quando disponibilizar os recursos disponíveis na rede. Para análise, adequação, estudo e evolução destas políticas de escalonamento, o uso de simulador é preferível a testes feitos em redes reais, por conta do custo e tempo necessários para reunir uma quantidade significativa de dados. Um grande intervalo de tempo pode ser simulado e analisado rapidamente utilizando simulações numéricas.

Diferentes políticas atendem necessidades diferentes, portanto, o estudo e seleção das políticas de escalonamento é desejável para permitir a seleção de uma política que melhor satisfaça os requerimentos de um usuário, seja ele economia de preço ou energia, redução do tempo de resposta, entre outros. É necessário que se realizem testes nas políticas de escalonamento selecionadas, para averiguar sua eficiência, estabilidade e funcionamento próprio. Contudo, realizar estes testes em redes reais em operação, é não só impraticável mas também altamente custoso e lento. Para contornar estas dificuldades, é feito o uso de simulações. Para simular um sistema real, primeiro é necessário transferir esse sistema para um modelo. Por isso, algumas suposições sobre sistema real são realizadas, gerando relacionamentos matemáticos, lógicos e outros para constituir um modelo que é então usado para compreender o comportamento abrangente do sistema. (TANG; LEU; ABBASS, 2020)

Para realização deste trabalho, foi utilizado o software EdgeSimPy, um simulador de redes *edge* desenvolvido em Python por (SOUZA; FERRETO; CALHEIROS, 2023). Trata-se de um *framework* para modelar e simular políticas de escalonamento de recursos em redes *edge*. Construída em uma arquitetura modular composta de abstrações de funcionalidades de servidores *edge*, dispositivos de rede e aplicações. Foi selecionado por conta de sua arquitetura

modular, provendo facilidade de implementação de diferentes políticas, algoritmos de mobilidade e configurações de cenário.

1.1 OBJETIVO GERAL

O objetivo geral do presente trabalho é realizar a simulação e comparação de algumas políticas de escalonamento em diferentes cenários utilizando o software EdgeSimPy, utilizando métricas de avaliação de desempenho, custo e tempo de resposta.

Com base no objetivo geral, os seguintes objetivos específicos são definidos:

1. Estudar redes *edge, cloud*, e sua colaboração;
2. Estudar a ferramenta de simulação;
3. Modelar e simular políticas de escalonamento;
4. Avaliar os resultados das simulações.

1.2 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado da seguinte forma: O Capítulo 2 apresenta a revisão de literatura, descrevendo as redes *edge, cloud e fog* assim como a colaboração entre estas redes. É elaborado sobre a arquitetura colaborativa. Contextualiza também o que é uma tarefa e o que é escalonamento, e porque este é importante em redes *cloud-edge*. Também são elencadas algumas das políticas existentes. Por fim são descritos alguns dos trabalhos relacionados ao tópico. O Capítulo 3 apresenta os principais conceitos necessários para uma simulação e alguns dos diferentes simuladores para redes *edge*. É também explicado o funcionamento, configuração e arquitetura do simulador EdgeSimPy, este que será utilizado para as simulações deste trabalho. O Capítulo 4 apresenta a implementação dos algoritmos para simulação, descrevendo os modelos utilizados nas simulações O capítulo 5 apresenta os cenários utilizados nas simulações, os dados obtidos e uma discussão sobre os resultados, além disso apresenta observações sobre a utilização do simulador.

Por fim, o capítulo 6 apresenta as conclusões finais do trabalho.

2 REVISÃO DE LITERATURA

No presente capítulo são apresentados os diferentes modelos de acesso e processamento de dados em *Cloud*, *Fog* e *Edge Computing*, de forma sequencial através da evolução do tempo. Posteriormente, as aplicações que utilizam as infraestruturas são apresentadas, bem como as definições sobre escalonamento de tarefas, e uma explicação de funcionamento de políticas de escalonamento proeminentes.

2.1 EVOLUÇÃO DE *CLOUD* PARA *CLOUD-EDGE CONTINUUM*

***Cloud Computing*.** Segundo (MELL; GRANCE et al., 2011), autores do Instituto Nacional de Padrões e Tecnologia Norte Americano, *Cloud Computing* é um modelo para permitir acesso onipresente, conveniente e sob demanda a uma rede compartilhada de recursos computacionais, que podem ser provisionados e liberados rapidamente com o mínimo de esforço de gerenciamento ou interação com o provedor de serviços. A centralização destes recursos em *datacenters* e grandes armazéns permite processamento e armazenamento localmente rápido, contudo, em muitas ocasiões, estes *datacenters* estão localizados longe de grandes centros urbanos ou áreas com alta demanda de processamento.

***Edge Computing*.** Por outro lado, *Edge Computing*, ou computação de borda, se refere a borda geográfica, na qual ao invés do processamento ser feito em um *datacenter* distante (ou parcialmente próximo, como no caso proposto pela *Fog Computing*), existem recursos de processamento menores, mais perto dos usuários ou fontes de dados (LOPEZ et al., 2015). Em diversos casos, as bordas podem utilizar o poder de processamento, armazenamento e comunicação localmente disponíveis. Contudo, se tratando de serviços de borda, os recursos computacionais não possuem a mesma capacidade ou disponibilidade de recursos oferecidos pelas nuvens. Assim, é latente o desafio de como propriamente escalonar as tarefas dos usuários, selecionando adequadamente recursos de *Edge* e *Cloud Computing*.

***Fog Computing*.** Como definido por (IORGA et al., 2018), é um modelo em camadas que permite acesso ubíquo a um *continuum* escalável de recursos computacionais. O modelo facilita o lançamento de aplicações e serviços distribuídos e de baixa latência. As principais diferenças entre *Edge* e *Fog* são seu escopo e localização. *Fog computing* é um método mais distribuído e colaborativo, tendo vários nós de processamento entre o usuário e os servidores em nuvem.

***Cloud-Edge Continuum*.** Pela inviabilidade de manter todos os serviços na borda ou nas nuvens, a colaboração entre várias partes (provedores, serviços, entre outros) constitui um cenário denominado *Cloud-Edge Continuum*. Na arquitetura, como descrita por (LUO et al., 2021) e exemplificada pela Figura 1, uma rede de computação de borda heterogênea em três níveis é composta. A primeira camada é o camada das coisas *(things)*, a segunda camada é a camada da borda *(edge)* e a terceira camada é a camada da nuvem *(cloud)*.

Dentro desta arquitetura, diferentes modos de colaboração entre níveis são possíveis, sendo elas: *thing-edge*, na qual ocorre a colaboração do usuário, ou a fonte de dados com a borda.

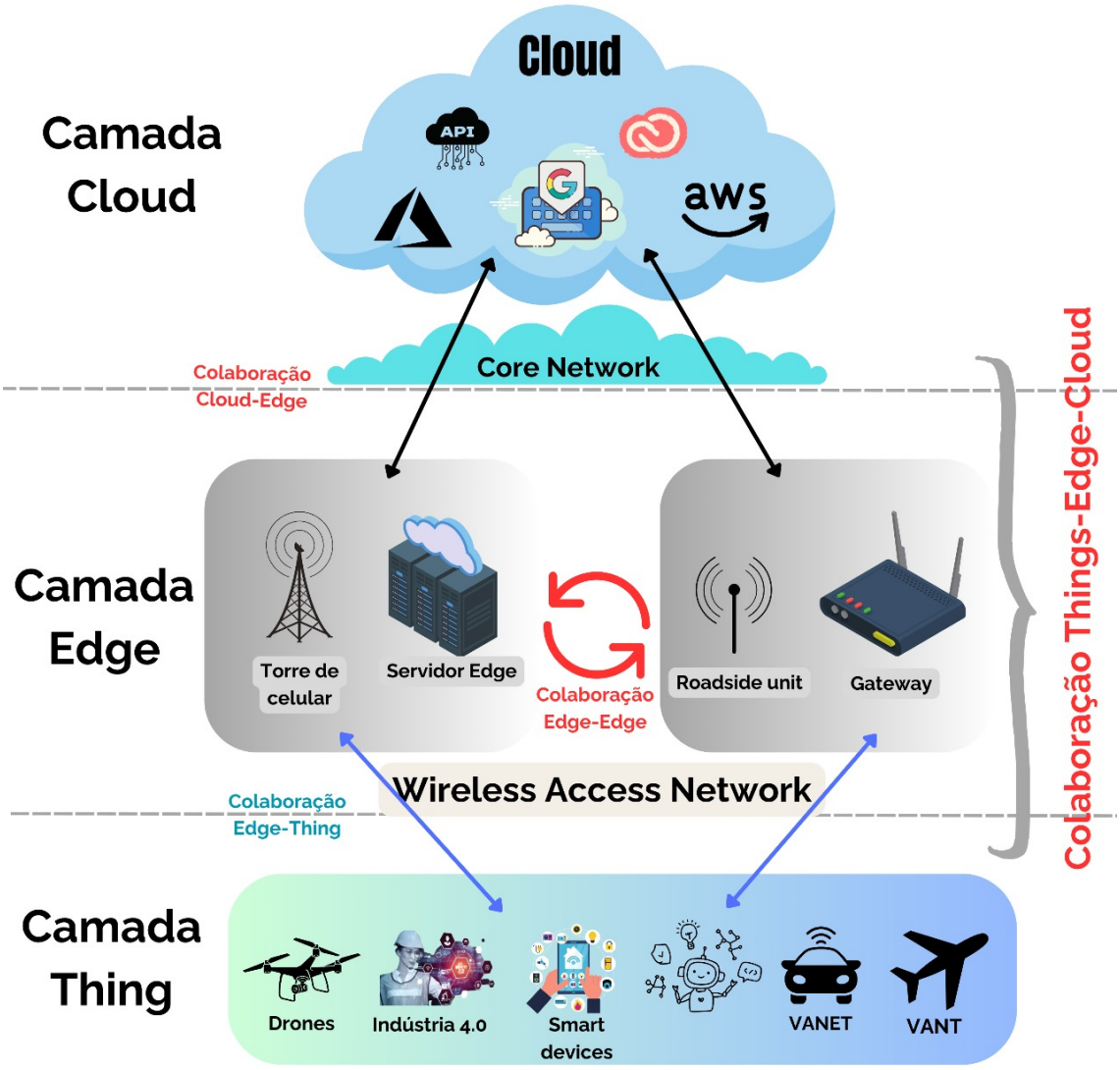


Figura 1 – Representação da arquitetura de *Cloud-Edge Computing* Baseado em: (LUO et al., 2021)

Geralmente se tratam de tarefas com menor demanda de processamento, sendo majoritariamente criadas por *smart devices*, podendo ser processadas localmente ou na borda. *Things-cloud-edge*, com a colaboração das três camadas, conta com a imensa capacidade de processamento da nuvem, sendo capaz de processar tarefas com mais demanda computacional ou de armazenamento. *Edge-edge* na qual diferentes componentes de *edge* colaboram entre si; e por fim *cloud-edge*. Utilizando estas colaborações é possível ter os benefícios de cada camada envolvida, enquanto são mitigados seus pontos fracos. O foco deste trabalho está na colaboração *cloud-edge*, que trata da colaboração entre as camadas *Cloud* e *Edge*.

Domínios administrativos. Enquanto *Cloud* e *Fog Computing* propuseram o gerenciamento de recursos realizado pelos provedores, *Edge Computing* permite que o usuário final (*e.g.*, uma empresa, universidade, residência) participe das decisões gerenciais. Assim, *Cloud-Edge Continnum* requer um gerenciamento combinado, pois as aplicações elencadas são executadas

de forma combinada sobre múltiplos recursos.

2.2 APLICAÇÕES E TAREFAS

No contexto de *cloud-edge Continuum*, o termo tarefa geralmente se refere a dados gerados ou requeridos pelo usuário, podendo variar de acordo com a aplicação e objetivos do sistema final. Tarefas realizadas em redes de *cloud* cobrem diversas funcionalidades, como armazenamento de dados, processamento de informação, hospedagem de serviços, controle de dispositivos IoT, cache de vídeo, etc. Além das tecnologias atualmente hospedadas em *clouds* que podem se beneficiar do uso de *edge*, diversas aplicações emergentes são elencadas. Um exemplo de tais aplicações são as *Vehicular Ad-Hoc Network*s (VANETs), nas quais os veículos comportam-se como fontes de dados, coletando informações nas estradas, quanto dispositivos de transmissão, levando dados próprios ou de outros veículos para *Roadside Unit*s (RSUs), sendo estas os servidores de *edge* do sistema, coletando, processando e enviando dados de volta aos veículos. Todo este processo se dá automaticamente enquanto o usuário dirige, sem necessitar de ações adicionais ou até mesmo conhecimento de que isto está ocorrendo. Estas redes possuem diversos usos, abrangendo informações de trânsito, compartilhamento de mídia e até mesmo acionamento de serviços de emergência. (PANDINI et al., 2022)

Contextualizando estas tarefas, o controle de dispositivos IoT pode ser realizado na *edge*, processando e enviando comandos sensíveis ao tempo, enquanto o processamento e avaliação de dados em larga escala pode ser realizado na *cloud*. Desta forma seriam utilizados ambos os vastos recursos da *cloud* e a rápida resposta da *edge*. Pode-se também citar a criação de *mini-datacenters* especializados (*e.g.*, com recursos para processamento de imagens, alto desempenho), próximos das fontes de dados e clientes, podendo utilizar a *cloud* para somente armazenar dados menos utilizados, ou para realização de *backups*. Atualmente um dos maiores provedores de acesso a serviços *edge* é a *Amazon Web Services* (AWS), formada pelo estabelecimento de parcerias com diversas companhias em diferentes países, disponibilizando numerosos serviços, abrangendo armazenamento, entrega de conteúdo, *machine learning*, e diversas ferramentas para o uso de dispositivos IOT, com latência inferior a 10 milissegundos. Ademais, *Distributed Cloud Edge* da Google usa a *Google Kubernet Engine* em hardware dedicado, com a mesma finalidade de fornecer conexões estáveis de baixa latência. De maneira semelhante, Microsoft Azure é uma plataforma que utiliza processamento tanto em *cloud* quanto *edge*, assim como a colaboração entre ambos.

2.3 ESCALONAMENTO EM *CLOUD-EDGE CONTINUUM*

De acordo com (LUO et al., 2021), se a maioria das tarefas são realizadas de forma centralizada na *cloud*, dentro da arquitetura de três camadas, alta latência será produzida, o que não irá satisfazer a *Quality of Experience* (QoE) do usuário final. O problema de alta latência pode ser melhorado enviando algumas ou todas as tarefas para a borda. Esta distribuição é o que

cria o *continuum*, que encurta as distâncias entre *datacenters* e usuários. Esta divisão é chamada de escalonamento de tarefas. Especificamente, o escalonamento de tarefas em *Cloud-Edge Continuum* (CEC) pode ser definido como o conjunto de ações e metodologias que participantes usam para alocar de forma efetiva os recursos para uma tarefa que precisa ser completa, e atingir os objetivos dos participantes baseado em disponibilidade de recursos.

Para determinar a viabilidade ou otimização de um modelo, podem ser utilizadas diversas métricas, sendo algumas delas: consumo de energia, custo, tempo de processamento, tempo de envio/recebimento e lucro. É chamada de política de escalonamento o conjunto de regras que utilizam estas variáveis para realizar a determinação da forma otimizada de alocação dos recursos. Esta determinação depende de diversos aspectos em conjunto, como tamanho das tarefas, a possibilidade de serem divididas, a distância entre usuários e centros de processamento, a capacidade de processamento de cada camada da arquitetura, entre outros, portanto, é operático determinar manualmente qual política irá ter melhor desempenho para cada aspecto. Como apontado por (SOUZA; FERRETO; CALHEIROS, 2023) pesquisadores e praticantes ainda enfrentam barreiras enquanto projetam políticas de escalonamento de recursos para borda.

Um fluxo sequencial de processamento é decomposto em tarefas, cujo objetivo é atender uma finalidade específica, como por exemplo realizar um cálculo complexo, a renderização de um vídeo, entre outros. Formalmente, uma tarefa representada por t, processa um determinado volume de dados (tamanho t_d) utilizando um conjunto t_r de recursos computacionais (*e.g.*, ciclos de CPU). As aplicações, decompostas em tarefas, devem ser escalonadas para executar utilizando os recursos computacionais das bordas e da nuvem. Assim, um recurso computacional é representado por p, possuindo um determinado poder computacional máximo (p_r), bem como um espaço máximo de armazenamento (p_d).

2.4 POLÍTICAS DE ESCALONAMENTO

Formalmente, uma política de escalonamento deve ordenar a execução das tarefas, respeitando os requisitos indicados pelas aplicações e os recursos disponíveis na *edge* e na *cloud*. Ou seja, considerando A o conjunto de tarefas atualmente escalonadas no recurso p, é necessário respeitar o limite de armazenamento e processamento, ou seja, $p_r \geq \sum_{t \in A} t_r$ e $p_d \geq \sum_{t \in A} t_d$.

Nesta seção são elencadas e explicadas algumas das políticas de escalonamento mais proeminentes e comumente aplicadas (pela simplicidade de implementação ou eficiência em cenários específicos). É importante ressaltar que as políticas selecionadas representam um ponto de partida para o estudo. Posteriormente, novas políticas (complexas) poderão ser adicionadas.

- *First-Fit* (FF): Na política de FF, as tarefas são realizadas por ordem de requisição, sendo alocados por ordem de chegada para o primeiro recurso que tiver capacidade disponível suficiente.

Variável	Significado
A	Conjunto de tarefas
t	Uma tarefa
t_d	Tamanho da tarefa
t_r	recursos computacionais necessários na tarefa
p	Recurso computacional
p_r	Poder computacional máximo
p_d	Espaço máximo de armazenamento

Tabela 1 – Variáveis utilizadas.

- *Best-Fit* (BF): A política de BF aloca as tarefas para recursos que tenham capacidade semelhante a necessidade da tarefa, ou seja, dada uma tarefa t, e diversos recursos p, a tarefa será alocada para o recurso que possuir p_r ou p_d disponíveis mais próximos da necessidade de A_r ou A_d.

- *Worst Fit* (WF): A política WF tem o comportamento oposto da política de BF, cujo WF procura alocar as tarefas aos recursos com mais disparidade entre p_r ou p_d e A_r ou A_d, como a tarefa não pode ser alocada a recursos que possuem p_r ou p_d menor que os necessários para a tarefa, na prática, WF sempre aloca a tarefa ao recurso com mais p_r ou p_d disponível.

- *Round-Robin* (RR): A política RR funciona definindo um valor chamado *quantum*, o qual é utilizado para escalonar diferentes tarefas utilizando o mesmo recurso, fazendo um balanceamento de carga. RR tem uma fila por ordem de chegada das tarefas. O valor do *quantum* definido indica quanto tempo ou quantos ciclos de processamento um programa pode usar, antes de liberar o recurso para outra tarefa. Após o fim do *quantum*, caso a tarefa ainda não tenha sido finalizada, ela passa para o final da fila e espera sua vez novamente.

- *Shortest Job First* (SJF): Na política SJF, a tarefa na fila com menor A_r ou A_d necessários para executar a tarefa. É necessário avaliar o sistema antes de aplicar a política de SJF, pois caso o sistema esteja constantemente recebendo novas tarefas, uma tarefa grande pode nunca ser realizada.

- *First in First Out* (FIFO): O critério de ordem de requisição, em que a tarefa que chegou primeiro, será alocada primeiro.

2.5 CONSIDERAÇÕES PARCIAIS

Neste capítulo foram apresentadas as definições de *Cloud-Computing*, *Edge-Computing*, *Fog-Computing*, *cloud-edge Continuum*, suas aplicações e tarefas, assim como exemplos de provedores destes serviços. Também é contextualizado o escalonamento destas tarefas em *Cloud-Edge* e são elencadas e explicadas algumas políticas de escalonamento mais utilizadas. É então levantado o problema de como e por qual motivo realizar o escalonamento de tarefas em *cloud-edge Continuum*.

O próximo capítulo define e explica o funcionamento do software EdgeSimPy, que será utilizado para realizar as simulações das políticas de escalonamento.

3 SIMULAÇÃO DO ESCALONAMENTO EM *CLOUD-EDGE CONTINUUM*

No presente capítulo é dada uma breve explicação do que é uma simulação e seus componentes básicos. É então apresentado o simulador EdgeSimPy, que será utilizado para realizar as simulações das políticas de escalonamento. Seu funcionamento é detalhado e são explicados os elementos essenciais para o uso neste trabalho.

3.1 SIMULAÇÃO

Para definir como ocorrerá a simulação, é necessário formalizar os termos modelo, simulador e simulação (TANG; LEU; ABBASS, 2020). Inicialmente, um modelo é uma representação estática do sistema real. Por sua vez, um simulador codifica o modelo, enquanto uma simulação é um processo geral que usa o simulador com entradas amostradas para reproduzir o comportamento do sistema real. Para simular um sistema real, primeiro é necessário transferir esse sistema para um modelo. Por isso, algumas suposições sobre sistema real são realizadas, gerando relacionamentos matemáticos, lógicos e outros para constituir um modelo que é então usado para compreender o comportamento abrangente do sistema. O modelo é o componente fundamental de uma simulação.

Ademais, conforme indicado nos objetivos específicos do presente trabalho, para o entendimento das simulações, é necessário abordar alguns conceitos:

- Cenário: corresponde a representação de um sistema, ou seja, os elementos e entidades do sistema são mapeados e convertidos para elementos equivalentes no cenário. O cenário deve ser capaz de representar diferentes tipos de entradas para diferentes situações do sistema, assim como ser capaz de interpretar seus efeitos e particularidades. Para criação de um cenário, estes dados devem ser convertidos para entradas válidas para o simulador, como por exemplo: quantidade de arquivos, tamanho de arquivos, tempo de envio e recebimento, distância entre dados e recursos, etc.

- Simulador: É o software ou linguagem utilizados para utilizar este cenário, com o intuito de realizar uma simulação.

- Simulação: É o processo no qual o simulador é manipulado dado um cenário qualquer, para gerar dados de simulação baseados no cenário.

- *Global Virtual Time* (GVT): é um conceito fundamental de sincronização em simulações. É definido como o menor intervalo de tempo dentro do conjunto de eventos não processados pendentes em uma simulação distribuída (STEINMAN et al., 1995). Em outras palavras, semelhante a um *clock* operando em *ticks*. Um intervalo de tempo arbitrário definido para a simulação, no qual tarefas e eventos são simulados, é representado por ∇. É utilizado para manter a coerência de tempo da simulação, por exemplo, caso seja

definido um ∇ como representando 1 minuto, é indiferente se a simulação durar um segundo ou uma hora, cada intervalo de tempo será equivalente a um minuto.

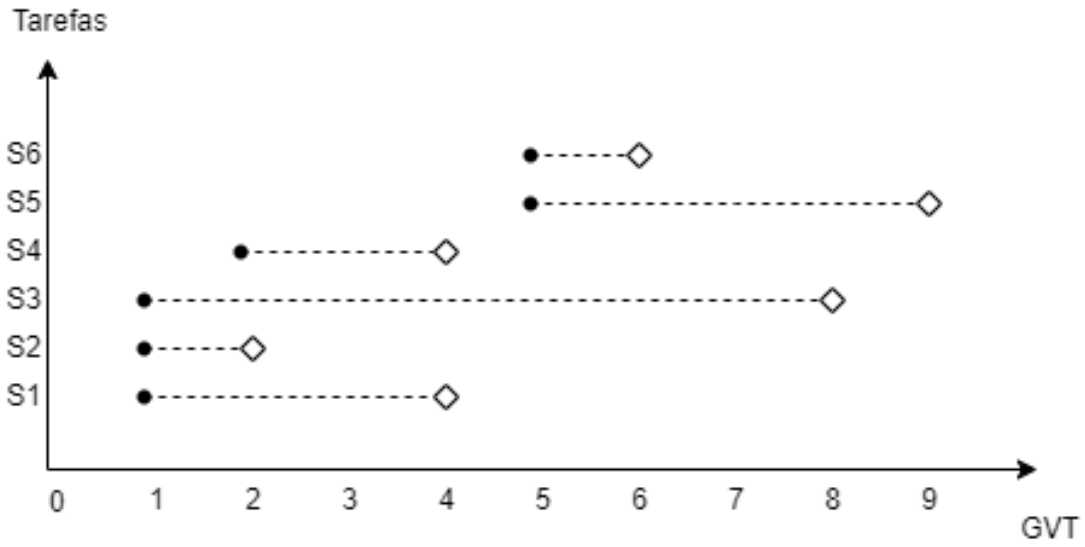


Figura 2 – Representação básica de uma simulação.

A Figura 2 apresenta um exemplo básico do funcionamento de uma simulação. É representado um cenário com seis tarefas, inicializadas em diferentes instantes da simulação, executados por um intervalo de tempo e finalizados. No instante um do GVT, três tarefas são requisitadas, neste momento a simulação os escalona utilizando a política programada no simulador, e as tarefas começam a ser processadas. No instante dois, mais uma tarefa é requisitada e começa a ser alocada e realizada, além disto, a tarefa S2 é finalizada, e os recursos alocados para sua realização são liberados. No instante quatro, as tarefas S1 e S4 são finalizadas e têm seus recursos liberados. No instante cinco mais duas tarefas são requisitadas. Nos instantes seis, oito e nove, as tarefas S6, S3 e S5 são finalizadas.

É importante ressaltar que durante o escalonamento, os requisitos de computação, armazenamento e comunicação devem ser satisfeitos, ou seja, as tarefas devem ser escalonadas respeitando as especificações do usuário e os recursos disponíveis no sistema. Por fim, como a simulação está utilizando GVT, a simulação utilizou nove unidades de tempo ∇. Por exemplo, ao definir ∇ como representando uma hora, a simulação seria equivalente a nove horas no sistema original, mesmo que para simular estes dados o simulador tenha usado menos ou mais tempo que isto.

3.2 SIMULADOR EDGESIMPY

É notável a necessidade do uso eficiente de recursos, portanto, isto também se torna um desafio. Políticas novas (ou refinamentos em consolidadas) precisam ser desenvolvidas ou integradas em sistemas *Cloud-Edge* (CE), e consequentemente testadas em tais redes, para

garantir seu funcionamento e eficiência. Contudo, além do custo e tempo necessários para realizar estes testes, a natureza distribuída das estruturas *edge* impõem barreiras adicionais para experimentos empíricos das políticas de escalonamento nas redes. Exemplos das dificuldades são instabilidade nas redes ou falta de energia durante testes. (SOUZA; FERRETO; CALHEIROS, 2023) Estes fatores deram motivação para o surgimento de diversos simuladores de redes *edge*, visando reduzir custos e tempo necessários, assim como facilitar o teste e prototipação de projetos.

Simulador	Energia do servidor	Energia da rede	Roteamento de rede	Mobilidade de usuário	Composição de aplicação	Ciclo de vida do contêiner
EdgeCloudSim	x	x	x	✓	x	x
FogNetSim++	✓	✓	✓	✓	x	x
MobFogSim	✓	x	x	✓	x	x
ECSNeT++	✓	✓	✓	✓	✓	x
YAFS	✓	x	✓	✓	✓	x
IoTSim-Edge	✓	x	✓	✓	✓	x
IoTSim-Osmosis	✓	x	✓	x	✓	x
IFogSim2	✓	x	✓	✓	✓	x
EdgeSimPy	✓	✓	✓	✓	✓	✓

Tabela 2 – Simuladores e funções.

Referencias	Simulador	Migração de serviço	Escalonamento de fluxo de rede	Gerenciamento de registro de contêiner	Operações de manutenção
(SONMEZ; OZGOVDE; ERSOY, 2018)	EdgeCloudSim	x	x	x	x
(QAYYUM et al., 2018)	FogNetSim++	x	✓	x	x
(PULIAFITO et al., 2020)	MobFogSim	✓	x	x	x
(AMARASINGHE et al., 2020)	ECSNeT++	x	✓	x	x
(LERA; GUERRERO; JUIZ, 2019)	YAFS	x	x	x	x
(JHA et al., 2020)	oTSim-Edge	x	✓	x	x
(ALWASEL et al., 2021)	IoTSim-Osmosis	x	✓	x	x
(MAHMUD et al., 2022)	iFogSim2	✓	✓	x	x
(SOUZA; FERRETO; CALHEIROS, 2023)	EdgeSimPy	✓	✓	✓	✓

Tabela 3 – Casos de uso.

A Tabela 2 lista os diferentes simuladores desenvolvidos para utilização em projetos de CE, assim como um sumário de funções disponíveis em cada um deles. As funcionalidades de energia de servidor e rede são úteis para uso como critério de avaliação, pois podem ser utilizados para diretamente avaliar a qualidade de uma política em quesito de economia de energia.

De forma complementar, a Tabela 3 mostra exemplos de possíveis casos de uso para cada um dos simuladores testados. É importante a capacidade do simulador representar casos como migrações de serviços e operações de manutenção é gerar situações adversas na rede, e testar o comportamento das políticas nestas ocasiões.

A proposta deste trabalho é a realização de diversas simulações, muitas vezes com políticas, métricas e cenários diferentes, portanto é necessário um simulador capaz de realizar essa variedade de mudanças, de maneira fácil e rápida, idealmente de forma modular.

Diante da análise realizada, para realização deste trabalho, será utilizado o software EdgeSimPy, um simulador de redes *edge* desenvolvido em Python por (SOUZA; FERRETO; CALHEIROS, 2023). Trata-se de um *framework* para modelar e simular políticas de escalonamento de recursos em redes *edge*. Construída em uma arquitetura modular composta de abstrações de funcionalidades de servidores *edge*, dispositivos móveis participantes na rede e outros componentes.

O objetivo principal do EdgeSimPy é dar suporte a pesquisadores interessados em avaliar estratégias de escalonamento de recursos em infraestruturas de *Edge computing*. EdgeSimPy suporta diferentes modelos de alocação de recursos, enquanto considera as diferentes situações no contexto de *edge*, como consumo de energia, capacidade de processamento, mobilidade, requerimentos de performance, etc. O simulador irá oferecer uma base para simulação, contudo poucas políticas já estão implementadas em suas bibliotecas, sendo elas a aleatória e FIFO.

O EdgeSimPy supera as limitações de antigos simuladores de redes de borda existentes, tendo uma arquitetura modular que inclui abstrações para servidores de borda, dispositivos de redes, aplicações e usuários. Possui também um modelo flexível de entrada que permite ao usuário definir parâmetros personalizados para entidades simuladas, dando enorme costumabilidade as simulações realizadas.

3.2.1 Funcionamento do simulador

Para executar uma simulação em EdgeSimPy, o simulador necessita de um arquivo de entrada definindo o cenário. Os arquivos de entrada para EdgeSimPy são no formato *JavaScript Object Notation* (JSON), o que facilita a integração de EdgeSimPy com outras ferramentas. Os arquivos de entrada então organizam os metadados de cada entidade da simulação em uma estrutura composta de dois grupos de informação:

- Atributos: Se referem a características internas de cada entidade, como por exemplo, a capacidade de um dado servidor edge, banda larga da conexão, latência da conexão, entre outros.

- Relacionamentos: Representa a associação entre as entidades, se existentes.

A Figura 3 demonstra um exemplo de um arquivo de entrada contendo os metadados para uma entidade qualquer. Entidades simuladas podem também conter metadados geo-espaciais, que facilitam a integração de *datasets* contendo coordenadas reais ou imaginárias, e permite a simulação de eventos como mobilidade de usuários. Por padrão é utilizado o modelo proposto por (ARAL; MAIO; BRANDIC, 2021), que divide o ambiente em células hexagonais.

Quando a simulação de fato inicia, um mecanismo de monitoramento é ativado, o qual guarda uma *snapshot* do estado da entidade a cada passo da simulação. O histórico da simulação é salvo em MessagePack, um formato de serialização binário. Ao invés de salvar no disco a cada passo, o *output* é salvo em intervalos configuráveis durante a simulação.

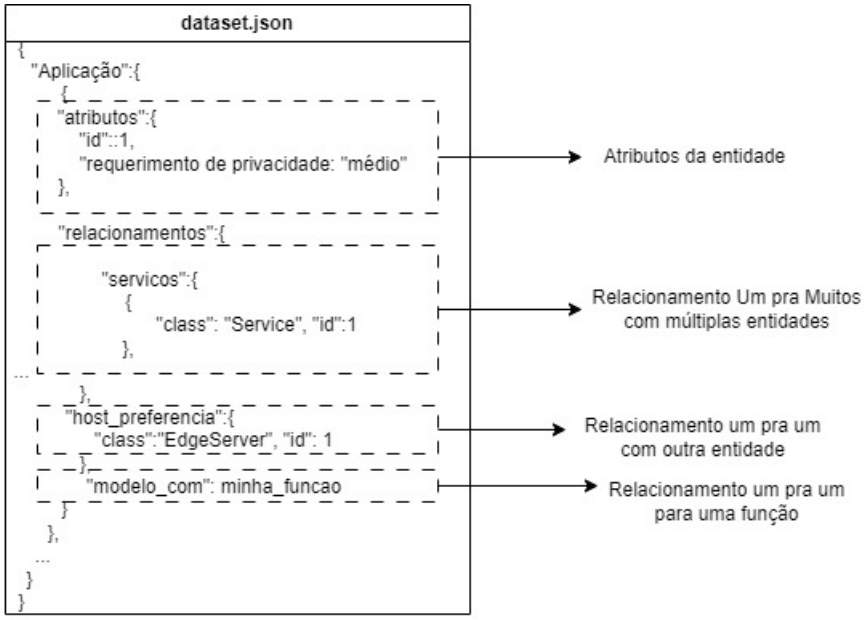


Figura 3 – Exemplo básico de um arquivo de entrada JSON para o EdgeSimPy.

3.2.2 Decomposição em Camadas

A flexibilidade do simulador é criada pela arquitetura modular que divide as abstrações em quatro camadas, os componentes e funções de cada uma das camadas serão explicados nessa seção.

3.2.2.1 Camada do núcleo:

É a camada responsável pelo gerenciamento da simulação em passos representados pelo Fluxograma 4 incremento do GVT, ativação de funções e tratamento de propriedades emergentes. O fluxo de funcionamento do simulador, na Figura 4 demonstra as ações tomadas

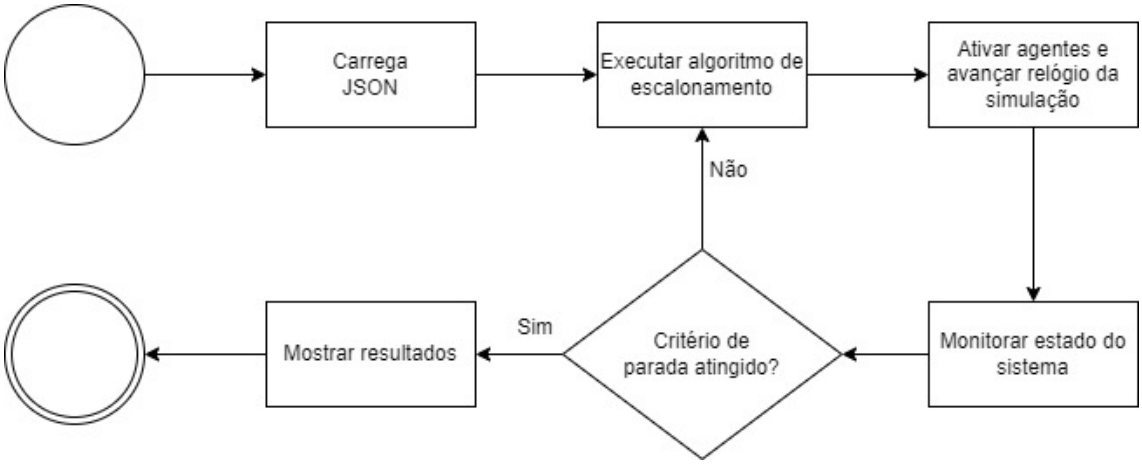


Figura 4 – Fluxograma de uma simulação em EdgeSimPy.

pelo simulador a cada passo da simulação: é iniciada com o carregamento do *dataset*, seguido de um processo iterativo de simulação e monitoramento, até que o critério de parada determinado

pelo usuário seja atingido. A cada passo a política de escalonamento definida é executada, o GVT é incrementado e dados são coletados sobre o estado do sistema. Quando o critério de parada é atingido, a simulação é encerrada e as métricas e dados coletados durante a simulação são mostrados.

3.2.2.2 Camada física

É a camada que contém as abstrações de usuários e recursos que compõe a infraestrutura *edge*. Independente de quais as suas funções e atributos, todos os elementos da camada física possuem um atributo de coordenada, que carrega a informação geo-espacial do componente.

Os componentes da camada física são:

- Estações base: Funcionam como os pontos de entrada para a rede *edge*, permitindo comunicação entre usuários e servidores. O simulador assume que estações base cobrem toda a área simulada, de modo que usuários sempre possam ter conexão, independente de sua posição geográfica. Por esta presunção, para quesitos de simulação, usuários não podem estar em uma posição não coberta por pelo menos uma estação base. Estações base possuem diversos atributos customizáveis para a simulação, como consumo de energia e latência, permitindo mudar os dados para diferentes necessidades de teste.

  O simulador lida automaticamente com a mobilidade de usuário, passando a conexão de uma base para outra, de acordo com a posição do usuário, permitindo uma simulação mais realista de movimentos e mudanças durante trânsito de uma base para outra.

- *Network Switches*: São os componentes que provêm conectividade cabeada entre componentes da rede. Modelados como nós no grafo que representa a topologia da rede, são configurados com vários parâmetros, como número de portas, *delay* e banda larga. Apesar de que os *switches* são os nós por padrão, outras entidades podem ser configuradas, como por exemplo, veículos, em redes veiculares, ou Veículo Aéreo Não Tripulados (VANTs) para coberturas móveis. O simulador assume que os requerimentos dos usuários são protegidos por políticas de *Quality of Service* (QoS), portanto o requerimento de um usuário não afeta negativamente os outros. Por outro lado, grandes transferências de dados entre servidores são tratados como fluxo de rede, compartilhando a banda larga de nós da rede, cuja duração depende da banda larga dos nós, e sua disponibilidade segundo a política empregada. Quando um fluxo de rede inicia ou acaba, o simulador executa um algoritmo de escalonamento para atualizar a ocupação dos nós envolvidos, redistribuindo quando aplicável a banda larga da rede.

- *Edge Servers:* Servidores *edge* são utilizados para hospedar serviços. O simulador assume que a infraestrutura é virtualizada, de modo que consiga hospedar múltiplos serviços simultaneamente. Possui parâmetros como CPU, RAM, memória de disco e Instruções por segundo. Especificações como modelos de *hardware* afetam o consumo de energia

em uma simulação que esteja analisando tais parâmetros. O simulador permite implementação do desligamento temporário de servidores para economizar energia.

- Usuários: Como parte da camada física, todos os usuários têm um atributo de coordenadas, que define seu lugar no mapa. Usuários podem permanecer estáticos durante toda a simulação ou se moverem de acordo com modelos de mobilidade. Por padrão são incorporados dois modelos: *Random e pathway*, os quais podem ser trocados por outros modelos de mobilidade. Usuários e aplicações são ligados por relações de um-para-muitos, o que permite que um usuário acesse vários serviços, ou que um serviço seja acessado por vários usuários. Deste modo cada usuário tem parâmetros que definem a latência e disponibilidade para cada aplicação que estes acessam. Para determinar qual aplicação cada usuário utiliza, o simulador incorpora dois *templates* de padrão de acesso: aleatório e circular. Enquanto o aleatório determina o acesso de forma arbitrária, o circular estabelece um padrão que repete indefinidamente.

3.2.2.3 Camada lógica

A camada lógica é composta pelas abstrações funcionais das aplicações sendo executadas pela infraestrutura *edge*, seus componentes são:

- Aplicações: Aplicações são modeladas como entidades abstratas, representando fluxos de dados envolvendo múltiplos serviços, tendo como atributos, por exemplo, prioridade e orçamento disponível. O simulador calcula a latência de uma aplicação qualquer baseado no tempo necessário para visitar todos os servidores necessários para realizar a tarefa. a aplicação padrão assume que o fluxo de dados inicia no usuário e passa por todos os serviços necessários sequencialmente.

- Serviços: Serviços são modelados como instâncias de contêineres dentro da infraestrutura, tendo atributos como demanda de disco e memória. Fazem uso de uma infraestrutura em camadas para separar seus componentes, para garantir que mudanças em um serviço não afetem serviços hospedados no mesmo servidor.

- *Container Registries:* O principal componente quando alocado um serviço na infraestrutura *edge*, tendo que a imagem do contêiner do serviço é extraída do servidor. Desta forma os registros também possuem seus próprios requisitos de CPU e memória para realizar o processamento e distribuição de imagens.

3.2.2.4 Camada de gerenciamento

Por fim, a camada de gerenciamento permite um controle um controle fino sobre os recursos e a rede *edge*, permitindo simulações como posicionamento de serviços, migração de serviços, manutenção e agendamento de fluxo de rede.

3.2.3 Mobilidade

O EdgeSimPy suporta a simulação de mobilidade em usuários, nós e servidores. Para isso, o simulador utiliza o posicionamento dos usuários e servidores para determinar a cobertura. O movimento dos usuários é determinado por algoritmos de padrões de mobilidade. Como mencionado previamente, os algoritmos de mobilidade de usuário presentes são:

- Aleatório: Os usuários se movem de forma aleatória pelo mapa da simulação.
- Pathway: Os usuários se movimentam por caminhos determinados pelo mapa da simulação (BAI; HELMY, 2004).

Pela natureza modular do simulador, a implementação de outros algoritmos de mobilidade podem ser implementados, caso surja necessidade. A simulação de movimento de nós e servidores também é suportada, o que é útil para simular redes que tem sua infraestrutura em constante movimento ou alteração, como por exemplo, redes VANET, na qual veículos funcionam como nós ou computadores embarcados com capacidade de processamento, como exemplo podemos pensar na utilização de carros estacionados como pontos de processamento da rede.

3.2.4 Validação da simulação

Um dos aspectos mais valiosos de um estudo de simulação é explicar e entender fenômenos do mundo real que são custosos de serem realizados em laboratório, ou difíceis de coletar em experimentos de campo (XIANG et al., 2005). Como modelar todos os detalhes e comportamentos de um sistema do mundo real é inviável, dada a alta complexidade envolvida, simuladores fazem abstrações de comportamentos dos sistemas (SOUZA; FERRETO; CALHEIROS, 2023). Enquanto estas abstrações auxiliam a reduzir a complexidade das simulações, inevitavelmente também irão introduzir inconsistências ao modelo.

Portanto, uma das tarefas mais importantes em estudos de simulação é verificar se o simulador atinge níveis de precisão aceitáveis dentro das abstrações implementadas. Esta tarefa envolve dois processos: validação e verificação (STEINMAN et al., 1995). O modelo conceitual do simulador é baseado em abstrações conhecidas para a representação de mobilidade, escalonamento e provisionamento de aplicações, os quais já foram formalizados (SOUZA; FERRETO; CALHEIROS, 2023). Enquanto o processo de validação é coberto pelo uso de abstrações reconhecidas, o modelo de verificação utiliza duas técnicas: *animation and tracing* (XIANG et al., 2005), que mostram o comportamento das entidades simuladas conforme o passar do tempo da simulação, permitindo verificar a corretude da logica do simulador.

3.3 TRABALHOS RELACIONADOS

Nesta seção, são descritos alguns dos trabalhos que ofereceram novas políticas, ou refinaram as políticas existentes. Esses trabalhos abordam uma variedade de tópicos, incluindo

segurança, posicionamento de servidores, escalonamento de tarefas, dentre outros. As indicações de referências foram majoritariamente extraídas do *survey* elaborado por (LUO et al., 2021). A Tabela 2 faz um levantamento de diferentes trabalhos já realizados, levantando os métodos utilizados, seus objetivos, e a capacidade de levar em conta diferentes requerimentos, sendo eles a capacidade de computar tarefas, armazenar dados e transferir dados.

Inicialmente, foram revisados trabalhos que realizam o *offloading* de tarefas. O termo *offloading* refere-se ao processo de transferir parte do processamento de um dispositivo ou sistema para outro recurso, para melhorar o desempenho, economizar energia ou otimizar o uso dos recursos. O artigo de (HONG et al., 2019) utilizou uma estratégia de *offloading* e metodologia de teoria de jogos para minimizar o consumo de energia em dispositivos móveis, assim como o tempo de processamento das tarefas, foi testado em um cenário industrial, cujos dispositivos móveis eram artefatos IoT. O modelo usado foi o MCCM. Por sua vez, o trabalho de (GUO; LIU, 2018) também utiliza estratégia de *offloading* e metodologia de teoria de jogos para minimizar o consumo de energia de dispositivos móveis, mantendo o tempo de processamento dentro de um limite determinado. Foi utilizado um sistema de computação colaborativa para construção do modelo.

Em um segundo momento, foram selecionados trabalhos que realizam, principalmente, a alocação de recursos. Em resumo, a alocação de recursos refere-se ao processo de designar recursos específicos, como CPU, memória, largura de banda de rede, armazenamento, entre outros, para tarefas ou processos em um sistema computacional. Esse processo é crucial para garantir o uso eficiente dos recursos disponíveis e otimizar o desempenho do sistema. No artigo de (DINH et al., 2020), algoritmos *online* e *offline*, considerando o custo de processamento e a capacidade local de servidores *edge*, enquanto os comparava com alternativas em *cloud*. (WANG et al., 2019a) desenvolveu o algoritmo HetMEC, para otimização de latência durante o alocação de recursos para redes móveis e heterogêneas com múltiplas camadas em edge.

Em especial, o trabalho de (SOUZA et al., 2023) desenvolveu, testou e comparou contra outros, o algoritmo Thea, para distribuição de tarefas em redes *edge*. Tendo como critérios o consumo de energia dos servidores, latência e violações de confiança, o algoritmo Thea se mostrou competitivo quando comparado a algoritmos mais custosos computacionalmente. A configuração do cenário utilizada na pesquisa de (SOUZA et al., 2023) irá servir como base para o desenvolvimento e simulação das políticas deste trabalho.

É importante ressaltar que os trabalhos elencados são baseados em formulações e implementações complexas, utilizando, por exemplo, teoria de jogos, aprendizado de máquina, entre outros modelos e tecnologias não triviais.

Artigo	Computação	Comunicação	Armazenamento	Algoritmo	Objetivo
(KHALILI; ZARANDI; RASTI, 2019)	x	✓	x	Método MinMax	Consumo de energia
(KUANG et al., 2019)	x	✓	x	algoritmo genético	Consumo de energia
(LI et al., 2018)	x	✓	x	Abordagem baseada em leilão	Consumo de energia
(XU et al., 2019)	✓	x	x	ASGA-III	Tempo
(YU et al., 2018)	✓	x	x	Baseado em jogo	Tempo
(XU; CHEN; ZHOU, 2018)	✓	x	x	Otimização de Lyapunov	Tempo
(GUO et al., 2019)	✓	✓	x	DQN	Performance
(QIAN et al., 2019)	✓	✓	x	algoritmo de pareamento um para muitos	Custo
(WANG et al., 2019c)	✓	✓	x	ADMM	Lucro
(XING et al., 2019)	✓	✓	x	algoritmo heurístico	Latência
(ZHAO et al., 2019)	✓	✓	x	Técnica de decomposição dual com penalidad	Tempo
(WANG et al., 2019b)	✓	✓	✓	DQN	Performance
(LIANG et al., 2017)	✓	✓	✓	ADMM	Consumo de energia
(TAN et al., 2017)	✓	✓	✓	ADMM	Utilidade
(SOUZA et al., 2023)	✓	✓	✓	Thea	Confiança
(ZHOU et al., 2017)	✓	✓	✓	ADMM	Utilidade

Tabela 4 – Trabalhos relacionados.

3.4 CONSIDERAÇÕES PARCIAIS

Neste capítulo foi contextualizado o que é um simulador e conceitos necessários para seu funcionamento, também foi visualizado o funcionamento básico de um simulador. Foi então justificada a escolha do simulador EdgeSimPy para as simulações, assim como seu funcionamento, componentes e arquitetura. O próximo capítulo apresenta cada modelo de provisionamento selecionados implementados, explicações das métricas utilizadas para cada um, e um pseudocódigo de cada um deles.

4 DESENVOLVIMENTO E COMPARAÇÕES

Neste capítulo são exemplificadas e explicadas algumas das diferentes partes da implementação de algoritmos no simulador EdgeSimPy, com o propósito de auxiliar no entendimento dos diferentes componentes e seus atributos dentro do modelo. Após a explicação das partes do código, são definidas as métricas que serão avaliadas após a simulação, assim como a composição do modelo, especificando os modelos de servidores e tarefas, são expostas quais políticas foram implementadas.

4.1 IMPLEMENTAÇÃO DOS ALGORITMOS

O simulador selecionado, por padrão, apresenta duas políticas básicas de escalonamento já implementadas, sendo elas o escalonamento aleatório e a política FIFO. Por sua simplicidade e pequeno número de políticas disponíveis, aquelas elencadas na Seção 2.4 foram inicialmente implementadas no presente trabalho, posteriormente avançando para outras políticas complexas elencadas na literatura especializada. Como elaborado neste capítulo, a implementação de novas políticas é não só possível, como facilitada pela estrutura modular do simulador. A Figura 4.1.1 demonstra um exemplo da implementação dos componentes necessários para a simulação.

```
def escalonador(parameters): #definindo o algoritmo de escalonamento
    for service in Service.all():

        # Não escalonar serviços já escalonados
        if service.server == None and not service.being_provisioned:

            # É iterado sobre a lista para achar um servidor adequado
            #para a tarefa
            for edge_server in EdgeServer.all():

                # verifica se o servidor possui capacidade
                #para processar a tarefa
                if edge_server.has_capacity_to_host(service=service):

                    # inicia-se o provisionamento
                    service.provision(target_server=edge_server)

                    # Depois de terminar este serviço,
                    # é feito o mesmo com o próximo
                    break
```

Listing 4.1.1 – Exemplo de definição do algoritmo de escalonamento

No código ilustrativo da Figura 4.1.1 é realizada a definição de um simples algoritmo de escalonamento, neste caso está sendo implementado um escalonador de *First-Fit* (FF). Para toda tarefa na lista, é iterado sobre a lista de servidores, analisando um por um, para verificar se

algum servidor tem recursos suficientes para realizar a tarefa. Quando identificado, a tarefa é alocada para este servidor.

```
def stopping_criterion(model: object):
    # Define a variavel para manter contagem de serviços escalonados
    provisioned_services = 0

    # Itera sobre os serviços para verificar quantos foram escalonados
    #dentro da infraestrutura
    for service in Service.all():

        # Inicialmente uma tarefa não escalonada
        #tem seu atributo de servidor == None
        # Quando esse valor muda, entende se que
        #está escalonada para algum servidor
        if service.server != None:
            provisioned_services += 1

    return provisioned_services == Service.count()
```

Listing 4.1.2 – Exemplo de definição para o critério de parada

Após a definição da política de escalonamento, é necessária uma condição de parada. No código exemplo da Figura 4.1.2, o critério é o escalonamento de todas as tarefas, ou seja, é verificado a cada passo se o número de tarefas escalonadas é igual ao número total de tarefas submetidas. Quando o número for de fato igual, a simulação finaliza.

```
# Cria um objeto de simulador
simulator = Simulator(
    dump_interval=5,
    tick_duration=1,
    tick_unit="seconds",
    stopping_criterion=stopping_criterion,
    resource_management_algorithm=my_algorithm,
)

# Carrega o JSON
simulator.initialize(input_file="dataset.json")

# Executa a simulação
simulator.run_model()
```

Listing 4.1.3 – Exemplo da configuração do simulador e realização da simulação

Por fim, o código ilustrativo da Figura 4.1.3 representa a criação do objeto simulador, definindo os parâmetros explicados no Capítulo 3, sendo eles, em ordem: intervalo de gravação

da simulação, o GVT, definido aqui como um segundo, e definindo o critério de parada e o algoritmo de escalonamento. Após isso é carregado o arquivo JSON e iniciada a simulação.

4.2 ALGORITMOS IMPLEMENTADOS E MÉTRICAS ANALISADAS

Os modelos testados neste trabalho, como apresentados na Tabela 5 foram Thea (SOUZA et al., 2023), Argos (SOUZA et al., 2022), Faticanti (FATICANTI et al., 2020), DCF, o qual foi desenvolvido durante o trabalho, e SSP, também desenvolvido durante o trabalho. As métricas alvo são:

Para realização de testes e simulações, foram utilizados as métricas definidas por (SOUZA et al., 2023):

- Ocupação global dos servidores: a porcentagem de ocupação dos serviços de todos os servidores.
- Consumo de energia geral: dado pelo consumo de energia de todos os servidores. O consumo de energia de um servidor começa com o valor de consumo mínimo, como demonstrado na tabela 6, aumentando linearmente com a ocupação do servidor, até chegar no valor de consumo máximo, quando tiver a ocupação de 100%
- Violação de *Service Level Agreement* (SLA) de atraso: quantas violações de SLA de atraso ocorreram. Contado apenas uma vez por usuário.
- Violação de SLA de confiança: quantas violações de SLA de confiança ocorreram entre todos os servidores.
- Custo: calculado pela média geométrica entre o consumo e energia e ambas violações de SLA.

Adicionalmente este trabalho incluiu também as métricas de:

- Quantidade de serviços provisionados: A quantidade de serviços que um modelo conseguiu provisionar na simulação.
- Função de distribuição acumulada dos caminhos na topologia: Quantas vezes um caminho é utilizado para provisionamento de todos os serviços.

Cada um dos algoritmos implementados apresenta diferentes métricas que visa otimizar, para isso, cada um utiliza diferentes abordagens e dados. Esta seção entrará mais afundo em cada um dos algoritmos e como provisionam os serviços.

Modelo	Métricas alvo				
	Latência	Privacidade	Consumo de energia	Distribuição de serviços	Otimização de recursos
Argos (SOUZA et al., 2022)	✓	✓	x	x	x
Thea (SOUZA et al., 2023)	✓	✓	✓	x	x
Faticanti (FATICANTI et al., 2020)	✓	✓	x	x	✓
DCF	x	✓	x	✓	✓
SPP	✓	x	x	x	x

Tabela 5 – Algoritmos implementados no trabalho e suas métricas alvo.

4.2.1 Faticanti

Os autores realizaram uma comparação entre o custo e a viabilidade entre uma abordagem de busca em largura em comparação a uma busca de profundidade para o provisionamento dos serviços (FATICANTI et al., 2020). O algoritmo, demonstrado no pseudocódigo 1, para cada serviço na lista, ordena todos os servidores na rede, primeiramente pelo nível de confiança do usuário, uma variável que cada usuário tem para cada servidor da rede, demonstrada no pseudocódigo como $usuario.confianca(servidor)$, após isso o *delay* entre um usuário e o servidor, demonstrada no pseudocódigo 1 como $distancia(usurio, servidor)$, e por ultimo a quantidade de CPU disponível no servidor, obtido subtraindo quantos cores estão sendo utilizados no servidor do total de cores do servidor. Com a lista de servidores ordenada, o modelo então itera sobre a lista, verificando se o serviço está sem provisionamento e se o servidor possui recursos disponíveis para provisionar o serviço, utilizando a função $servidor\_tem\_capacidade(servico)$ no pseudocódigo 1, caso ambos sejam verdade, o serviço é provisionado.

Nos passos seguintes, primeiramente o modelo calcula o tamanho do caminho entre o usuário e o servidor que provisiona seu serviço, criando uma lista dos 10 usuários mais distantes, salvando o serviço e a distancia entre ele e o servidor no dicionário $top\_10\_servicos[]$. O modelo então continua da mesma forma do primeiro passo até o ponto de provisionar o serviço, que caso o modelo esteja tentando provisionar esteja no dicionário, e encontrar um servidor com recursos disponíveis, o serviço é migrado para o novo servidor. Esta lógica de migração após o primeiro passo se repete para os próximos modelos.

4.2.2 Argos

O modelo Argos (SOUZA et al., 2022) é voltado primeiramente para migrações, em seu estudo original, foi avaliado com base em ambas as violações de SLA, assim como o número de migrações realizadas. O intuito principal das migrações é reduzir a latência entre serviço e servidor. Primeiramente este modelo ordena a lista de serviços pelo SLA de atraso de cada serviço. Após ordenar os usuários, são ordenados os servidores, desta vez por por SLA de privacidade, seguido pelo SLA de atraso. É então iterado sobre a lista de servidores verificando se possuem recursos o suficiente para provisionar o serviço. Este modelo foi modificado para se adequar a migração.

A lógica para os próximos passos de simulação onde ocorre a migração é igual a explicada

Algoritmo 1: Faticanti

```
servicos ← lista_de_servicos();
foreach servico ∈ servicos do
    if servico.servidor() is not NULL then
        calcular_tamanho_path(servico.servidor())
    end
end
top_10_path ← obter_10_maiores;
foreach servico ∈ top_10_path() do
    top_10_servicos[] ← servico;
end
foreach servico ∈ servicos do
    usuario ← servico.usuario;
    servidores ←
     ordenar_servidores(−usuario.confianca(servidor), distancia(usurio, servidor), servidor.cpu −
     servidor.cpu_usada);
    foreach servidor ∈ servidores do
        if servidor_tem_capacidade(servico) and servico.servidor is NULL then
            provisionar_servico(servico, servidor);
            Parar;
        else
            if servidor_tem_capacidade(servico) and servico ∈ top_10_servicos[] then
                provisionar_servico(servico, servidor);
                Parar;
            end
        end
    end
end
```

previamente no algoritmo 1.

4.2.3 Thea

Thea foi testado originalmente para as métricas de ambas as violações de SLA, tamanho do caminho de comunicação das aplicações, ocupação dos servidores e consumo de energia (SOUZA et al., 2023). Originalmente não foi testado em uma simulação com mais de um passo, e por consequência, não foi testado em um cenário onde exista mobilidade de usuários. O modelo Thea foi modificado para se adequar ao cenário 2, por originalmente não ser compatível com um ambiente que não consiga provisionar todos os serviços. Outra modificação realizada no algoritmo original foi a reintrodução de serviços já provisionados, visto que originalmente, se todos os serviços de uma aplicação já constassem como provisionados, toda a aplicação era marcada como provisionada e não era verificada novamente em passos seguintes, o que impedia a migração de serviços entre servidores. O modelo Thea, representado no pseudocódigo 3 difere dos outros, primeiramente ordenando as aplicações, utilizando como chave a normalização do SLA de atraso dos serviços da aplicação, assim como a normalização do dos SLA de privacidade

Algoritmo 2: Argos

```
servicos ← lista_de_servicos();
foreach servico ∈ servicos do
    if servico.servidor() is not NULL then
        calcular_tamanho_path(servico.servidor());
    end
end
top_10_path ← obter_10_maiores;
foreach servico ∈ top_10_path do
    top_10_servicos[] ← servico;
end
aplicacao ← ordenar_aplicacao(user.delay_sla);
foreach app ∈ aplicacao do
    usuario ← app.usuario;
    servicos ←
     ordenar_servicos(−servico.requerimento_privacidade, −servico.demanda_cpu)
     foreach servico ∈ servicos do
        servidores ←
         ordenar_servidores(−usuario.confianca(servidor), distancia(usuario));
        foreach servidor ∈ servidores do
            if servidor_tem_capacidade(servico) and servico.servidor is NULL then
                provisionar_servico(servico, servidor);
                Parar;
            else
                if servidor_tem_capacidade(servico) and servico ∈ top_10_servicos[]
                 then
                    provisionar_servico(servico, servidor);
                    Parar;
                end
            end
        end
    end
end
```

dos serviços da aplicação. Desta forma, o modelo dá prioridade para aplicações com maiores requerimentos para ambos SLAs, provisionando seus serviços primeiro. Após isso, o modelo itera pela lista serviços de cada aplicação, ordenando a lista de servidores, para cada aplicação, utilizando como critério o custo de alocação do serviço, o consumo de energia resultante ao provisionar o serviço no servidor e o atraso entre o usuário e o servidor. Após ordenar os servidores o modelo então itera sobre a lista, provisionando o serviço no primeiro servidor que tenha recursos suficientes para provisionar o serviço.

A lógica para os próximos passos de simulação onde ocorre a migração é igual a explicada previamente no algoritmo 1.

Algoritmo 3: Thea

```
servicos ← lista_de_servicos();
foreach servico ∈ servicos do
    if servico.servidor() is not NULL then
        calcular_tamanho_path(servico.servidor());
    end
end
top_10_path ← obter_10_maiores;
foreach servico ∈ top_10_path do
    top_10_servicos[] ← servico;
end
metadata_aplicacao ← ordenar_aplicacao(normalizacao(delay,
delay_max, delay_min), normalizacao(privacidade,
 privacidade_max, privacidade_min))
foreach aplicacao ∈ metadata_aplicacao do
    foreach servico ∈ aplicacao.servico do
        servidores ←
         ordenar_servidores(normalizacao(custo, custo_max, custo_min),
        normalizacao(consumo_energia, consumo_max, consumo_min), SLA_atraso)
        if servidor_tem_capacidade(servico) and servico.servidor is NULL then
            provisionar_servico(servico, servidor)
            Parar;
        else
            if servidor_tem_capacidade(servico) and servico ∈ top_10_servicos[]
             then
                provisionar_servico(servico, servidor)
                Parar;
            end
        end
    end
end
```

4.2.4 DCF

O modelo DCF (Distribuição para Caso de Falha) foi criado durante o desenvolvimento deste trabalho, tendo como principal propósito a distribuição de serviços de uma mesma aplicação para diversos servidores, de modo a minimizar a perda de serviços de uma mesma aplicação em uma instancia de falha de servidor. Como demonstrado no pseudocódigo da Figura 4, o modelo, para cada serviço, contabiliza quantos serviços da mesma aplicação ao serviço que está sendo provisionado atualmente já estão nos servidores, essa informação então é utilizada para ordenar os servidores, usando primeiramente o número de serviços da mesma aplicação no servidor, seguido do *delay* entre o usuário e o servidor, e por ultimo a quantidade de CPU disponível.

A lógica para os próximos passos de simulação onde ocorre a migração é igual a explicada previamente no algoritmo 1.

Algoritmo 4: DCF

```
servicos ← lista_de_servicos();
foreach servico ∈ servicos do
    app ← servico.aplicacao
     foreach servidor ∈ servidor.all() do
        servicos_por_servidor{servidor} ← 0
    end
    foreach servidor ∈ servico_por_servidor do
        foreach servico_aux ∈ servidor.servico do
            if servico_aux.aplicacao == app then
                servicos_por_servidor{servidor} += 1
            end
        end
    end
    if servico.servidor() is not NULL then
        calcular_tamanho_path(servidor)
    end
    if servico then
    end
end
top_10_path ← obter_10_maiores;
foreach servico ∈ top_10_path do
    top_10_servicos[] ← servico;
end
foreach servico ∈ servicos do
    usuario ← servico.usuario;
    servidores ←
     ordenar_servidores(servicos_por_servidorservidor, path_delay(usuario, servidor), server.cpu −
     server.cpu_demand;
    foreach servidor ∈ servidores do
        if servidor_tem_capacidade(servico) and servico.servidor is NULL then
            provisionar_servico(servico, servidor) Parar;
        else
            if servidor_tem_capacidade(servico) and servico ∈ top_10_servicos[] then
                provisionar_servico(servico, servidor) Parar;
            end
        end
    end
end
```

4.2.5 SPP

O modelo SPP (Shortest Path Picker) foi criado durante o desenvolvimento deste trabalho, tendo uma lógica de execução simples, demonstrada no pseudocódigo da Figura 5, onde apenas calcula a distancia entre o usuário e os servidores, utilizando a função $path\_delay(usuario, servidor)$ no pseudocódigo 5, ordenando os servidores do mais próximo ao mais distante, e iterando sobre

esta lista verificando se tem condição de provisionar o serviço.

A lógica para os próximos passos de simulação onde ocorre a migração é igual a explicada previamente no algoritmo 1.

Algoritmo 5: SPP

```
servicos ← lista_de_servicos();
foreach servico ∈ servicos do
    if servico.servidor() is not NULL then
        calcular_tamanho_path(servidor)
    end
end
top_10_path ← obter_10_maiores;
foreach servico ∈ top_10_path do
    top_10_servicos[] ← servico;
end
foreach servico ∈ servicos do
    usuario ← servico.usuario;
    servidores ← ordenar_servidores(path_delay(usuario, servidor);
    foreach servidor ∈ servidores do
        if servidor_tem_capacidade(servico) and servico.servidor is NULL then
            provisionar_servico(servico, servidor) Parar;
        else
            if servidor_tem_capacidade(servico) and servico ∈ top_10_servicos[] then
                provisionar_servico(servico, servidor) Parar;
            end
        end
    end
end
```

Uma das mudanças feitas neste trabalho foi a adequação dos algoritmos para uso em cenários de migração e em infraestruturas sem recursos suficientes para o provisionamento de todos os serviços. O método para a migração utilizado neste trabalho é selecionar os cinco serviços mais distantes de seus servidores, e realizar o *loop* do modelo novamente, deste modo tentando provisionar o serviço em um servidor mais adequado, dada a movimentação dos usuários

4.3 CONSIDERAÇÕES PARCIAIS

Neste capítulo foram exemplificados e elaborados diferentes trechos de código necessários para a utilização do simulador *EdgeSimPy*. Também foram dadas as especificações do funcionamento do cenário e seus componentes, como servidores e tarefas. Após isso, foram demonstrados e explicados os modelos que foram simuladas no decorrer deste trabalho.

O próximo capítulo apresenta os cenários utilizados, os resultados obtidos e uma discussão estes resultados.

5 SIMULAÇÕES E RESULTADOS

Neste capítulo são apresentados os cenários utilizados para as simulações, descrevendo sua topologia e configuração, assim como os resultados obtidos por cada algoritmo nestes cenários. Após isso é feita uma análise e discussão do desempenho de cada modelo. Por fim são feitos comentários sobre a utilização do simulador EdgeSimPy.

5.1 COMPOSIÇÃO DO CLOUD-EDGE CONTINUUM

A infraestrutura *edge* utiliza uma rede celular, na qual um conjunto ε de servidores *edge* são posicionados próximos de estações base β, que são conectados por um conjunto de *links* de rede ι. O mapa é composto de um grupo de células hexagonais, como na pesquisa de (ARAL; MAIO; BRANDIC, 2021), na qual cada célula representa a área de cobertura de uma estação base. Enquanto as estações base permitem conexão sem fio para usuários, os *links* de rede permitem conexão entre servidores e diferentes estações base. Um *link* é modelado como $L_f = g_f$, no qual g_f representa a latência de L_f.

Serviços são hospedados em um conjunto de servidores S. Cada servidor é modelado como $\varepsilon_i = C_i, R_i, Q_i, M_i$, sendo que C_i e R_i representam respectivamente a capacidade de CPU e a capacidade de memória RAM do servidor ε_i, e Q_i, M_i representam o consumo estático (ou consumo quando não utilizado) e máximo (quando completamente utilizado). Esta estrutura de consumo de energia pode ser integrada em diferentes modelos, como especificado em (BELOGLAZOV; ABAWAJY; BUYYA, 2012).

Considera-se um conjunto de usuários μ posicionados em pontos pré-definidos no mapa, requisitando recursos e tarefas. Uma tarefa é representada por $T_j = u_j, b_j, \lambda_j$. O atributo u_j se refere ao usuário requisitante. b_j representa o caminho de comunicação entre usuário-servidor, e λ_j representa o limite de tempo de resposta aceitável pela tarefa.

O tempo de latência entre um par de elementos na infraestrutura, representados como ω_1, ω_2, pode indicar qualquer um dos pontos da rede, sendo obtido pela função $\psi(B_1, B_2)$, que encapsula o algoritmo de (DIJKSTRA, 1959), o *Shortest Path Algorithm* (SPA). Neste cenário, a latência é calculada como a soma da latência de todos os pontos entre $B_1 e B_2$.

O modelo foi projetado com três modelos de servidores diferentes, como demonstrado na Tabela 4, cada um com diferentes especificações de recursos e uso de energia. Estas especificações foram extraídas de servidores reais, como descritos em (ISMAIL; MATERWALA, 2021). O número de cada tipo de servidor varia em diferentes simulações, e estes serão agrupados pelo algoritmo *K-Means*, por (MACQUEEN et al., 1967). A topologia e as definições de latência também variarão em diferentes simulações. As tarefas por sua vez serão divididas em quatro tamanhos, como demonstrado na Tabela 5. A quantidade de cada tarefa irá variar em diferentes simulações.

Modelo	CPU	RAM	Consumo inativo	Consumo máximo
Modelo 1	32 núcleos	32GB	265W	1387W
Modelo 2	48 núcleos	64GB	127W	559W
Modelo 3	36 núcleos	64GB	45W	276W

Tabela 6 – Especificações dos servidores *Edge*

Tamanho	Demanda de CPU	Demanda de RAM
Menor	1 núcleo	1GB
Pequena	2 núcleos	2GB
Média	4 núcleos	4GB
Grande	8 núcleos	8GB
Enorme	16 núcleos	16GB

Tabela 7 – Especificações das tarefas

5.2 CENÁRIOS

Para as simulações foram utilizados dois cenários, ambos contando com 100 passos de simulação, sendo que cada passo foi definido como um minuto na simulação. O primeiro cenário, representado na Figura 5, conta com 18 servidores, representados pelos nodos de cor vermelha. Cada servidor pode apresentar uma das três configurações disponíveis na Tabela 6. O cenário conta com 16 usuários, representados como os nodos de tamanho maior. Os usuários também possuem um requerimento de tempo e privacidade, chamados respectivamente de *Delay SLA* e *Privacy SLA*, assim como o grau de confiança que cada usuário tem em cada uma das configurações de servidores, portanto, se um serviço do usuário for escalonado para um servidor muito distante, ou no qual o usuário tem pouca confiança, ocorre uma violação de SLA. Cada usuário faz uso de uma aplicação, e cada aplicação tem um número *n* de serviços, com um total de 60 serviços e 16 aplicações. Cada serviço pode apresentar uma das configurações disponíveis na Tabela 7, com exceção das tarefas de tamanho "Menor", que são utilizadas somente no cenário 2. Adicionalmente, o espaço em disco é composto pelas imagens e registros de contêiner de cada serviço, o que é calculado automaticamente pelo simulador. Este primeiro cenário contem poucos serviços, existindo recursos suficientes entre os servidores para o provisionamento de todos os serviços. O intuito deste cenário é comparar o desempenho dos modelos em um ambiente ideal.

O segundo cenário, representado na figura 6 também conta com 18 servidores, com as mesmas configurações disponíveis na tabela 6, contudo, apresenta 24 usuários, 24 aplicações e 136 serviços. Os serviços apresentam uma das configurações disponíveis na tabela 7. Este segundo cenário apresenta uma quantidade maior de serviços que o primeiro, desta forma não existem recursos suficientes para o provisionamento de todos os serviços. o intuito deste cenário é comparar o desempenho dos modelos em um cenário mais complexo e ocupado.

Em ambos os cenários, os usuários são móveis, utilizando o modelo de mobilidade

random. Por consequência as aplicações e serviços se movem juntamente com seus usuários, desta forma surge a necessidade de migrar serviços entre servidores, pois o cenário está em constante mudança.

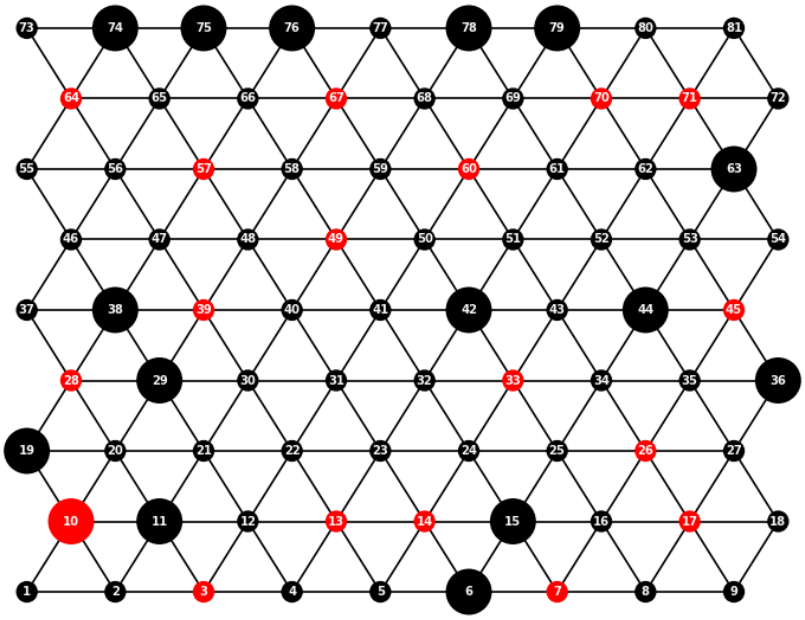


Figura 5 – Topologia do cenário 1.

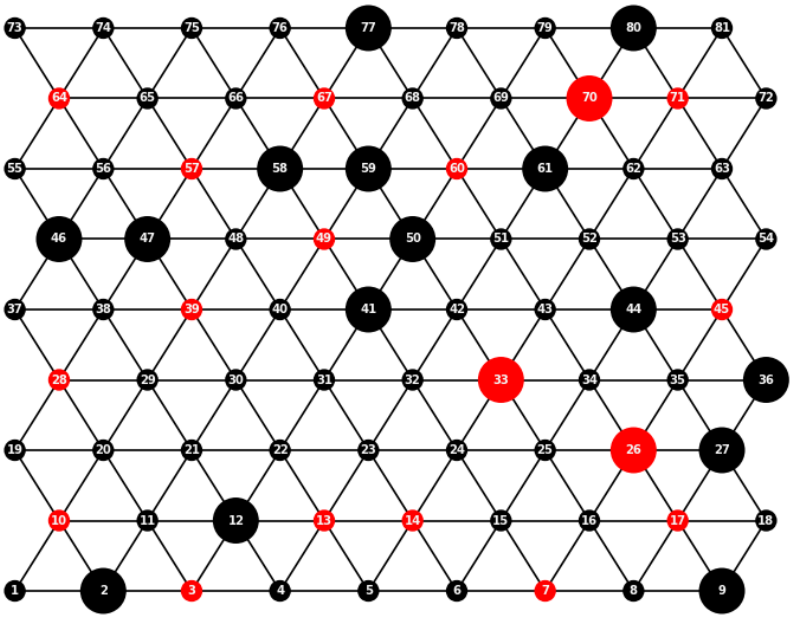


Figura 6 – Topologia do cenário 2.

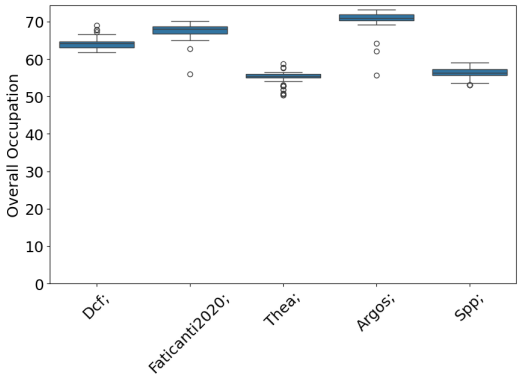
5.3 ANÁLISE DE RESULTADOS

A seguir serão apresentados os resultados das simulações, primeiramente com os dados do primeiro cenário, e então do segundo. As figuras 7a, 7b, 7c, 7d, 7e, 7f são interpretações gráficas da tabela 8 em representação por *boxplot*. As figuras exibem a mediana, os quartis e os valores extremos. A caixa central representa 50% dos dados, com a linha interna indicando a mediana. As "extensões"(ou bigodes) mostram a variação dos dados fora da caixa, enquanto os pontos isolados são valores atípicos. A tabela, de forma semelhante, apresenta os valores para média (μ), desvio padrão (σ), mediana (M_d) e os 3 quartis (Q1, Q2, Q3) para cada métrica de cada modelo. As figuras 8a e 9b representam a quantidade de serviços total de cada cenário, quantos serviços foram provisionados e quantos ficaram sem provisionamento. As figuras 7f e 9a representam a Função de Distribuição Acumulada (CDF) da quantidade de serviços que tomam um caminho específico entre dois nós na rede, calculada com todos os passos da simulação. No eixo horizontal (x), temos a quantidade de vezes que um caminho entre dois nós é tomado em um determinado passo da simulação, e no eixo vertical (y), a probabilidade acumulada, que varia de 0 a 1.

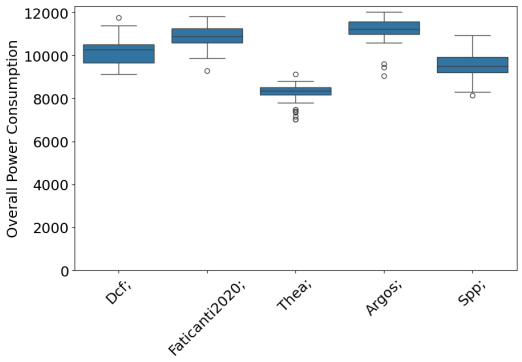
Métrica		Argos	Thea	Faticanti	DCF	SPP
Ocupação	$\mu \pm \sigma$	70,75 ± 2,12	55,11 ± 1,40	67,62 ± 1,75	64,05 ± 1,33	56,32 ± 1,35
	M_d	70,90	55,40	68,03	64,17	56,27
	Q1	70,28	54,94	66,82	63,12	55,61
	Q2	70,90	55,40	68,03	64,17	56,27
	Q3	71,93	55,91	68,62	64,67	57,24
Energia	$\mu \pm \sigma$	11234,38 ± 499,21	8300,41 ± 374,61	10913,12 ± 475,12	10170,25 ± 570,38	9542,62 ± 552,41
	M_d	11243,64	8344,89	10874,79	10267,87	9508,04
	Q1	10991,00	8181,00	10582,36	9670,64	9210,18
	Q2	11243,64	8344,89	10874,79	10267,87	9508,04
	Q3	11589,86	8525,20	11266,79	10507,48	9917,29
Violação de atraso	$\mu \pm \sigma$	15,46 ± 0,74	14,06 ± 0,86	14,04 ± 1,04	14,04 ± 2,05	12,74 ± 1,73
	M_d	16,00	15,00	16,00	14,00	13,00
	Q1	15,00	14,00	12,00	13,00	12,00
	Q2	16,00	15,00	15,00	14,00	13,00
	Q3	16,00	15,00	16,00	15,00	14,00
Violação de privacidade	$\mu \pm \sigma$	19,39 ± 1,39	13,94 ± 2,23	20,55 ± 0,80	17,80 ± 2,20	28,63 ± 2,85
	M_d	19,00	13,00	21,00	18,00	28,00
	Q1	19,00	13,00	20,00	17,00	27,00
	Q2	19,00	13,00	21,00	18,00	28,00
	Q3	20,00	15,00	21,00	30,00	38,00
Custo	$\mu \pm \sigma$	149,73 ± 5,13	118,83 ± 5,82	147,16 ± 8,02	136,05 ± 7,31	150,26 ± 7,25
	M_d	150,16	117,53	149,51	135,72	150,26
	Q1	147,64	115,08	140,22	132,00	145,87
	Q2	150,16	117,53	149,51	135,72	150,26
	Q3	153,17	121,21	153,90	140,87	154,81

Tabela 8 – Métricas da simulação do Cenário 1.

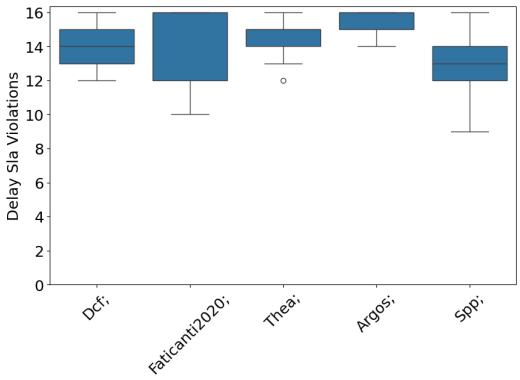
A Figura 7b representa o consumo de energia no cenário 1. Percebe-se que o modelo Thea apresentou os menores valores de consumo de energia, tendo consumo médio de 8300,41, apresentou também uma quantidade alta de *outliers*, mostrando-se potencialmente instável nesta métrica. O modelo argos, além de ter as médias mais altas, com consumo médio de 11234,38, apresentou também os *outliers* mais distantes da média, portanto além de ter o pior desempenho médio, é também o mais instável entre os modelos testados. O modelo SPP teve também um



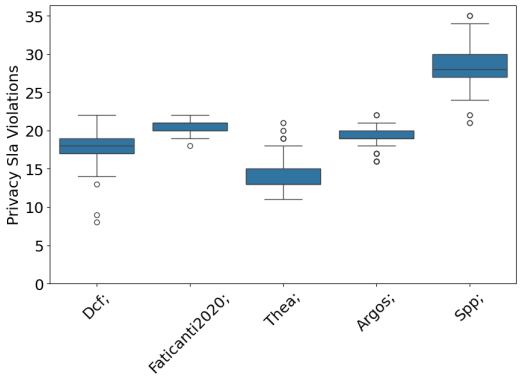
(a) Ocupação geral dos servidores no cenário 1.



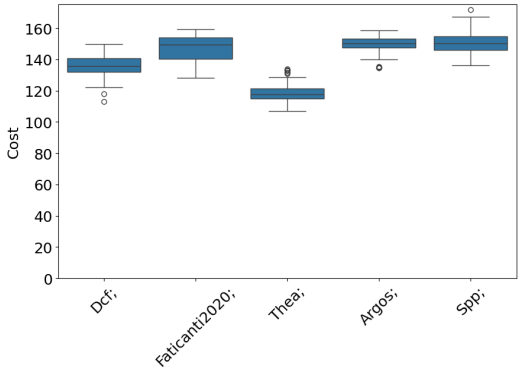
(b) Consumo de energia total no cenário 1.



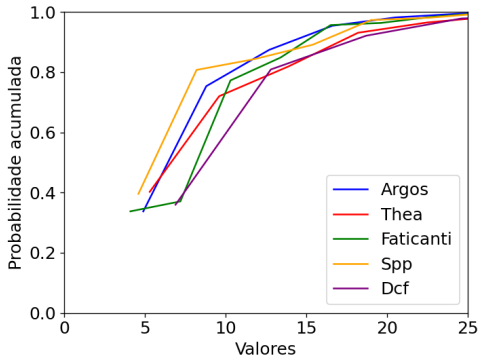
(c) Violações de SLA de atraso no cenário 1.



(d) Violações de SLA de privacidade no cenário 1.

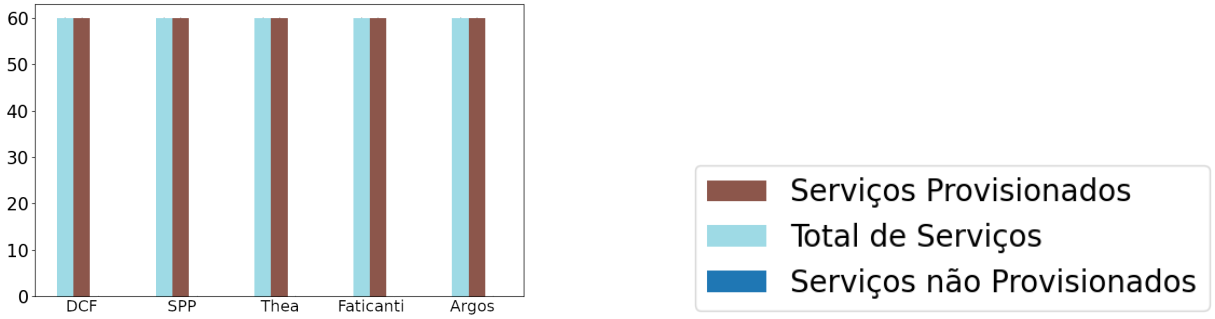


(e) Custo total no cenário 1.



(f) Distribuição dos caminhos no cenário 1.

Figura 7 – Resultados da simulação no cenário 1.



(a) Provisionamento de serviços no cenário 1. (b) Legenda para os gráficos 8a e 9b.

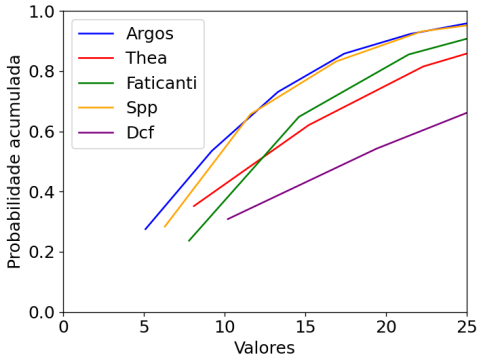
Figura 8 – Continuação dos resultados da simulação no cenário 1.

bom desempenho, mostrando um consumo médio de 9542,62, e apesar de ter um desvio padrão maior que a maioria dos outros modelos, com 552,41 de desvio, tem apenas um *outlier*.

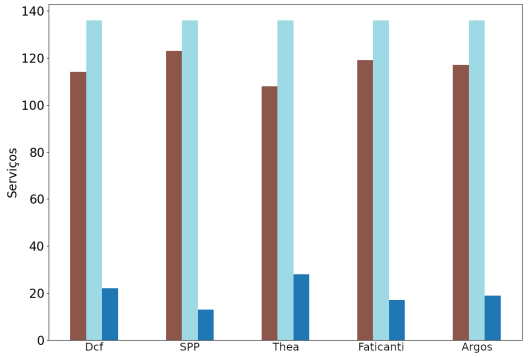
A Figura 7c representa a quantidade de violações de SLA para atraso. Os melhores resultados foram obtidos pelo modelo SPP, tendo a menor mediana de 13, contudo, também foi o modelo mais instável dos cinco, tendo uma variação entre 9 e 16 violações e um desvio de 1,73, sendo o segundo maior desvio, ainda assim apresentou em media a menor quantidade de violações com 12,74. Faticanti também apresentou uma grande instabilidade em violações de atraso, com desvio de 1,04, mas nenhum outlier. O modelo mais estável foi o Argos, porém também foi o modelo com pior desempenho, com média de 15,46.

A Figura 7d representa os dados de violações de SLA para privacidade. O modelo DCF apresentou os melhores resultados em passos individuais, tendo instancias com 7 e 9 violações, contudo, estes resultados foram *outliers* e não representam o funcionamento médio do modelo, que foi de 17,80. O modelo Thea apresentou em media os melhores resultados nesta métrica, com média de 13,94, porém apresenta alguns *outliers* elevados, representando picos de violação durante sua execução, de modo que em certos passos foi pior que outros modelos. Os modelos Argos e Faticanti foram os mais estáveis, apresentando poucos *outliers* e uma distribuição compacta, com desvios de 1,39 e 0,80 respectivamente, indicando que seu comportamento é previsível durante seu funcionamento. Por fim o modelo SSP apresentou os piores resultados nesta métrica, com média de 28,63, assim como grande variação nos valores, com desvio de 2,85, o maior entre os modelos.

A Figura 7e representa o custo total de execução dos modelos, custo que é determinado com a média geométrica entre o consumo de energia e ambas as violações de SLA. O modelo Thea teve o menor custo médio, com 118,83, contudo novamente apresentou vários *outliers* com valores similares aos médios do DCF. Apesar dos *outliers*, o modelo Thea ainda teve os melhores resultados entre os modelos testados tendo um desvio padrão de 5,82, sendo o segundo menor. O modelo Argos apresentou uma mediana elevada de 149,73, mas foi o mais estável, com desvio de 5,13. Faticanti foi o menos estável dos analisados, que apesar de não ter outliers, teve desvio de 8,02. SPP apresentou em média o pior desempenho, com 150,26, e uma mediana semelhante ao Argos e Faticanti de também 150,26.



(a) Distribuição dos caminhos no cenário 2.



(b) Provisionamento de serviços no cenário 2.

Figura 9 – Resultados da simulação no cenário 2.

Métrica		Argos	Thea	Faticanti	DCF	SPP
Ocupação	$\mu \pm \sigma$	87,20 ± 0,00	87,20 ± 0,00	87,20 ±0,00	87,20 ±0,00	86,83 ± 0,05
	M_d	87,20	87,20	87,20	87,20	86,82
	Q1	87,20	87,20	87,20	87,20	86,81
	Q2	87,20	87,20	87,20	87,20	86,82
	Q3	87,20	87,20	87,20	87,20	86,88
Energia	$\mu \pm \sigma$	13332,00 ± 0,00	13332,00 ± 0,00	13332,00 ± 0,00	13332,00 ± 0,00	13286,79 ± 18,29
	M_d	13332,00	13332,00	13332,00	13332,00	13281,52
	Q1	13332,00	13332,00	13332,00	13332,00	13278,93
	Q2	13332,00	13332,00	13332,00	13332,00	13281,52
	Q3	13332,00	13332,00	13332,00	13332,00	13305,00
Violação de atraso	$\mu \pm \sigma$	24 ± 0,00	24 ± 0,00	24 ± 0,00	24 ± 0,00	24 ± 0,00
	M_d	24,00	24,00	24,00	24,00	24,00
	Q1	24,00	24,00	24,00	24,00	24,00
	Q2	24,00	24,00	24,00	24,00	24,00
	Q3	24,00	24,00	24,00	24,00	24,00
Violação de privacidade	$\mu \pm \sigma$	39,00 ± 0,00	20,00 ± 0,00	47,00 ± 0,00	34,00 ± 0,00	56,00 ± 0,81
	M_d	39,00	20,00	47,00	34,00	56,00
	Q1	39,00	20,00	47,00	34,00	55,00
	Q2	39,00	20,00	47,00	34,00	56,00
	Q3	39,00	20,00	47,00	34,00	57,00
Custo	$\mu \pm \sigma$	231,94 ± 0,00	185,65 ± 0,00	246,83 ± 0,00	221,57 ± 0,00	261,48 ± 1,29
	M_d	231,94	185,65	246,83	221,57	261,49
	Q1	231,94	185,65	246,83	221,57	259,93
	Q2	231,94	185,65	246,83	221,57	261,49
	Q3	231,94	185,65	246,83	221,57	262,87

Tabela 9 – Métricas da simulação do cenário 2

As Figuras 9b e 9a, assim como a Tabela 9 representam os resultados da simulação no segundo cenário, percebe se que como o esperado, nenhum dos algoritmos conseguiu provisionar todos os serviços, em destaque, DCF e Thea foram os que menos provisionaram entre os modelos testados, sendo o modelo SPP o que conseguiu provisionar o maior número de serviços. Analisando a tabela 9, percebemos que existe pouca ou nenhuma variação entre passos na simulação. Este resultado se deu ao fato de que no primeiro passo da simulação, os modelos lotaram todos os servidores de modo que uma migração era impossível, visto que um serviço não tinha para onde migrar. O modelo SPP conseguiu realizar algumas migrações, e deste modo foi o único que teve qualquer alteração entre passos da simulação. Devido a isto, a análise é mais

limitada quando comparada a que foi feita no cenário 1.

Na métrica de ocupação, todos os modelos tiveram 87,20% de ocupação, com exceção do modelo SPP, que teve em média 86,83%. De forma semelhante, visto que o consumo de energia está diretamente atrelado a ocupação dos servidores, todos os modelos apresentaram consumo de 13332,00 com exceção do SPP, que teve consumo médio de 3286,79. Para a métrica de violação de SLA de atraso, todos os modelos violaram pelo menos um SLA de cada usuário em todos os passos, portanto todos apresentaram 24 violações.

Para a métrica de violações de SLA de privacidade, o comportamento apresentado no cenário 1 novamente é exibido no cenário 2, com Thea tendo o menor número de violações, com 20, seguido de DCF com 34, Argos com 39, Faticanti com 47 e por ultimo SSP com 56. Para o custo, como foi determinado por uma média geométrica entre consumo de energia e ambas as violações, a única diferença entre os modelos ocorre na violação de privacidade, portanto o custo segue a mesma ordem das violações, tendo com o menor custo Thea, seguido de DCF, Argos, Faticanti e por ultimo SSP.

A figura 9b demonstra o total de serviços provisionados e os não provisionados, o modelo que mais conseguiu provisionar serviços foi o SPP, com 123 serviços, seguido de Faticanti com 119, Argos com 117, DCF com 114, e por último Thea com 108. Nota-se que o simulador não leva em os serviços não provisionados para o calculo das métricas avaliadas, ou seja, serviços não provisionados não estão violando SLA de privacidade, e consequentemente de preço.

Analisando a figura 9a, percebemos que o modelo DCF tem uma distribuição de caminhos mais diversa, tendo menos caminhos com altos níveis de trafego, Thea e Faticanti tiveram um desempenho similar nesta métrica e tiveram uma distribuição com caminhos mais trafegados que o modelo DCF, mas menos que SPP e Argos.

5.3.1 Discussão

Avaliando os dados das simulações de ambos os cenários, o modelo Thea se apresenta preferível em cenários com recursos suficientes para o provisionamento de todos os serviços, apresentando o menor custo, consumo de energia e menos violações de privacidade, assim como resultados geralmente estáveis na maioria das métricas. Contudo, aparenta não fazer um uso otimizado dos recursos do cenário, o que resulta em um menor número de provisionamentos em cenários sem recursos suficientes. Além disto, o modelo em sua forma original não verifica serviços previamente provisionados, potencialmente perdendo desempenho em cenários que apresentam mobilidade de serviços, servidores ou usuários.

O modelo DCF se mostrou competitivo com os outros modelos, contudo se mostra instável em algumas métricas. O modelo também apresenta o benefício de distribuir os serviços entre servidores, para evitar a perda total de uma aplicação em caso de mal funcionamento, o que pode se provar um ponto interessante para cenários em *edge* que façam uso de servidores móveis, instáveis ou que não tenham uma operação continua.

5.4 OBSERVAÇÕES SOBRE O SIMULADOR EDGESIMPY

O simulador EdgeSimPy se provou simples de utilizar, com uma boa documentação, exemplos e tutoriais, possuindo a maioria das ferramentas necessárias para simular e coletar os dados desejados, sua estrutura modular permitiu a integração de novos modelos de forma rápida e simples. O ajuste de parâmetros na simulação e no cenário também não representaram um obstáculo, mesmo sem experiencia prévia com o simulador.

Uma limitação encontrada no decorrer dos experimentos foi no modo em que o simulador realiza a detecção de violações de SLA de atraso, que diferente das violações de privacidade, foi contabilizada por usuário, e não por serviço, o que dificulta uma análise mais profunda desta métrica. Outra limitação no uso foi encontrada em cenários lotados, nos quais a migração, por ocorrer apenas no fim de cada passo da simulação, não libera, ou sinaliza a possível liberação de recursos de um servidor, de modo que nestes cenários, os serviços fiquem presos em um servidor, não podendo realizar migrações que melhorariam os resultados do cenário. Adicionalmente, o simulador não é capaz de operar com passos de duração maior que um minuto, isto pode ser uma limitação para implementações dependentes de tempo, como modelos de mobilidade.

5.5 CONSIDERAÇÕES PARCIAIS

Neste capítulo foram apresentados os cenários utilizados na simulação dos modelos, e então apresentados, explicados e discutidos os resultados obtidos. Por fim foi o simulador foi brevemente avaliado, apresentando seus pontos positivos, assim como limitações encontradas na realização dos experimentos.

6 CONCLUSÃO

Computação em nuvem é um modelo de processamento e armazenamento de dados extremamente utilizado atualmente, contudo conta com limitações de velocidade, decorrentes da distancia entre os grandes centros de processamento, as fontes de dados e usuários. Por outro lado, servidores *edge* contam com tempos de latência mais baixos, mas não apresentam a mesma capacidade de computação e memória dos servidores em nuvem.

A colaboração entre estes dois modelos é chamada de *cloud-edge continuum*, e busca utilizar das vantagens individuais de cada modelo para minimizar as limitações do conjunto. A utilização destes recursos em conjunto possibilita que diversas tecnologias existentes e emergentes sejam desenvolvidas, especialmente em aplicações que necessitam de tempos de latência menores, assim como é capaz de reduzir o custo energético e de utilização de servidores.

Desta colaboração surge a questão de como fazer a divisão de tarefas entre os diferentes recursos disponíveis em uma determinada rede. Esta divisão é feita por políticas de escalonamento, algoritmos que, utilizando parâmetros tanto das tarefas como dos servidores, decide onde e quando disponibilizar os recursos para execução destas tarefas. Neste trabalho foram levantados e estudados diferentes pesquisas que realizaram testes de políticas de escalonamento dentro e fora de redes *edge*.

Antes da utilização destas políticas em grande escala, é necessário testa-las, contudo, realizar estes testes em redes reais e ativas é lento e custoso. Para diminuir tanto o tempo quanto o custo destes testes, é possível converter as características destas redes em modelos, abstraindo seus componentes e comportamentos. Estes modelos, são inseridos em simuladores, para obter uma simulação do comportamento dos sistemas reais, permitindo a obtenção de dados para verificação das políticas em uma fração do tempo, e sem necessitar da implementação em larga escala.

Este trabalho realizou as simulações no *software* EdgeSimPy, um simulador em Python que incorpora as abstrações funcionais de servidores, dispositivos de rede e aplicações. Foram simulados e avaliados 3 modelos já existentes na literatura, e 2 modelos novos modelos foram desenvolvidos no decorrer dos experimentos, um destes modelos, o DCF se provou competitivo com os modelos existentes, tendo a distribuição de serviços como um diferencial que pode se provar interessante em um cenário de infraestrutura instável ou emergente.

Os modelos estão disponíveis no link: <https://github.com/EduardoPandini>

REFERÊNCIAS

ALWASEL, Khaled et al. Iotsim-osmosis: A framework for modeling and simulating iot applications over an edge-cloud continuum. **Journal of Systems Architecture**, v. 116, p. 101956, 2021. ISSN 1383-7621. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1383762120302083>. Citado na página 22.

AMARASINGHE, Gayashan et al. Ecsnet++ : A simulator for distributed stream processing on edge and cloud environments. **Future Generation Computer Systems**, v. 111, p. 401–418, 2020. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X19300494>. Citado na página 22.

ARAL, Atakan; MAIO, Vincenzo De; BRANDIC, Ivona. Ares: Reliable and sustainable edge provisioning for wireless sensor networks. **IEEE Transactions on Sustainable Computing**, IEEE, v. 7, n. 4, p. 761–773, 2021. Citado 2 vezes nas páginas 23 e 39.

BAI, Fan; HELMY, Ahmed. A survey of mobility models in wireless adhoc networks. **Wireless ad hoc and sensor networks**, Kluwer Academic Publishers Dordrecht, p. 1–30, 2004. Citado na página 27.

BELOGLAZOV, Anton; ABAWAJY, Jemal; BUYYA, Rajkumar. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. **Future generation computer systems**, Elsevier, v. 28, n. 5, p. 755–768, 2012. Citado na página 39.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numer. Math.**, Springer-Verlag, Berlin, Heidelberg, v. 1, n. 1, p. 269–271, dec 1959. ISSN 0029-599X. Disponível em: <https://doi.org/10.1007/BF01386390>. Citado na página 39.

DINH, Thinh Quang et al. Online resource procurement and allocation in a hybrid edge-cloud computing system. **IEEE transactions on wireless communications**, IEEE, v. 19, n. 3, p. 2137–2149, 2020. Citado na página 28.

FATICANTI, Francescomaria et al. Deployment of application microservices in multi-domain federated fog environments. In: IEEE. **2020 international conference on omni-layer intelligent systems (COINS)**. [S.l.], 2020. p. 1–6. Citado 2 vezes nas páginas 32 e 33.

GUO, Fengxian et al. Adaptive resource allocation in future wireless networks with blockchain and mobile edge computing. **IEEE Transactions on Wireless Communications**, IEEE, v. 19, n. 3, p. 1689–1703, 2019. Citado na página 29.

GUO, Hongzhi; LIU, Jiajia. Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks. **IEEE Transactions on Vehicular Technology**, IEEE, v. 67, n. 5, p. 4514–4526, 2018. Citado na página 28.

HONG, Zicong et al. Multi-hop cooperative computation offloading for industrial iot–edge–cloud computing environments. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 30, n. 12, p. 2759–2774, 2019. Citado na página 28.

IORGA, Michaela et al. **Fog Computing Conceptual Model**. [S.l.]: Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2018. Citado na página 14.

ISMAIL, Leila; MATERWALA, Huned. Escove: energy-sla-aware edge–cloud computation offloading in vehicular networks. **Sensors**, MDPI, v. 21, n. 15, p. 5233, 2021. Citado na página 39.

JHA, Devki Nandan et al. Iotsim-edge: a simulation framework for modeling the behavior of internet of things and edge computing environments. **Software: Practice and Experience**, Wiley Online Library, v. 50, n. 6, p. 844–867, 2020. Citado na página 22.

KHALILI, Ata; ZARANDI, Sheyda; RASTI, Mehdi. Joint resource allocation and offloading decision in mobile edge computing. **IEEE Communications Letters**, IEEE, v. 23, n. 4, p. 684–687, 2019. Citado na página 29.

KUANG, Zhufang et al. Partial offloading scheduling and power allocation for mobile edge computing systems. **IEEE Internet of Things Journal**, IEEE, v. 6, n. 4, p. 6774–6785, 2019. Citado na página 29.

LERA, Isaac; GUERRERO, Carlos; JUIZ, Carlos. Yafs: A simulator for iot scenarios in fog computing. **IEEE Access**, v. 7, p. 91745–91758, 2019. Citado na página 22.

LI, Lan et al. An energy-aware task offloading mechanism in multiuser mobile-edge cloud computing. **Mobile Information Systems**, Hindawi, v. 2018, 2018. Citado na página 29.

LIANG, Chengchao et al. Energy-efficient resource allocation in software-defined mobile networks with mobile edge computing and caching. In: IEE E. **2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)**. [S.l.], 2017. p. 121–126. Citado na página 29.

LOPEZ, Pedro Garcia et al. **Edge-centric computing: Vision and challenges**. [S.l.]: ACM New York, NY, USA, 2015. 37–42 p. Citado na página 14.

LUO, Quyuan et al. Resource scheduling in edge computing: A survey. **IEEE Communications Surveys & Tutorials**, IEEE, v. 23, n. 4, p. 2131–2165, 2021. Citado 4 vezes nas páginas 14, 15, 16 e 28.

MACQUEEN, James et al. Some methods for classification and analysis of multivariate observations. In: OAKLAND, CA, USA. **Proceedings of the fifth Berkeley symposium on mathematical statistics and probability**. [S.l.], 1967. v. 1, n. 14, p. 281–297. Citado na página 39.

MAHMUD, Redowan et al. ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments. **Journal of Systems and Software**, v. 190, p. 111351, 2022. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121222000863>. Citado na página 22.

MELL, Peter; GRANCE, Tim et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011. Citado 2 vezes nas páginas 12 e 14.

PANDINI, Eduardo et al. Uma proposta para avaliacção de confiança em redes vanets. In: SBC. **Anais Estendidos do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**. [S.l.], 2022. p. 304–311. Citado na página 16.

PULIAFITO, Carlo et al. Mobfogsim: Simulation of mobility and migration for fog computing. **Simulation Modelling Practice and Theory**, v. 101, p. 102062, 2020. ISSN 1569-190X. Modeling and Simulation of Fog Computing. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1569190X19301935>. Citado na página 22.

QAYYUM, Tariq et al. Fognetsim++: A toolkit for modeling and simulation of distributed fog environment. **IEEE Access**, v. 6, p. 63570–63583, 2018. Citado na página 22.

QIAN, Li Ping et al. Noma-enabled mobile edge computing for internet of things via joint communication and computation resource allocations. **IEEE Internet of Things Journal**, IEEE, v. 7, n. 1, p. 718–733, 2019. Citado na página 29.

SONMEZ, Cagatay; OZGOVDE, Atay; ERSOY, Cem. Edgecloudsim: An environment for performance evaluation of edge computing systems. **Transactions on Emerging Telecommunications Technologies**, Wiley Online Library, v. 29, n. 11, p. e3493, 2018. Citado na página 22.

SOUZA, Paulo et al. Thea-a qos, privacy, and power-aware algorithm for placing applications on federated edges. In: IEEE. **2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)**. [S.l.], 2023. p. 136–143. Citado 5 vezes nas páginas 28, 29, 32, 33 e 34.

SOUZA, Paulo et al. Latency-aware privacy-preserving service migration in federated edges. In: **CLOSER**. [S.l.: s.n.], 2022. p. 288–295. Citado 2 vezes nas páginas 32 e 33.

SOUZA, Paulo S; FERRETO, Tiago; CALHEIROS, Rodrigo N. Edgesimpy: Python-based modeling and simulation of edge computing resource management policies. **Future Generation Computer Systems**, Elsevier, 2023. Citado 5 vezes nas páginas 12, 17, 22, 23 e 27.

STEINMAN, Jeffrey S et al. Global virtual time and distributed synchronization. **ACM SIGSIM Simulation Digest**, ACM New York, NY, USA, v. 25, n. 1, p. 139–148, 1995. Citado 2 vezes nas páginas 20 e 27.

TAN, Zhiyuan et al. Virtual resource allocation for heterogeneous services in full duplex-enabled scns with mobile edge computing and caching. **IEEE Transactions on Vehicular Technology**, IEEE, v. 67, n. 2, p. 1794–1808, 2017. Citado na página 29.

TANG, Jiangjun; LEU, George; ABBASS, Hussein A. Introduction to fundamentals of simulation. Wiley-IEEE Press, 2020. Citado 2 vezes nas páginas 12 e 20.

WANG, Pengfei et al. Hetmec: Latency-optimal task assignment and resource allocation for heterogeneous multi-layer mobile edge computing. **IEEE Transactions on Wireless Communications**, IEEE, v. 18, n. 10, p. 4942–4956, 2019. Citado na página 28.

WANG, Xiaofei et al. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. **Ieee Network**, IEEE, v. 33, n. 5, p. 156–165, 2019. Citado na página 29.

WANG, Yue et al. Effective capacity-based resource allocation in mobile edge computing with two-stage tandem queues. **IEEE Transactions on Communications**, IEEE, v. 67, n. 9, p. 6221–6233, 2019. Citado na página 29.

XIANG, Xiaorong et al. Verification and validation of agent-based scientific simulation models. In: THE SOCIETY FOR MODELING AND SIMULATION INTERNATIONAL SAN DIEGO, CA, USA. **Agent-directed simulation conference**. [S.l.], 2005. v. 47, p. 55. Citado na página 27.

XING, Hong et al. Joint task assignment and resource allocation for d2d-enabled mobile-edge computing. **IEEE Transactions on Communications**, IEEE, v. 67, n. 6, p. 4193–4207, 2019. Citado na página 29.

XU, Jie; CHEN, Lixing; ZHOU, Pan. Joint service caching and task offloading for mobile edge computing in dense networks. In: IEEE. **IEEE INFOCOM 2018-IEEE Conference on Computer Communications**. [S.l.], 2018. p. 207–215. Citado na página 29.

XU, Xiaolong et al. A computation offloading method over big data for iot-enabled cloud-edge computing. **Future Generation Computer Systems**, Elsevier, v. 95, p. 522–533, 2019. Citado na página 29.

YU, Shuai et al. Computation offloading with data caching enhancement for mobile edge computing. **IEEE Transactions on Vehicular Technology**, IEEE, v. 67, n. 11, p. 11098–11112, 2018. Citado na página 29.

ZHAO, Pengtao et al. Context-aware tdd configuration and resource allocation for mobile edge computing. **IEEE Transactions on Communications**, IEEE, v. 68, n. 2, p. 1118–1131, 2019. Citado na página 29.

ZHOU, Yuchen et al. Resource allocation for information-centric virtualized heterogeneous networks with in-network caching and mobile edge computing. **IEEE Transactions on Vehicular Technology**, IEEE, v. 66, n. 12, p. 11339–11351, 2017. Citado na página 29.