OFFLOADING DE APLICAÇÕES ESTRUTURADAS COMO DAGS PARA A BORDA DA REDE

Gabriel Fontes Carvalho de Queiroz

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: José Ferreira de Rezende
Valmir Carneiro Barbosa

Rio de Janeiro
Fevereiro de 2024

OFFLOADING DE APLICAÇÕES ESTRUTURADAS COMO DAGS PARA A BORDA DA REDE

Gabriel Fontes Carvalho de Queiroz

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: José Ferreira de Rezende
Valmir Carneiro Barbosa

Aprovada por: Prof. José Ferreira de Rezende
Prof. Valmir Carneiro Barbosa
Prof.ª Rosa Maria Meri Leão
Prof. Célio Vinicius Neves de Albuquerque
Prof.ª Sand Luz Corrêa

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2024

Queiroz, Gabriel Fontes Carvalho de

Offloading de aplicações estruturadas como DAGs para a borda da rede/Gabriel Fontes Carvalho de Queiroz. – Rio de Janeiro: UFRJ/COPPE, 2024.

XIV, 81 p.: il.; 29, 7cm.

Orientadores: José Ferreira de Rezende

Valmir Carneiro Barbosa

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2024.

Referências Bibliográficas: p. 69 – 81.

1. Multi-access Edge Computing. 2. Distribuição de aplicações. 3. DAGs. I. Rezende, José Ferreira de *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

A meus pais,
com amor e gratidão.

Agradecimentos

Agradeço primeiramente a Deus, em todas as suas manifestações e nomes sagrados, pela infinitude de misericórdia, amor, carinho e bondade para me ajudar a seguir meu destino com fé.

Agradeço à minha família com muito amor. A meus pais, por todo amparo, cuidado e por sempre me incentivarem no caminho do estudo. Em especial, à minha mãe, por ser minha inspiração na vida e na profissão e me ensinar desde pequeno sobre a educação e o amor. Obrigado, mãe, por me inspirar a seguir nessa carreira tão linda que é ensinar, meu ofício que faço com amor e zelo para trazer luz na vida dos meus queridos alunos. Agradeço a meus sobrinhos por serem a fonte de alegria e leveza nos nossos encontros, sempre renovando minhas energias para continuar na caminhada do doutorado. Em especial, agradeço à minha amada avó Catarina, por todos os ensinamentos e acolhimento.

Minha gratidão será eterna a vocês dois, meus queridos orientadores, professor Rezende e professor Valmir. Não há cesta de chocolate que mensure o quanto vocês são importantes na minha vida. Obrigado, antes de mais nada, por me aceitarem como aluno, por terem me estendido a mão e me ensinado o caminho da ciência e da pesquisa. Obrigado por todas as reuniões leves e descontraídas. Eu peço que Deus permita que eu possa orgulhá-los por toda a minha carreira como professor e pesquisador, pois eu quero dar continuidade aos seus ensinamentos. Espero que possamos trabalhar juntos em vários projetos e publicações depois que eu concluir o curso. Vocês não sabem o tamanho da estima que eu tenho por vocês.

Agradeço a meus amigos do PEE, do PEMM e do PESC pelo conforto e pelas palavras de apoio em todos os momentos, principalmente quando eu mais precisei. Obrigado pela torcida. Obrigado por me lembrar que desistir não era uma opção, Dianne. Obrigado por ser a primeira amiga que fiz no LAND, Joanna. Obrigado pela amizade e pelo abrigo, no sentido literal, Gaspare e Érica. Obrigado por sempre me fazerem companhia, Beatriz, Luiza e Mariana. Obrigado por me lembrarem que tudo passa! Até a uva, não é, Ana Elisa? Amigos de sala de aula, de mestrado, de doutorado e de toda a vida. Agradeço a todos vocês: amigos que hoje já são professores, amigos de comer pizza e bater papo, amigos do trabalho e amigos que estão no exterior. Claro, eu não poderia deixar de mencionar, obrigado aos melho-

res grupos de carona da história da UFRJ, Muda/Usina e Grajaú/Vila, vocês são demais! Vocês que viabilizaram a minha ida ao Fundão por todos esses anos, muito obrigado!

Agradeço a todos os professores do PESC, principalmente do LAND e a professora Marcia Fampa, da Otimização, e a todos que estiveram comigo nesse tempo do doutorado. Edmundo e Rosa, obrigado pelos almoços de sempre! Meu obrigado mais que especial para nossa querida Carolzinha, nossa eterna madrinha do LAND, que foi a primeira a me receber no laboratório com um abraço caloroso e um lindo sorriso. Obrigado pelos parabéns de aniversário e por todos os cafés natalinos de confraternização sempre! Carol, você é a doçura do LAND!

Agradeço à minha instituição de trabalho, CEFET/RJ, e, em especial, aos meus queridos alunos das Engenharias e aos professores do CCGTEL, minha coordenação. Obrigado pela compreensão quanto à minha carga-horária intensa por todos esses anos. Obrigado pelo apoio nas tarefas do dia a dia. Trabalhar em nossa coordenação é um bálsamo e eu fico muito grato por poder ter esse nosso espaço. Somos um grupo pequeno, nosso corpo discente pode não ser o maior em número, mas cada um vale ouro. Meus alunos, especialmente os meus orientados, eu me esforço muito para lembrar os nomes de todos vocês e ensiná-los tudo o que eu posso passar para vocês. Eu me importo com cada um de vocês, quero que sigam seus destinos, suas vocações. Eu desejo do fundo do meu coração que vocês encontrem aquilo que os faça verdadeiramente felizes na vida. Amo vocês e obrigado por entenderem o malabarismo que eu faço com meus horários para conseguir atendê-los, mesmo que no fuso-horário de Tóquio. Obrigado por sempre me incentivarem e perguntarem sobre minha tese e minha defesa!

Agradeço aos membros da banca, professora Sand, da UFG, professor Célio, da UFF, e professora Rosa, da UFRJ, pela disponibilidade de tempo e esforço para tecerem os apontamentos necessários ao melhoramento da minha tese de doutorado. Agradeço ao professor Igor, também da UFF, que esteve comigo também nessa caminhada e fez parte da banca da minha qualificação junto à professora Rosa. Quanto mais se lapida o trabalho, mais ele se torna um belo diamante.

Por fim, gostaria de agradecer às agências de fomento, CNPq, CAPES, FAPERJ e FAPESP pelo financiamento parcial da pesquisa desta tese de doutorado.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

OFFLOADING DE APLICAÇÕES ESTRUTURADAS COMO DAGS PARA A BORDA DA REDE

Gabriel Fontes Carvalho de Queiroz

Fevereiro/2024

Orientadores: José Ferreira de Rezende
Valmir Carneiro Barbosa

Programa: Engenharia de Sistemas e Computação

O avanço tecnológico recente impulsiona o surgimento de novas aplicações, tais como jogos e *streaming* de vídeo, que apresentam requisitos intensivos de processamento, latência etc. Executar essas aplicações pode não ser viável nem no dispositivo móvel de usuário, devido a limitações de *hardware*, e nem na nuvem, devido à latência. Uma solução para isso é a computação de borda, um paradigma emergente no qual as aplicações são executadas em servidores de borda, próximos do usuário, podendo reduzir seu tempo de conclusão (*makespan*). No entanto, o descarregamento (*offloading*) de aplicações por completo para o servidor de borda pode sobrecarregá-lo ou congestionar o canal de comunicação. Como as aplicações de rede podem ser estruturadas como tarefas e suas trocas de dados, formando *Directed Acyclic Graphs* (DAGs), o *offloading* pode ser feito para apenas parte das tarefas. Particionar aplicações em tarefas e escolher quais tarefas devem ou não ser *offloaded* é um problema difícil, que não se resolve em tempo polinomial. Esta tese propõe, portanto, a heurística *Transmission-Avoiding fLexible OffloadiNg* (TALON) para encontrar decisões de *offloading* com *makespan* baixo para DAGs genéricos. São extraídos 1322 DAGs realistas do *Alibaba Cluster Trace Program*, para os quais TALON busca decisões de *offloading* e é comparada à sua referência principal e a duas soluções triviais. O *makespan* alcançado por TALON está apenas 3,9% a 8,9% acima do *makespan* ótimo em todos os casos. Inclusive, no caso multiusuário, TALON atinge um *makespan* equivalente ao da referência principal utilizando oito vezes menos recursos. TALON também supera de longe as soluções triviais. Além disso, TALON não onera o servidor de borda, dividindo a execução entre o dispositivo móvel e a borda, tal qual a decisão ótima, com a qual TALON apresenta uma semelhança que chega a 87%.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

OFFLOADING DAG APPLICATIONS TO THE NETWORK EDGE

Gabriel Fontes Carvalho de Queiroz

February/2024

Advisors: José Ferreira de Rezende
Valmir Carneiro Barbosa

Department: Systems Engineering and Computer Science

Recent technological advances drive the emergence of new applications, such as gaming and video streaming, which have stringent requirements for processing, latency, etc. Processing these applications may not be feasible either on the user's mobile device, due to its hardware limitations, or on the cloud, due to high latency. Edge computing is a possible solution to this, as it is an emerging paradigm in which applications run on edge servers, which are closer to the user, and it can reduce their completion time (makespan). However, transferring (offloading) applications' execution entirely to the edge server can heavily burden it or congest the communication channel. As network applications can be structured as a set of tasks and their data transmissions, forming Directed Acyclic Graphs (DAGs), offloading can be done for only part of the tasks. The application partitioning into tasks and the decision about which tasks should or should not be offloaded is a difficult problem, which cannot be solved in polynomial time. Hence, this thesis proposes the heuristic Transmission-Avoiding fLexible OffloadiNg (TALON) to find offloading decisions with low makespan for generic DAGs. A set of 1322 realistic DAGs is extracted from Alibaba Cluster Trace Program, for which TALON makes offloading decisions, as TALON is compared with a main proposal from the literature and two baseline solutions. The makespan achieved by TALON is only 3.9% to 8.9% above the optimal makespan in all cases. In fact, in the multiuser case, TALON achieves a makespan equivalent to the one achieved by its competitor, but using eight times fewer resources than it. TALON also far outperforms the baseline solutions. Besides, TALON does not burden the edge server, balancing the task processing between the mobile device and the edge server, just like the optimal decision, with which TALON achieves up to 87% similarity.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

4G	*4th Generation* 11
5G	*5th Generation* 11
6G	*6th Generation* 18
AP	*Access Point* 5
AR	*Augmented Reality* 1
CMU	*Carnegie Mellon University* 15
CPU	*Central Processing Unit* 2
D2D	*Device-to-Device* 24
DAG	*Directed Acyclic Graph* 6
DNN	*Deep Neural Network* 24
E2E	*End-to-End* 19
eMBB	*enhanced Mobile BroadBand* 18
ETSI	*European Telecommunications Standards Institute* 11
FIFO	*First In, First Out* 25
FLOP	*FLoating-point OPerations* 47
HOA	*Heuristic Offloading Algorithm* 51
IA	Inteligência Artificial 4
IoT	*Internet of Things* 1
ITU	*International Telecommunication Union* 19

LTE	*Long Term Evolution*	54
MCC	*Mobile Cloud Computing*	23
MEC	*Multi-access Edge Computing*	5
mMTC	*massive Machine Type Communication*	18
OEC	*Open Edge Computing*	15
OFC	*Open Fog Consortium*	15
P2P	*Peer-to-Peer*	19
PODAG	Posicionamento e *Offloading* de DAGs	7
QoE	*Quality of Experience*	3
QoS	*Quality of Service*	13
RAN	*Radio Access Network*	11
TALON	*Transmission-Avoiding fLexible OffloadiNg*	9
uRLLC	*ultra Reliable Low Latency Communication*	18
VR	*Virtual Reality*	1

Capítulo 1

Introdução

O crescente avanço das tecnologias de comunicação permite que a sociedade esteja cada vez mais conectada, trocando mais informação em menos tempo e a todo o tempo. Paradigmas novos de rede expandem os horizontes para novas aplicações de rede. Um exemplo é a *Internet of Things* (IoT) [1], que viabiliza aplicações para *smart homes* e monitoramento de saúde, por exemplo. A IoT renova o conceito de dispositivo de rede e passa a incluir quaisquer tipos de objetos inteligentes que disponham de capacidade de comunicação, além de apenas *notebooks*, *desktops* ou servidores [1, 2].

A ascenção recente de novas aplicações e serviços de rede, como os jogos *mobile* [3, 4], ambas *Augmented Reality* (AR) e *Virtual Reality* (VR) [5], a telemedicina [2, 6, 7], a automação industrial [8], as redes sociais móveis [9, 10], o *streaming* de vídeo ao vivo [11–13] e o metaverso [14], por exemplo, representam a aurora de uma nova época para os sistemas de telecomunicações existentes.

Isso se deve ao fato de que entregar ao usuário uma experiência de baixa latência, imersiva, responsiva e de conexão ubíqua depende de diferentes fatores, como alta capacidade de processamento, armazenamento em disco, disponibilidade do dispositivo e uma alta taxa de transmissão de dados [15]. No entanto, alcançar essas características em geral não é possível atualmente, pois os dispositivos móveis dos usuários que executam as aplicações não possuem poder de processamento, memória e largura de banda suficientes para tal.

A execução dessas novas aplicações em *smartphones* ou óculos de AR/VR, por exemplo, pode levar também a um alto consumo de energia, esgotando a bateria em menos tempo e diminuindo o tempo que o usuário pode utilizar o serviço antes de precisar recarregar o dispositivo. Outra limitação também é o aquecimento dos dispositivos, o qual, no caso de óculos de AR/VR, causa também desconforto para o usuário.

Muitas das aplicações móveis são altamente populares, como os jogos Pokémon GO[1], que contém elementos de AR, e Minecraft[2]. Já outras aplicações são essenciais para as pessoas, como a assistência cognitiva [16] viabilizada pelo uso de dispositivos *wearable*. Com isso, a sociedade evolui e adere massivamente às novas aplicações para *smartphones* e IoT, demandando mais das tecnologias de comunicação e retroalimentando a busca pelo avanço tecnológico.

Especificamente no caso de jogos, essa categoria pode ser vista como uma *killer application* [17] pelo uso intensivo de recursos gráficos, uso de rede em modos *online* cooperativos e competitivos, o que exige muito dos recursos limitados de *hardware* e *software* dos dispositivos *mobile*. Satyanarayanan [18] recorda que a própria *World Wide Web* foi uma *killer application* na década de 90, dependendo de uma infraestrutura suficiente de Internet para seu desenvolvimento.

Um dos requisitos mais frequentes dessas novas aplicações é o baixo tempo de resposta, dependendo de uma comunicação *online* e de forma interativa com o usuário. A fluidez na execução dessas aplicações é um fator crítico. No entanto, as restrições de *hardware* associadas aos dispositivos *mobile* e de IoT são conhecidas e frequentemente ressaltadas na literatura ao longo dos últimos anos [19–22]. Isso dificulta o processamento dessas aplicações no próprio dispositivo e pode ser custoso em termos de tempo de execução e vida útil da bateria, que geralmente não estão ligados à rede elétrica durante seu uso.

Muito embora seja possível argumentar que os dispositivos móveis estejam em constante desenvolvimento, como os *tablets* e *smartphones* de última geração, e sejam capazes de executar de forma satisfatória as novas aplicações de rede, isso pode trazer desvantagens notáveis: ter um longo tempo de execução, tornar o dispositivo lento e pouco responsivo aos comandos do usuário, aquecer o dispositivo e esgotar a bateria [23].

Uma possível forma de superar as limitações de *hardware* dos dispositivos móveis é o descarregamento (*offloading*) computacional, uma vez que o processamento da aplicação é mais lento no próprio dispositivo, para um ponto na rede com alto poder computacional, no qual a execução seja mais rápida [24]. Isso alivia a demanda imposta ao dispositivo móvel do usuário, pois passam a ser utilizados os recursos de *Central Processing Unit* (CPU), memória e armazenamento, por exemplo, de um servidor remoto localizado no núcleo da rede.

O princípio de migração do processamento é usado há décadas, seja na submissão de tarefas a um *cluster* de servidores, uma *grid* ou uma nuvem [20, 25–27], seja na colaboração direta entre nós em uma comunicação *Peer-to-Peer* (P2P) [28, 29]. Os próprios serviços de reconhecimento de voz da Google e a Siri da Apple descarregam

[1] `https://pokemongolive.com/`
[2] `https://www.minecraft.net`

Figura 1.1: Localização dos servidores da Google Cloud[3].

o processamento para a nuvem [18]. No entanto, embora o tempo de execução seja reduzido em servidores de nuvem, é necessário considerar o tempo de transmissão, que pode ser muito alto. A Figura 1.1 mostra, por exemplo, a localização dos servidores da Google Cloud.

A menos que um usuário esteja próximo o bastante de algum dos pontos destacados na figura, o tempo de transmissão pode se tornar tão grande que a redução esperada no tempo de processamento não seja suficiente para justificar o *offloading* para uma nuvem. Por exemplo, aplicações que tenham requisitos estritos de latência, como AR/VR e telemedicina, podem ser severamente prejudicadas se a transmissão dos dados necessários para a execução levar muito tempo, o que pode acontecer caso o servidor esteja muito distante do usuário, a quantidade de dados para transmissão seja muito grande ou a taxa de dados no canal de comunicação seja baixa. Todos esses fatores contribuem para uma piora na *Quality of Experience* (QoE) do usuário [30].

1.1 Motivação

É possível que a infraestrutura de computação em nuvem nos moldes tradicionais possa não conseguir atender aos requisitos das novas aplicações que sejam computacionalmente intensivas ou sensíveis a latência, além de possivelmente ficar sobrecarregada no futuro com o aumento vertiginoso no número de usuários executando

[3] `https://cloud.google.com/about/locations/`

aplicações *mobile* [24]. Tampouco os próprios dispositivos móveis são capazes de executar essas aplicações sem incorrer em esgotamento de bateria e demais desvantagens apontadas anteriormente.

Com a expectativa de haver 30 bilhões de dispositivos conectados, dos quais 13 bilhões sejam dispositivos móveis, até o final de 2023 feita pela Cisco [31], os servidores de nuvem existentes podem não ter capacidade de abarcar a crescente demanda computacional e de comunicação desses usuários. A Cisco [31] também aponta que aproximadamente 300 milhões de aplicativos serão baixados por usuários de dispositivos móveis também até o fim de 2023, incluindo aplicativos de redes sociais, finanças e jogos. Várias aplicações novas associadas à Inteligência Artificial (IA) geradora de conteúdo, como os *chatbots* Bard[4], da Google, e ChatGPT[5], da OpenAI, ou o criador de arte digital Midjourney, da Midjourney Inc., têm ganhado notoriedade recentemente e já apresentam lucro em um mercado recém-criado[6][7].

A Figura 1.2 mostra o interesse das pessoas por essas novas aplicações na última década com base nos números de buscas da ferramenta Google Trends. A popularidade é calculada de forma relativa, atribuindo o valor 100 à semana que apresenta o maior número de pesquisas sobre a categoria (assunto ou campo de estudo), independente do número absoluto de pesquisas sobre esse tópico. Os resultados refletem o interesse em escala mundial para as aplicações mostradas.

Enquanto jogos *online*, VR e redes sociais se mantêm como interesses sólidos ao longo da década, o *streaming* se destaca com picos de popularidade no começo do ano de 2020 e um crescimento relativamente maior, em média, a partir disso. Por outro lado, dentre as aplicações em questão, a IA é definitivamente o assunto mais popular no mundo até então, em rápido crescimento desde 2021, e é por volta de três vezes mais popular do que o *streaming* ao vivo até o presente momento. Inclusive, a parte tracejada ao final da curva referente à IA indica uma estimativa do valor real, pois os dados de pesquisa disponíveis até o momento são tidos como parciais ou incompletos.

Como a geração dos dados de aplicação ocorre na extremidade da rede, nos dispositivos móveis dos usuários, que estão conectados a redes locais domésticas e institucionais ou a redes móveis, esses dados precisam ser transmitidos até os servidores de nuvem nos quais devem ser processados. Esse aumento no número de usuários e de tráfego das aplicações implica em um maior sobrecarga de processamento, tamanho de fila em *buffers* de transmissão e escassez de largura de banda

[4] https://bard.google.com/

[5] https://openai.com/chatgpt

[6] https://www.theregister.com/2022/08/01/david_holz_midjourney/

[7] https://www.euronews.com/culture/2021/11/30/botto-the-robot-creating-works-of-art-makes-its-first-million-at-auction

[8] https://trends.google.com/trends/

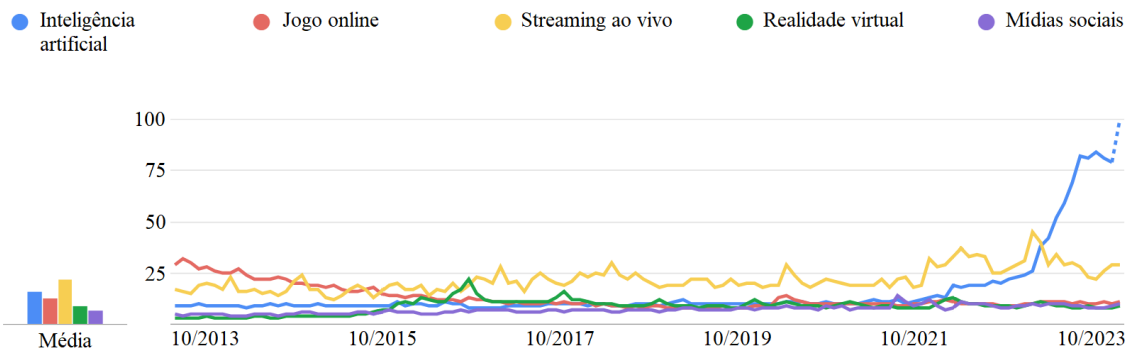


Figura 1.2: Popularidade das novas aplicações na ferramenta Google Trends[8].

nos enlaces que levam aos servidores de nuvem [24]. Dessa forma, uma abordagem diferente se faz útil para lidar com esse problema.

É nesse contexto que a *Multi-access Edge Computing* (MEC) [32], ou apenas computação de borda, surge como uma chave para resolver os problemas da infraestrutura de rede atual, de novos serviços e aplicações de rede e do aumento massivo do número de usuários [33]. A MEC é um paradigma de rede que traz parte da capacidade de computação, armazenamento e comunicação presente nos servidores de alto custo do núcleo da rede, como os *data centers* das nuvens, para a borda da rede, onde as aplicações de rede passam a ser executadas [30].

O conceito de borda contempla todos os dispositivos com capacidade de computação e comunicação de rede que estejam entre os dispositivos de usuário ou de IoT e os *data centers* de uma nuvem [33]. Os dispositivos da borda, ou servidores de borda, geralmente estão a um salto de distância dos usuários finais, próximos a estações base da rede móvel ou *Access Points* (APs) sem-fio [32, 34], e podem ser servidores *off-the-shelf* para oferecer serviços e executar as aplicações dos usuários. Com isso, a MEC pode superar as limitações físicas dos dispositivos móveis para a execução de aplicações de rede mais intensivas e ainda ser capaz de reduzir a latência e o *jitter* e desafogar os enlaces de acesso ao núcleo da rede [30, 35].

Em termos de disponibilidade, a borda pode servir os dispositivos que, por exemplo, tenham conexões intermitentes para a nuvem e prover serviços mesmo que os servidores de nuvem estejam em manutenção. Assim, a borda também pode atuar como um *proxy* para a nuvem e evitar que o serviço fique indisponível [18, 36]. No final, todas essas características desejáveis se refletem no aumento da QoE, implicando em ganhos para usuários e operadoras.

1.2 Problema Atacado

Apesar dos benefícios evidentes e do crescente interesse pela MEC, depender de um servidor de borda para executar aplicações de usuário e lidar com a demanda de múltiplos usuários pode ser muito oneroso [37]. Os servidores de borda não possuem tanta capacidade quanto os servidores de nuvem e a escala de provisionamento de serviços de um servidor de borda é reduzida [38], já que o mesmo visa atender usuários mais próximos de suas redondezas, diferentemente dos servidores de nuvem.

Essa escassez de recursos possibilita o surgimento de dois problemas. Primeiro, se há muitos usuários na região dependendo unicamente de um servidor de borda, esse excesso de demanda pode sobrecarregá-lo e atrasá-lo, o que pode resultar em um desempenho insatisfatório em termos de latência. Isso pode levar a um aumento do tempo de execução da aplicação até sua conclusão, aqui definido como *makespan*, o que é ainda mais crítico no caso de uma aplicação sensível a latência. Segundo, o descarregamento de uma aplicação para que ela seja totalmente executada no servidor de borda pode gerar uma subutilização dos recursos físicos do dispositivo móvel de usuário. Essa subutilização ocorre porque o dispositivo móvel não contribui para o processamento da aplicação e fica ocioso, apenas aguardando pelo fim da execução no servidor de borda e pelo recebimento do resultado desse processamento.

Para reduzir o makespan das aplicações e também a carga imposta ao servidor de borda, é possível particionar as aplicações em componentes menores executáveis, como funções ou classes em um código, aqui definidas como tarefas, e permitir que algumas delas sejam executadas no próprio dispositivo móvel. Esse processo é conhecido como *offloading* parcial da execução de aplicações na literatura de MEC [30].

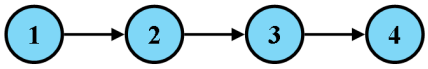
O particionamento em tarefas torna possível estruturar uma aplicação de rede não mais como um bloco monolítico, mas como um *Directed Acyclic Graph* (DAG) [39]. Esses DAGs são compostos pelas tarefas (vértices) e as transferências de dados entre as mesmas (arestas), que formam as dependências da ordem de execução na aplicação. Em termos de granularidade, esse particionamento da aplicação pode, por exemplo, alcançar o nível de métodos no código da aplicação [40]. Quando uma das tarefas do DAG de aplicação é enviada para execução no servidor de borda, diz-se que ocorreu o *offloading* daquela tarefa.

O processo de particionamento de uma aplicação em tarefas e decisão sobre quais delas devem ser descarregadas para a borda ou executadas localmente é um problema difícil e reconhecido na literatura de MEC [41]. Este é o problema atacado no contexto desta tese, aqui definido como problema de Particionamento e *Offloading* de DAGs de aplicação (PODAG), cujo objetivo é a redução do *makespan* dos DAGs de aplicação. Conforme dito anteriormente, muitas aplicações emergentes, tais como jogos, AR/VR e telemedicina, precisam de um tempo de execução baixo. Isso é

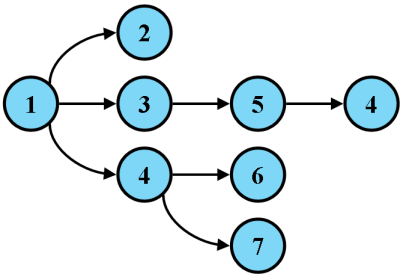
essencial para melhorar, por exemplo, a responsividade do *hardware* aos comandos do usuário e atender aos requisitos de latência necessários para fornecer uma boa QoE. Embora o consumo de energia também seja um fator importante, esta tese prioriza a redução do *makespan*. Para isso, são consideradas aplicações cuja execução não tenha uma duração longa a ponto de esgotar totalmente a bateria do dispositivo de usuário. Adicionalmente, considera-se que o servidor de borda, para o qual as tarefas são *offloaded*, esteja conectado à rede elétrica, de modo que o consumo de energia não seja uma limitação para o mesmo [30].

Segundo Lin *et al.* [17], dada a execução distribuída das aplicações de rede pelos dispositivos móveis e de borda, é desafiador realizar um particionamento do código das aplicações de forma robusta e flexível. Na prática, uma aplicação pode ser representada por uma miríade de estruturas diferentes de DAGs, conforme exemplificado pela Figura 1.3, o que torna o problema PODAG ainda mais desafiador. De fato, conforme mostrado por Mao *et al.* [30] e Guo *et al.* [42], o problema PODAG pode ser escrito como um problema de programação inteira mista, no qual as decisões de *offloading* para as tarefas são variáveis binárias: ou a execução ocorre no dispositivo móvel ou no servidor de borda. Esse tipo de problema é conhecidamente NP-difícil e as soluções para o mesmo ainda estão em sua infância, com soluções apenas para aplicações estruturadas como DAGs sequenciais, na forma de caminhos, ou de árvores, como nas Figuras 1.3a e 1.3b. Mao *et al.* [30] ainda frisam que soluções para aplicações modeladas como grafos de estruturas mais complexas ou genéricas, como nas Figura 1.3c e 1.3d, permanecem pouco exploradas.

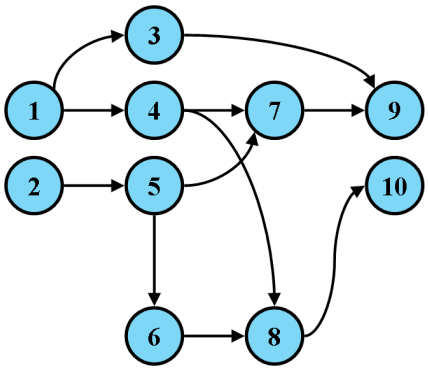
Existem muitos trabalhos disponíveis na literatura que já atacaram versões correlatas ou simplificadas do problema PODAG [39, 43–46]. No entanto, esta tese destaca três aspectos pouco explorados no estado da arte. Primeiro, as limitações de largura de banda no canal de comunicação e de número de CPUs disponíveis tanto nos dispositivos móveis quanto nos servidores de borda geralmente não são consideradas [24]. Essa simplificação assume que os recursos estão amplamente disponíveis, o que não é realista, pois quanto maior é o número de transmissões ou tarefas ocorrendo simultaneamente, menor é a fração de largura de banda do canal e de alocação de CPU, respectivamente, que cada uma delas recebe individualmente, potencialmente aumentando o tempo de conclusão. Segundo, há uma falta de dados reais para caracterizar os DAGs de aplicação na literatura, que costuma apresentar DAGs com estruturas muito simplificadas [39, 47, 48] ou puramente sintéticos [49]. Esses DAGs representam casos particulares do problema geral e as soluções encontradas na literatura frequentemente são situacionais e não podem ser aplicadas a todos os casos [24]. Terceiro, há um consenso adotado há mais de uma década na literatura [50–53] conhecido como política *one-climb*, que é tida como uma regra aplicável a todos os tipos de DAGs de aplicação.



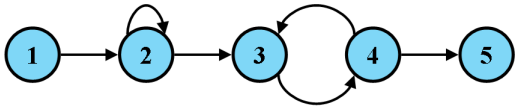
(a) Execução sequencial.



(b) Execução em árvore.



(c) Execução genérica.



(d) Execução com recursão de tarefas.

Figura 1.3: Distribuição dos grafos de acordo com suas propriedades.

A política *one-climb* professa que a decisão de *offloading* ótima, ou seja, a decisão sobre qual dispositivo executa qual tarefa para fornecer o menor *makespan* possível para a aplicação, nunca deve ter tarefas marcadas para execução no dispositivo móvel entre duas ou mais tarefas marcadas para *offloading* no servidor de borda. Isto é, em uma sequência de tarefas, da primeira até a última, o processamento só pode migrar do dispositivo móvel para o servidor de borda uma vez, no máximo, e posteriormente retornar ao dispositivo móvel.

Até onde vai a pesquisa desta tese, esta é a primeira vez que a obrigatoriedade de se atender à política *one-climb* é confrontada. Um contraexemplo é elaborado para mostrar que a decisão de *offloading* ótima para um DAG de aplicação genérico não precisa seguir a política *one-climb*.

1.3 Objetivos

Os objetivos desta tese são:

- Conhecer os problemas associados ao *offloading* computacional de aplicações de rede presentes na literatura de computação de borda.
- Realizar o levantamento do estado da arte sobre computação de borda e técnicas de *offloading* computacional de aplicações de rede.

- Mostrar através de um contraexemplo que a solução ótima de *offloading* não obedece necessariamente a uma política *one-climb*.

- Propor uma heurística eficiente e suficientemente genérica para resolver o problema PODAG, considerando as limitações de largura de banda do canal e as limitações de processamento do dispositivo móvel e do servidor de borda.

- Obter dados reais de aplicações de rede para avaliar o desempenho da heurística proposta nesta tese e compará-la com soluções correlatas disponíveis na literatura.

1.4 Contribuições

Tendo em vista os desafios associados ao problema PODAG, esta tese propõe o algoritmo *Transmission-Avoiding fLexible OffloadiNg* (TALON). TALON gera um pequeno conjunto de decisões de *offloading*, cada uma ditando as tarefas do DAG a serem executadas pelo dispositivo móvel e pelo servidor de borda.

As decisões de *offloading* são geradas em duas fases. A primeira fase é uma abordagem gananciosa que marca algumas tarefas para *offloading*. Isso é feito através da análise das tarefas aos pares, buscando tarefas entre as quais haja uma transmissão de dados intensiva de uma para a outra e marcando ambas para *offloading*, já que, se ambas estiverem no mesmo dispositivo, não há necessidade de transmissão no canal de comunicação. A segunda fase é a geração de todas as decisões de *offloading* possíveis envolvendo as tarefas que não foram marcadas para *offloading* na fase anterior. Por fim, todas as decisões geradas por TALON são testadas e aquela que fornecer o menor *makespan* é eleita como a decisão de *offloading* para o DAG da aplicação.

As contribuições desta tese são listadas a seguir.

- Formular o problema PODAG, que leva em consideração a disponibilidade de CPUs no dispositivo móvel e no servidor de borda e a limitação de largura de banda no canal de comunicação, para calcular o *makespan*.

- Elaborar um contraexemplo para mostrar que a decisão de *offloading* ótima não precisa obedecer à política *one-climb* em DAGs genéricos, ou seja, contraria um consenso utilizado amplamente na literatura. Inclusive, cerca de 17% das decisões ótimas para mais de mil DAGs de aplicação contrariam a política *one-climb*.

- Propor a heurística TALON para resolver o problema PODAG de forma eficiente e com baixo tempo de execução, alcançando uma decisão de *offloading*

que resulte em um *makespan* reduzido. Os resultados desta tese mostram que o *makespan* alcançado ao utilizar TALON são apenas 3,9 % a 8,9 % maiores que o *makespan* mínimo associado à decisão de *offloading* ótima em todos os cenários de simulação, que são: caso do usuário único, caso dos múltiplos usuários e caso das múltiplas taxas de transmissão de dados.

- Comparar e analisar a proximidade entre as decisões de *offloading* tomadas por TALON com a respectiva decisão ótima para cada DAG de aplicação. Em média, TALON alcança uma similaridade de até 87% com a decisão ótima. Além disso, TALON também não sobrecarrega o servidor de borda, apresentando uma taxa de *offloading* de tarefas em torno de 40% e um paralelismo de processamento de, em média, duas tarefas no servidor de borda. Ambos os valores são semelhantes aos da decisão ótima.

- Analisar o *Alibaba cluster trace* e extrair mais de mil DAGs de aplicação reais. Esses DAGs são utilizados para avaliar o desempenho da heurística TALON e compará-la com o principal competidor da literatura e com duas soluções triviais. TALON supera todas elas em todos os cenários testados.

1.5 Publicações

Todas as contribuições feitas nesta tese encontram-se disponíveis em um artigo completo [54] publicado na revista *Journal of Network and Computer Applications*, com fator de impacto 8,7 e classificação Qualis A1.

1.6 Organização do Texto

Esta tese está organizada da seguinte forma. O Capítulo 2 apresenta a fundamentação teórica sobre MEC e o *offloading* computacional, além de mostrar os trabalhos relacionados presentes no estado da arte. O Capítulo 3 define o problema PODAG, a modelagem do sistema e dos DAGs de aplicação, bem como a modelagem do cálculo de *makespan*. O Capítulo 4 discute a proposta da heurística TALON, mostrando o seu funcionamento e detalhes do algoritmo. Já no Capítulo 5, são mostradas a coleta e a caracterização dos DAGs de aplicação reais extraídas do *data set* do *Alibaba Cluster Trace Program*. No Capítulo 6, são apresentados os cenários de teste e os resultados obtidos com a avaliação do desempenho de TALON em comparação ao principal competidor da literatura e mais duas soluções triviais, sendo que TALON supera todas elas. Por fim, o Capítulo 7 conclui esta tese e apresenta as direções futuras de pesquisa a partir dos resultados obtidos.

Capítulo 2

Levantamento Bibliográfico

Este Capítulo apresenta uma visão holística da computação de borda e do *offloading* computacional. A Seção 2.1 define o conceito de computação de borda, introduz suas vantagens e desvantagens, discute as definições de paradigmas semelhantes à MEC, como a *fog computing* e as *cloudlets*. A Seção 2.2 apresenta aplicações de rede emergentes que podem se beneficiar com o avanço da computação de borda. Ao final, a Seção 2.3 discorre sobre os trabalhos relacionados a esta tese, versando sobre a teoria geral de *offloading* em MEC e os principais trabalhos que atacam versões correlatas do problema PODAG.

2.1 Computação de Borda

O conceito de computação de borda tem evoluído ao longo da última década [55], desde sua primeira proposta realizada em 2013 [56], em uma parceria entre Nokia e IBM, como uma plataforma de provisionamento de serviços a usuários de dispositivos móveis implementada nas estações base. Em sequência, uma adaptação desse conceito é proposta pelo *European Telecommunications Standards Institute* (ETSI)[1], originalmente em 2014 [57], na qual a computação de borda surge como uma plataforma para provisionamento de recursos de nuvem e tecnologia da informação dentro da *Radio Access Network* (RAN) nas proximidades do usuários de dispositivos móveis [57]. Tal definição restringe a MEC, então conhecida como *Mobile Edge Computing*, às redes móveis, como as redes *4th Generation* (4G) e *5th Generation* (5G).

Os esforços de padronização da MEC seguem em pleno desenvolvimento [57–68]. Atualmente, a computação de borda possui uma definição mais abrangente [58], sendo um amplo conjunto de técnicas que direcionam os recursos computacionais e de armazenamento das nuvens remotas, sejam elas públicas ou privadas, para perto

[1] `https://www.etsi.org/media-library/white-papers`

da origem dos dados. Para o ETSI [61], a computação de borda é tida como uma evolução da computação em nuvem e o processamento das aplicações de rede passa a ocorrer em uma localização fisicamente mais próxima dos usuários que geram e consomem os dados dessas aplicações. Por conseguinte, a partir de 2016, a MEC passa a ser conhecida como *Multi-Access Edge Computing* [55]. Essa nova definição contempla diferentes tecnologias de rede de acesso, como Wi-Fi e Ethernet, além das redes móveis [63], o que permite que um escopo maior de aplicações e dispositivos conectados seja contemplado pelos benefícios da MEC.

2.1.1 Vantagens da computação de borda

Os benefícios trazidos pela computação de borda podem ser aproveitados tanto pelos usuários finais, quanto pelos provedores de serviços e de infraestruturas de telecomunicações [55]. Algumas das vantagens da computação de borda recorrentes na literatura [30, 36, 69, 70], com relação ao usuário final, são:

- Prover baixa latência e *jitter* reduzido na execução de aplicações, o que melhora a QoE de aplicações sensíveis a latência.
- Permitir o *offloading* computacional para diminuir o fardo do processamento de aplicações computacionalmente intensivas nos dispositivos de usuário, como *smartphones*.
- Economizar bateria, reduzir temperaturas e melhorar a responsividade do *hardware* do usuário ao realizar o *offloading*.
- Melhorar o acesso a recursos computacionais para dispositivos com conexão intermitente à nuvem, como em carros e *drones*.
- Executar tarefas *context-aware*, isto é, tarefas que considerem informações de geolocalização, clima ou atividade realizada pelo usuário, para dispositivos com restrições de recurso quando a nuvem não estiver disponível.

Da perspectiva dos provedores de serviços [30, 71, 72], como Google Cloud e Alibaba Cloud, e de infraestruturas de telecomunicações, como AT&T e Verizon, a computação de borda oferece as seguintes vantagens:

- Melhorar a *Quality of Service* (QoS) prestada aos usuários finais, o que pode trazer ganhos financeiros para o provedor ao mantê-los satisfeitos.
- Distribuir o processamento, o armazenamento, o uso de largura de banda e o controle pelos servidores na borda da rede para realizar balanceamento de carga.

- Utilizar servidores na borda da rede como *backup* para a nuvem, caso o usuário não consiga acessá-la, o que melhora a resiliência da rede e lida com pontos únicos de falha.
- Tornar mais escalável o provisionamento dos serviços de rede e de telecomunicações, resolvendo localmente as demandas dos usuários associados a servidores de borda.
- Diminuir o atraso nas filas de requisição de usuários nos servidores centralizados da nuvem.
- Reduzir a exaustão dos recursos físicos e virtuais dos servidores de nuvem, melhorando o tempo de vida útil dos equipamentos da rede.
- Tornar ubíquo o acesso à infraestrutura de rede e melhorar a disponibilidade do serviço provido.
- Desafogar os enlaces centrais da rede, melhorando a utilização de largura de banda e possíveis congestionamentos de rede.
- Viabilizar aplicações de serviços críticos, como detecção e alerta de desastres naturais e acidentes, em locais inóspitos e com baixa latência.

2.1.2 Desvantagens da computação de borda

Apesar dos benefícios trazidos pela MEC para usuários finais e provedores, melhorando o provisionamento de serviços como AR, VR, IoT e *streaming* de vídeo [35], existem algumas possíveis desvantagens associadas à adoção desse modelo [71, 73], tais como:

- Ocasionar maior complexidade no gerenciamento da rede devido ao maior número de servidores distribuídos pela rede.
- Ter mais gastos financeiros com a criação e a manutenção da infraestrutura de borda além da infraestrutura de nuvem.
- Dispor de conjuntos de dados pequenos ou informações incompletas, associados apenas aos usuários próximos, o que pode impactar na precisão dos resultados de aplicações de IA, por exemplo.
- Aumentar os riscos à segurança da rede dada a característica distribuída da MEC e por não estar em um ambiente presumidamente seguro como o de um *data center* de nuvem, o que pode comprometer a confidencialidade e a integridade dos dados dos usuários.

- Haver escassez de recursos físicos e virtuais, uma vez que os servidores de borda não possuem a mesma capacidade dos servidores de nuvem, o que pode fazer com que seja necessário executar aplicações no próprio dispositivo de usuário.

É importante ressaltar que as desvantagens da MEC descritas acima podem ser interpretadas como desafios ou problemas temporários, uma vez que a implementação da infraestrutura de MEC ainda está em estágios iniciais [24, 55]. Isso justifica, por exemplo, os problemas associados aos custos de aquisição dos servidores de borda e das discussões ainda iniciais sobre aspectos de segurança em MEC [66].

Com relação à escassez de recursos, convém lembrar que o *offloading* computacional de tarefas se destaca como uma solução promissora [35, 41], pois ele busca utilizar de forma eficiente tanto os recursos do dispositivo do usuário quanto do servidor de borda, conforme visto no Capítulo 1, uma vez que o problema PODAG atacado nesta tese é um problema de *offloading* computacional.

2.1.3 Computação em Névoa e *Cloudlets*

Para finalizar as discussões da fundamentação teórica sobre computação de borda, são trazidas aqui algumas discussões semânticas sobre paradigmas de rede semelhantes à MEC para fins de esclarecimento das definições. Ao longo dos anos [35], três nomenclaturas têm sido utilizadas para expressar ideias semelhantes, frequentemente de forma intercambiável e adotados na indústria [33]: computação de borda (ou MEC), *fog computing* e *cloudlets*.

Ainda há muita discussão e pouco consenso com relação às definições de borda, *fog computing* (ou computação em névoa) e *cloudlets*. Chiang e Zhang [36] afirmam que o conceito de névoa é tido como mais abrangente, mas não há uma diferenciação clara entre os termos, embora os autores concordem que tanto a computação em névoa quanto a de borda surgem da ineficiência das nuvens para lidar com as aplicações emergentes em IoT e redes 5G. Santos *et al.* [74] consideram a definição de *cloudlet* como uma infraestrutura de processamento e armazenamento de dados na mesma rede local do dispositivo móvel com recursos limitados. No entanto, essa visão também pode contemplar a MEC ou a computação em névoa.

Basicamente, embora os objetivos de cada paradigma sejam semelhantes, isto é, trazer recursos de processamento, memória, armazenamento e rede para mais perto do usuário final, que é quem executa as aplicações de rede, há uma diferença temporal entre as definições desses conceitos e de quais organizações estão envolvidas nesse processo.

O conceito de *cloudlet* é proposto em 2009 [75], antes das iniciativas da computação de borda e em névoa. Diferentemente dessas, as *cloudlets* são desenvolvidas

no meio acadêmico pela *Carnegie Mellon University* (CMU) como uma camada intermediária entre dispositivos móveis e nuvem. O termo *cloudlet* é estabelecido no contexto de nuvens descentralizadas, aplicações *wearable* e de assistência cognitiva. Uma *cloudlet* é um *small-box data center* localizado a um salto de distância do dispositivo móvel [56].

A computação em névoa [76, 77] surge na indústria em 2012 [78] com o objetivo de trazer maior capacidade computacional e de comunicação de forma ubíqua [56] para diferentes dispositivos de rede com recursos limitados, porém voltada principalmente para objetos inteligentes e IoT. A computação em névoa é uma definição proposta pela Cisco, que integra o *Open Fog Consortium* (OFC), formado em 2015, junto a outras empresas e também universidades. A computação em névoa prevê uma integração maior com a nuvem, o que não é necessariamente o caso da MEC, como uma arquitetura de três camadas (dispositivo de usuário, névoa e nuvem), e que a névoa deve colaborar com o núcleo da rede [36]. Outro ponto particular da computação em névoa é que ela não precisa estar necessariamente a um salto do dispositivo móvel, ela pode ser construída como uma hierarquia de múltiplas camadas, das mais distantes às mais próximas do núcleo da rede.

Conforme visto anteriormente, o surgimento da MEC ocorre em 2013 e os esforços de padronização iniciais do ETSI em 2014, paralelamente à computação em névoa. Já em 2015, além do surgimento do OFC, surge também a parceria *Open Edge Computing* (OEC) criada pela CMU, responsável pela introdução do termo *cloudlets*, para impulsionar o crescimento da MEC [18]. Atualmente, essa parceria inclui diversas operadoras de redes móveis, como a Nokia, e também universidades. A OEC também atua no desenvolvimento das *cloudlets* para impulsionar essa tecnologia.

Muitos trabalhos disponíveis na literatura [79–83] fornecem uma pesquisa detalhada acerca das diferenças entre MEC, computação em névoa e *cloudlets* e maiores informações podem ser encontradas nessas referências. De forma geral, as semelhanças entre os três paradigmas incluem a implementação distribuída e o fato de proporem uma camada intermediária entre o dispositivo final e a nuvem, que está fisicamente mais próxima do usuário final (ligada a APs, roteadores, servidores na rede local ou estações base). Já as diferenças podem incluir a distância em relação ao usuário final, a modelagem, as aplicações e, conforme visto anteriormente, a origem de cada paradigma. A Tabela 2.1 resume uma comparação mais detalhada entre os três paradigmas.

Aqui é importante relembrar que um dos objetivos desta tese é propor uma solução de *offloading* para o problema PODAG para quaisquer tipos de aplicação, independente da estrutura subjacente dos DAGs que as compõem, e tais aplicações emergentes são contempladas pela MEC. Isso é tratado na sequência, na Seção 2.2. Além disso, devido ao caráter generalista das definições de MEC, à distância de um

Tabela 2.1: Semelhanças e diferenças entre MEC, *fog computing* e *cloudlets*.

Características	**MEC**	***Fog computing***	***Cloudlets***
Distância para o usuário final	Um salto	Até poucos saltos	Um salto
Inclusão da nuvem no modelo	Não necessariamente	Sim	Sim
Número de camadas na arquitetura	Duas ou três	Três ou mais (a névoa pode ter múltiplas camadas até a nuvem)	Três
Aplicações	Uso geral, redes 4G e 5G	Essencialmente IoT	Aplicações móveis e assistência cognitiva
Organização responsável pela proposta	Nokia e IBM	Cisco	CMU
Esforços de padronização e consórcios envolvidos	ETSI MEC	OFC	OEC
Descentralização de recursos	Sim	Sim	Sim
Virtualização de recursos	Máquinas virtuais e contêineres	Máquinas virtuais e contêineres	Máquinas virtuais
Acesso à rede	Redes móveis e Wi-Fi	Bluetooth, ZigBee, LoRa e Wi-Fi	Wi-Fi
Suporte a mobilidade de usuários	Sim	Sim	Sim
Dispositivo no qual ocorre o processamento	Servidores de borda conectados à estação base ou aos APs	Dispositivos ao longo da malha de roteamento (APs, comutadores, sensores etc.)	*Cloudlets* (ou *data centers "in a box"*)
Suporte a aplicações emergentes e *context-aware*	Sim	Sim	Sim

salto entre o usuário final e o servidor de borda e à independência com relação à infraestrutura de nuvem, a MEC é escolhida para ser o arcabouço da modelagem do sistema proposto nesta tese, a ser discutida no Capítulo 3.

2.2 Aplicações de Rede Emergentes

Muitas aplicações de rede têm surgido ao longo da década [15, 30, 84, 85], tais como: *streaming* de vídeo em alta resolução, Internet tátil[2], telemedicina, automação industrial[3] [86, 87], cidades inteligentes, dispositivos *wearable* (como aparelhos de *health care* e assistência cognitiva para auxiliar em tarefas do cotidiano [75]), jogos

[2]Aplicações que captam a movimentação de mãos humanas por meio de luvas ou sensores e replicam a sensação do tato em um robô, por exemplo, para realizar uma cirurgia remota.

[3]Por exemplo, aplicações de controle em tempo real, sensoriamento, monitoramento e movimentação de robôs.

mobile e *online*, IA geradora de conteúdo, experiências imersivas em 3D (como no metaverso [14]), redes de *drones*, transportes inteligentes[4], VR e AR [88, 89]. Para tais aplicações, a computação de borda é uma necessidade [58], uma vez que o modelo fortemente centralizado da computação em nuvem pode penalizar os usuários de aplicações sensíveis a latência [90].

Essas aplicações emergentes implementam serviços cujos requisitos variam amplamente para alcançar a satisfação do usuário [15]. Algumas delas apresentam níveis críticos de latência e confiabilidade a serem atendidos, outras exigem alta largura de banda por transmitirem grandes quantidades de dados, e ainda há as que também dependam de um grande número de conexões e dispositivos conectados simultaneamente para funcionar [85]. A identificação desses requisitos leva à formação de categorias de serviços, chamadas de casos de uso, da próxima geração de redes [15]. No âmbito das redes 5G, mas sem se limitar à mesma, frequentemente são citados [30, 84, 86] três casos de uso:

- *enhanced Mobile BroadBand* (eMBB): o primeiro caso de uso a ser implementado ainda na transição das redes 4G para as redes 5G [91]. Essa categoria inclui os serviços gerais de banda larga, incluindo o provisionamento de serviços a grandes públicos e em eventos com alta densidade de usuários, para fornecer uma taxa de transmissão de dados elevada e suporte à mobilidade dos usuários. Algumas das aplicações que pertencem a essa categoria são: *streaming* de vídeo de alta definição, jogos *mobile* e *online* e IA geradora de conteúdo.

- *ultra Reliable Low Latency Communication* (uRLLC): esse caso de uso frequentemente é associado a respostas em tempo real, missões críticas que exijam responsividade imediata [89] e não possam falhar. Aplicações de uRLLC são consideradas as mais inovadoras das aplicações emergentes e as que possuem requisitos de latência, confiabilidade e recursos de *hardware* mais exigentes de se atender. Algumas das aplicações que pertencem a essa categoria são: aplicações de tempo real (como *health care* e automação industrial), Internet tátil e telemedicina.

- *massive Machine Type Communication* (mMTC): fortemente ligado à IoT [84], esse caso de uso envolve as aplicações baseadas em um número massivo de dispositivos conectados e se comunicando, enviando essencialmente dados de controle e sensoriamento de fenômenos do ambiente em que se encontram (como na natureza, em fábricas ou em áreas urbanas). Um diferencial desse caso de uso é que, dado o grande número de dispositivos, que pode chegar a

[4]Por exemplo, aplicações de controle de semáforo, difusão de mensagens sobre o trânsito e pilotagem de *drones* e veículos autônomos.

milhões [86], é desejável que os mesmos tenham baixo custo, alta eficiência espectral e baixo consumo de energia. Algumas das aplicações que pertencem a essa categoria são: comunicação de dispositivos *wearable*, redes de *drones* e sistemas de transporte inteligente.

Tabela 2.2: Aplicações de rede emergentes e suas características.

Aplicação	**Casos de uso**	**Latência E2E (em ms)**	**Confiabilidade**	**Taxa de dados (em Mbps)**
Streaming de vídeo 4K e 8K	eMBB	10 a 12	99%	Até 300
Internet tátil	uRLLC	0,5 a 10	99,999% a 99,9999999%	Até 1000
Telemedicina e telecirurgia	uRLLC	1 a 10	99,9999999%	100
Automação industrial	uRLLC e mMTC	1 a 100	99,999% a 99,9999999%	1 a 100
Cidades inteligentes	eMBB e mMTC	10 a 200, podendo chegar a 5000	99%	Até 300
Wearables (*health care*)	mMTC	Até 100	99,99%	2 a 5
Jogos *mobile* e *online*	eMBB	5 a 10	99,99%	Até 1000
IA geradora de conteúdo[5]	Possivelmente eMBB e uRLLC [92, 93]	Baixa, necessária para a responsividade ao usuário [94]	Alta, mais voltada para a precisão dos resultados [94]	Até 720 Mbps para treinamento e inferência [95]
Imersão 3D, AR e VR	eMBB, uRLLC e mMTC	1 a 10	99% a 99,999%	50 a 1000
Redes de *drones*	uRLLC e mMTC	2 a 50, podendo chegar a 150	99,9% a 99,999%	20 a 40
Transportes inteligentes	uRLLC e mMTC	1 a 100	99,9% a 99,99999%	10 a 700

A Tabela 2.2 lista algumas das aplicações emergentes que podem se beneficiar da computação de borda [96], bem como os casos de uso e seus requisitos específicos [15, 84–87, 89, 97–100]. Os requisitos de QoS para cada caso de uso são definidos de forma abrangente pela *International Telecommunication Union* (ITU)[6], mas podem variar conforme a aplicação. Por exemplo, taxas de dados de até 1 Gbps são

[5]Por serem uma aplicação muito recente e disruptiva, os requisitos necessários às IAs geradoras de conteúdo são previstos, na prática, para serem atendidos plenamente apenas pela *Sixth Generation* (6G) [94, 95].

[6]`https://www.itu.int/`

necessárias para aplicações eMBB; níveis de confiabilidade de, no mínimo, 99%, latência *End-to-End* (E2E) máxima de 1 ms e taxas de erro de pacote variando de 10^{-5} a 10^{-9} são necessários para aplicações uRLLC; e tanto uma alta eficiência espectral quanto conexões estáveis e ininterruptas de um número massivo de dispositivos são parâmetros a serem considerados para aplicações mMTC [15, 86, 101].

Nesta tese, a solução proposta no Capítulo 4 se endereça a qualquer tipo de aplicação de rede que tradicionalmente possa ser submetida para execução em uma nuvem (e, consequentemente, para a borda), sem se prender a nenhum caso de uso específico. Isso é uma questão levantada por Salaht *et al.* [24], que afirmam que muitas formulações e soluções atuais para versões correlatas do problema PODAG são muito específicas para cada aplicação. Além disso, os dados das aplicações reais utilizados nesta tese se referem a aplicações submetidas para execução na Alibaba Cloud, disponibilizados pelo *Alibaba Cluster Trace Program*[7]. Tais aplicações envolvem, mas sem se limitar a, simulações computacionais de alto desempenho, aprendizado de máquina, codificação de vídeo, renderização de AR/VR, servidores de jogos *mobile* e servidores de jogos *online*.

2.3 Trabalhos Relacionados

O descarregamento da execução de tarefas para outros dispositivos, que possam dispor de mais recursos computacionais para realizar o processamento, é um tema presente em diferentes paradigmas de redes e sistemas de computação ao longo dos anos [3, 47, 102, 103, 103, 104]. Esses sistemas, que surgem desde as *grids* [26] e redes *Peer-to-Peer* (P2P) [28] até a MEC, se baseiam na computação distribuída [105], viabilizando aplicações de rede que dependam da capacidade computacional e de comunicação de muitos dispositivos, o que seria inviável com apenas um dispositivo, como no caso do *streaming* de vídeo [106] e das redes de *drones* [100].

Com relação à computação de borda, surgem desafios dentro do problema PODAG, como o particionamento eficiente das aplicações [17], a seleção do dispositivo que deve processar a tarefa em uma rede com dispositivos heterogêneos [30], a variação das condições de transmissão [33] e o compartilhamento dos recursos físicos do sistema [100]. Muitos *surveys* estão presentes na literatura e discorrem sobre as principais características do *offloading* computacional, seja em IoT, computação em nuvem ou em MEC [15, 20, 21, 30, 32, 35, 41, 56, 69, 107–109]. Paralelamente, vários outros trabalhos estão disponíveis na literatura, propondo soluções para diferentes etapas do complexo processo de *offloading* de tarefas em MEC [45, 74, 110–114].

Esta Seção apresenta os principais trabalhos relacionados ao tema desta tese. Primeiro, são discutidas em linhas gerais as vertentes de pesquisa sobre *offloading*

[7] `https://github.com/alibaba/clusterdata`

em computação de borda e como esta tese está posicionada na literatura. Em seguida, são apresentadas as principais abordagens encontradas na literatura que buscam resolver problemas semelhantes ao problema PODAG.

2.3.1 Visão Geral sobre *Offloading* em MEC

O processo de *offloading* da execução de uma aplicação deve considerar diversos fatores [108], como as características de cada aplicação, as condições de processamento da borda da rede, as condições de transmissão do canal tanto no transmissor quanto no receptor e o próprio particionamento e a alocação das tarefas que compõem a aplicação [32, 103].

Lin *et al.* [17] dividem o processo de *offloading* em três etapas: particionamento, alocação e execução da aplicação. O particionamento envolve a análise e subdivisão do código, que pode ser feita através de um programa ou manualmente pelo programador da aplicação, e a definição da granularidade das subdivisões, que pode ser em nível de aplicação inteira, tarefas e métodos. A alocação trata da descoberta e do gerenciamento de recursos dos dispositivos móveis e de borda, bem como a tomada de decisão de *offloading* e o escalonamento das tarefas a serem executadas. Já a execução lida com o uso de arquiteturas de processamento heterogêneas e o uso de soluções de execução orientadas a sistema, como máquinas virtuais e *containers*, para execução [32, 33, 56]. Além disso, também existem soluções orientadas a linguagem de programação, como as baseadas em Javascript[8] e WebAssembly[9].

Ainda segundo Lin *et al.* [17], não é considerado adequado em MEC o particionamento em nível de método, o que pode ver visto como uma granularidade mais fina do que o necessário, mas sim em nível de tarefa. Dito isso, o problema PODAG investigado nesta tese é um problema que envolve as duas etapas iniciais do *offloading*: particionamento (em nível de tarefa, com a decomposição das aplicações em DAGs, cujos vértices são as tarefas) e alocação, pois a heurística TALON proposta nesta tese realiza a tomada de decisão de *offloading* para a aplicação.

Em Mao *et al.* [30], os trabalhos sobre *offloading* em computação de borda são classificados de acordo com outras três categorias com relação à modelagem do problema: binário, parcial e estocástico. A modelagem de tarefas binária pressupõe que toda a aplicação deva ser executada ou localmente, no dispositivo móvel, ou remotamente, no servidor de borda. Essa simplificação é semelhante a considerar que a aplicação seja um bloco monolítico e indivisível. A modelagem parcial permite que algumas das tarefas que compõem a aplicação sejam executadas localmente e o restante na borda. A modelagem estocástica, ao contrário da modelagem deter-

[8] `https://stackoverflow.blog/2023/02/23/how-edge-functions-move-your-back-end-close-to-your-front-end/`

[9] `https://wasmedge.org/`

minística dos casos anteriores, parte do princípio que a chegada das tarefas ocorre aleatoriamente e as mesmas são inseridas em *buffers* para execução posterior. Esse último caso lida com incertezas, que também podem estar associadas às condições do canal de transmissão, e tem como objetivo o desempenho no longo prazo.

Mao *et al.* [30] ainda destacam a relevância da teoria de grafos para modelar problemas de *offloading* de tarefas, que podem ser transformados em problemas de otimização. No entanto, tais problemas frequentemente são classificados como NP-difíceis, sendo intratáveis computacionalmente, e alternativas que contornem esse problema são necessárias, como as heurísticas [39, 50]. Dessa forma, esta tese utiliza a teoria de grafos para modelar as aplicações como DAGs e elabora o problema PODAG considerando-o como um problema de *offloading* parcial e propõe a heurística TALON para resolvê-lo de forma eficiente.

Mach e Becvar [108] classificam os trabalhos disponíveis na literatura sobre *offloading* em MEC quanto à tomada de decisão de *offloading* e quanto à alocação de recursos. A tomada de decisão avalia se o *offloading* é possível e se é benéfico, seja para o dispositivo móvel seja para o operador da infraestrutura de rede, em termos de redução de atraso, consumo de energia e outros fatores. Assim como em Mao *et al.* [30], os trabalhos nessa categoria são classificados em *offloading* total ou parcial. Em seguida, a alocação de recursos computacionais e de comunicação define, por exemplo, qual dispositivo de borda deve executar a aplicação (em um cenário com múltiplos dispositivos), a política de escalonamento para executar aplicações críticas primeiro, cooperação entre os nós e balanceamento de carga.

Mach e Becvar [108] destacam com relação à modelagem das aplicações os seguintes aspectos: capacidade de *offloading* e dependência de execução entre tarefas. Uma aplicação pode não ser capaz de realizar o *offloading* de todas as tarefas que a compõem. Por exemplo, tarefas de *Input*/*Output*, ou inicialização e finalização, dependem da interação com o usuário final da aplicação. Isso faz com que frequentemente a primeira tarefa e a última tarefa de uma aplicação devam ocorrer no dispositivo de usuário [39, 44]. Já a dependência de execução entre tarefas classifica se há uma relação de dependência na ordem de execução das tarefas, ou se elas podem ser executadas em paralelo. Essa classificação pode ser representada facilmente através das arestas de um DAG. Ambas as considerações feitas por Mach e Becvar [108] são contempladas nesta tese.

De acordo com as classificações apresentadas acima, o resumo abaixo lista o posicionamento desta tese junto à literatura:

- A decisão de *offloading* é feita para cada tarefa no DAG da aplicação, conforme a modelagem de *offloading* parcial no problema PODAG. Isso é discutido no Capítulo 3.

- Uma solução heurística para o problema PODAG é elaborada no Capítulo 4.
- O particionamento das tarefas é extraído do *data set* do *Alibaba Cluster Trace Program*, do qual se obtém os DAGs de aplicações com granularidade fina, em nível de tarefa. Com relação às aplicações, o *offloading* é possível apenas para algumas tarefas, a primeira e a última precisam ser executadas localmente, pois representam o dispositivo de usuário. Além disso, as aplicações possuem tanto tarefas que dependam da execução de outras quanto tarefas que possam ser executadas em paralelo caso necessário. Todas essas características são tratadas no Capítulo 5.

2.3.2 Referências Principais

Dentre os trabalhos que compõem o estado da arte sobre o *offloading* de aplicações, três categorias se destacam: os trabalhos que consideram aplicações monolíticas (portanto, indivisíveis), os trabalhos que consideram as aplicações como um aglomerado de *bytes* e os que consideram a subdivisão das aplicações em partes menores bem definidas, como tarefas [41].

Liao *et al.* [115] formalizam um problema de *offloading*, enfileiramento e execução de aplicações em redes celulares densas que contenham dispositivos de borda, utilizando algoritmos genéticos em sua solução para reduzir conjuntamente o tempo de conclusão, ou *makespan*, e o consumo de energia. Nas estações base, o processamento das aplicações ocorre apenas de forma sequencial.

Kuang *et al.* [116] propõem um problema de *offloading* computacional em MEC com o objetivo de reduzir o *makespan* das aplicações enquanto também se reduz o consumo de energia, através de técnicas de otimização convexa e decomposição lagrangiana.

Wang *et al.* [45] também recorrem a técnicas de otimização para solucionar um problema de *offloading* com o objetivo de reduzir *makespan* e consumo de energia. Isso é feito através da variação do nível de tensão e da frequência utilizados no sistema para economizar energia nos dispositivos móveis, que são limitados em bateria.

Liao *et al.* [115] e Kuang *et al.* [116] não mencionam a estrutura interna das aplicações e as consideram como um bloco monolítico. Isso leva ao *offloading* das aplicações por completo. Se uma aplicação for computacionalmente intensiva ou possuir uma estrutura de dados muito grande, todo o fardo fica a cargo ou do dispositivo de usuário ou do servidor de borda. Em um cenário com múltiplos usuários, caso todos optem por realizar *offloading*, é possível chegar a um ponto em que os dispositivos de usuário fiquem ociosos e o servidor de borda sobrecarregado. No caso de Wang *et al.* [45], o particionamento da aplicação chega a ser considerado,

mas isso é feito de forma não realista. Uma aplicação não pode ser vista como um aglomerado de *bytes*, sobre o quais é feito algum processamento, que possa ser particionado em qualquer ponto [30, 35]. Isso pode levar a inconsistências na execução, fragmentação errônea do código da aplicação e, consequentemente, à perda de informações de contexto, variáveis globais etc.

Os trabalhos relacionados mais próximos desta tese lidam com versões correlatas do problema PODAG, considerando a subdivisão da aplicação em tarefas e a transmissão de dados, representada pelas dependências, entre as mesmas.

Jia *et al.* [39] propõem uma heurística para a tomada de decisão de *offloading* no contexto de *Mobile Cloud Computing* (MCC), fazendo um balanceamento de carga entre as tarefas executadas localmente e na MCC. A MCC [107] é uma concepção de computação em nuvem que possui em sua infraestrutura usuários de redes móveis, o que permite que ela seja tida como precursora da MEC [17]. Jia *et al.* [39] elaboram uma solução para tarefas sequenciais e outra para tarefas paralelas com o objetivo de reduzir o makespan dos DAGs de aplicação. Uma solução genérica também é proposta, porém ela consiste em aglomerar tarefas de modo a transformá-las em tarefas sequenciais ou paralelas, reduzindo DAGs genéricos a árvores. Embora isso possa ser útil para determinadas aplicações, não é uma proposta suficientemente genérica, pois, dependendo da disposição das arestas em um grafo, pode não ser possível a aglomeração de tarefas.

Shu *et al.* [50] também recorrem à teoria de grafos para modelar as aplicações em DAGs e propõem uma heurística para reduzir o tempo de execução das aplicações em *cloudlets*. A heurística se baseia na busca pelos caminhos que envolvam todos os vértices do grafo, a partir dos vértices finais e retornando até o primeiro. Com isso, são obtidos vários caminhos sequenciais. Outro ponto a considerar em Shu *et al.* [50] é que as aplicações consideradas não são suficientemente genéricas. A topologia em malha é um conjunto de caminhos paralelos e as topologias em árvore e genérica são estruturalmente semelhantes. O compartilhamento de recursos entre os *workers* da *cloudlet* e os dispositivos móveis também não é mencionado.

An *et al.* [53] estudam um cenário de MEC voltado para DAGs de aplicações inteligentes, que unem IA e IoT, e aplicam o particionamento em tarefas para realizar o *offloading* parcial, cujos objetivos são reduzir o *makespan* e o consumo de energia dos dispositivos de IoT. As condições do canal de comunicação, como desvanecimento, são levadas em consideração.

Zhang *et al.* [3] tratam de aplicações de jogos *mobile* e visam minimizar o consumo de energia total dos dispositivos móveis, considerando o tempo de resposta como uma restrição do problema de otimização. Além da comunicação entre dispositivo móvel e servidor do jogo na borda de rede, também é possível a troca de dados *Device-to-Device* (D2D) entre os usuários que estejam jogando o mesmo jogo.

No entanto, essa interação D2D não é aprofundada. Dada a alta complexidade do problema, um algoritmo ganancioso de alocação de tarefas é desenvolvido. Zhang *et al.* [3] consideram o jogos MICShooter e "Minecraft clone" (inspirado no original) que são modeladas como grafos. No entanto, a estrutura do grafos varia conforme as ações dos jogadores (*gameplay*) e por quanto tempo o jogo roda. Além disso, ações repetitivas e a execução contínua do jogo levam à recursão de tarefas.

Essas aplicações estruturadas como grafos cíclicos pertencem a uma outra categoria de aplicações, como aplicações interativas em ambientes de AR/VR ou de execução contínua por horas em jogos *mobile* e *online*. Para esses casos, o problema de *offloading* apresenta desafios específicos. Por exemplo, a estrutura do grafo não é conhecida previamente, pois não há como saber quantas vezes um *loop* é realizado ou quais ações o usuário pode realizar, representando uma incerteza sobre qual a próxima tarefa do grafo a ser executada. Dadas as diferenças presentes nesse cenário, essa categoria de aplicações estruturadas como grafos cíclicos não é contemplada pelo problema PODAG descrito nesta tese.

Duan e Wu [48] elaboram um problema de escalonamento com *offloading* para MCC voltado exclusivamente para aplicações inteligentes de *Deep Neural Networks* (DNNs), utilizando os dispositivos móveis e a nuvem para diminuir os tempos de processamento e transmissão com o objetivo de reduzir o tempo de inferência da DNN. Duan e Wu [48] criam uma solução algorítmica para resolver o problema tanto de forma ótima quanto heurística, considerando que a maioria das aplicações de DNN são estruturadas como sequências ou árvores, embora também elaborem uma heurística para DAGs de aplicações de DNN genéricos.

Zhang *et al.* [47] formulam um problema de otimização para realizar o escalonamento da execução de aplicações de MCC. Nesse cenário, o dispositivo móvel e a nuvem colaboram entre si com o objetivo de minimizar o consumo de energia do dispositivo móvel e atender a restrições de atraso máximo permitido. São utilizados a relaxação lagrangiana e o processo de decisão de Markov para obtenção das decisões de *offloading* e obter uma política de escalonamento de tarefas.

Han *et al.* [117] lidam com o problema de particionamento e *offloading* de DAGs de aplicações de IA estruturadas como árvores em MEC. Isso é feito levando em conta a localização geográfica dos dispositivos e a interferência gerada pelo *offloading* de tarefas adicionais sobre a execução das tarefas que já foram *offloaded* anteriormente e estejam em execução, uma vez que a capacidade de processamento e a de transmissão são compartilhadas entre os serviços (aplicações). Para resolver o problema, uma heurística baseada no algoritmo de colônia de formigas é desenvolvida.

Yang *et al.* [49] tratam do escalonamento e do *offloading* de tarefas em MEC através de um problema de otimização de pior caso, no qual o atraso máximo é analisado e minimizado. Como o problema é NP-difícil, são propostos uma heurística

para realizar o escalonamento e o *offloading* e um algoritmo de identificação de prioridade de tarefas para resolver o problema. A identificação de prioridade é feita com base na proximidade entre *makespan* e o prazo de conclusão da tarefa (*deadline*), o que leva à marcação de algumas tarefas como altamente prioritárias para que sejam prontamente executadas.

À parte de Zhang *et al.* [3], que lida com aplicações de execução contínua, modeladas como grafos cíclicos e não como DAGs, esses trabalhos relacionados têm em comum se baseiam em DAGs de aplicação muito simples ou específicos de um tipo de aplicação apenas. Há DAGs compostos apenas por tarefas sequenciais e paralelas [39, 47, 48, 50], ou que sejam estruturados como árvores cujas tarefas sejam homogêneas [117], ou que apresentem uma quantidade muito pequena de tarefas [53], ou ainda que sejam gerados aleatoriamente sem representar uma aplicação real (DAGs sintéticos) [49].

Por outro lado, o trabalho de Guo *et al.* [46], que estende um trabalho anterior [42], é a referência mais próxima do trabalho desta tese. Em Guo *et al.* [46], é proposta uma solução para o problema de *offloading* parcial para minimizar o tempo de resposta das aplicações de MEC (ou MCC [42]). Um problema de programação inteira mista é formulado e três algoritmos são propostos para eleger a decisão de *offloading* para os seguintes casos: DAGs de aplicação sequenciais, DAGs de aplicação genéricos, e múltiplos DAGs em um cenário multiusuário. Para resolver o problema de forma eficiente, uma heurística realiza o *offloading* das tarefas levando em consideração o congestionamento no canal de comunicação, evitando que mais de uma transmissão ocorra no canal por vez, agendando-as de maneira *First In, First Out* (FIFO), ou seja, não há paralelismo de transmissão. Outra restrição imposta faz com que as tarefas só possam ser executadas se houver alguma CPU física totalmente livre, evitando *multithreading*.

Assim como esta tese, Guo *et al.* [46] atacam o problema PODAG e consideram DAGs de aplicações reais, recursos computacionais limitados pelo número de CPUs e capacidade limitada no canal de comunicação. No entanto, Guo *et al.* [46] consideram a limitação de CPUs apenas para o servidor de borda, assumindo que o dispositivo móvel de usuário sempre tem CPUs livres o suficiente para executar qualquer DAG de aplicação.

Outra particularidade é o fato de Guo *et al.* [46] recorrerem muito ao processamento feito integralmente no dispositivo móvel, ou seja, não ocorre *offloading*. Sempre que o *makespan* calculado para uma decisão de *offloading* é maior que o *makespan* alcançado ao se executar a aplicação totalmente no dispositivo móvel, a solução de Guo *et al.* [46] desiste de realizar o *offloading* para aquela aplicação e a destina à execução local por completo. Tal pressuposto é uma limitação desse algoritmo, pois a redução do *makespan* não é sempre monotônica. Por exemplo, realizar

Tabela 2.3: Características analisadas nas principais referências em comparação a esta tese.

Referência	Paradigma	Tipo de solução	Modelagem da aplicação	Tipo de grafo	Granularidade	Tipo de dados	*Offloading*	Recursos compartilhados
Liao *et al.* [115] e Kuang *et al.* [116]	MEC	Ótimo e heurística	Monolítica	Não há	Aplicação	Sintético	Total	Não
Wang *et al.* [45]	MEC	Ótimo e heurística	Aglomerado de *bytes*	Não há	Nível de *bit*	Sintético	Parcial	Não
Jia *et al.* [39]	MCC	Ótimo e heurística	DAG	Sequências e árvores	Tarefas	Sintético	Parcial	Sim
Shu *et al.* [50]	*Cloudlets*	Ótimo e heurística	DAG	Sequências e árvores	Tarefas	Sintético	Parcial	Não
An *et al.* [53]	MEC	Ótimo	DAG	Sequências	Tarefas	Sintético	Parcial	Não
Zhang *et al.* [3]	MEC e *fog computing*	Ótimo e heurística	Grafos cíclicos	Genéricos	Métodos	Sintético e realista	Parcial	Não
Duan e Wu [48]	MCC	Ótimo e heurística	DAG	Principalmente sequências e árvores	Tarefas	Realista	Parcial	Não
Zhang *et al.* [47]	MCC	Ótimo e heurística	DAG	Sequências	Métodos (ou módulos)	Sintético	Parcial	Não
Han *et al.* [117]	MEC	Heurística	DAG	Árvores	Tarefas	Realista	Parcial	Sim
Yang *et al.* [49]	MEC	Heurística	DAG	Genéricos	Tarefas	Sintético	Parcial	Não
Guo *et al.* [42, 46]	MEC e MCC	Heurística	DAG	Genéricos	Tarefas	Realista	Parcial	Sim
Esta tese	**MEC**	**Ótimo e heurística**	**DAG**	**Genéricos**	**Tarefas**	**Realista**	**Parcial**	**Sim**

o *offloading* de uma única tarefa para a borda implica, no mínimo, em um atraso de transmissão referente ao *uplink* e outro referente ao *downlink*. Dependendo da quantidade de dados trocados nessas duas transmissões, essa decisão de *offloading* pode resultar em um *makespan* maior do que o da execução local, mas isso não significa que outras decisões de *offloading* necessariamente resultem em um *makespan* maior.

A Tabela 2.3 resume as características das principais referências e posiciona esta tese em relação às mesmas. Em resumo, esta tese resolve o problema PODAG, considerando as trocas de dados (dependências) entre tarefas, ambas as limitações de processamento e de transmissão, representadas pelo número de CPUs no dispositivo de usuário e no servidor de borda e pela largura de banda no canal de comunicação compartilhado, respectivamente. Isso é feito através da heurística TALON, que busca decisões de *offloading* para reduzir o *makespan* de aplicações reais, cujas estruturas são mais complexas e variadas do que aquelas encontradas nas referências principais.

Capítulo 3

Definição do Problema

Este Capítulo apresenta o problema PODAG, que é composto por três elementos: o dispositivo móvel do usuário, o servidor de borda e as aplicações estruturadas como DAGs. A aplicação é particionada em tarefas e, para cada tarefa, é tomada uma decisão sobre em qual dispositivo o processamento acontece, seja localmente ou remotamente (*offloading* para a borda). Tanto a modelagem do sistema quanto a das aplicações são discutidas a seguir.

3.1 Modelagem do Sistema

A modelagem do sistema MEC é descrita pela Figura 3.1, que mostra o cenário com apenas um usuário. As aplicações são estruturadas como DAGs e executadas no dispositivo móvel do usuário final, como um *smartphone*. Essas aplicações podem ser de qualquer natureza [15], como *streaming* de vídeo, AR/VR, IA geradora de conteúdo, jogos *mobile*, telemedicina etc. Além disso, as aplicações podem ser particionadas em tarefas, sequenciais ou paralelas, que são partes individuais de código a serem executadas no dispositivo móvel ou *offloaded* para um servidor próximo na borda da rede. Aqui se considera que tanto o dispositivo móvel quanto o servidor de borda sejam completamente capazes de executar as aplicações, isto é, ambos possuem todas as bibliotecas necessárias, requisitos de *hardware* e *software* instalados e o código da aplicação.

A Figura 3.1 ainda mostra um exemplo de decisão de *offloading* para a aplicação, que possui oito tarefas. As tarefas 2, 3, 4 e 5 são marcadas para *offloading* para o servidor de borda, enquanto as tarefas 1, 6, 7 e 8 permanecem atribuídas ao dispositivo móvel para processamento local. A transmissão de dados utiliza um canal de comunicação sem fio confiável entre o dispositivo móvel e o servidor de borda. As transmissões são realizadas sempre que o processamento da tarefa é *offloaded* e podem ocorrer tanto do dispositivo móvel para o servidor de borda quanto no sentido oposto. Os dados transmitidos correspondem à estrutura de dados passada de uma

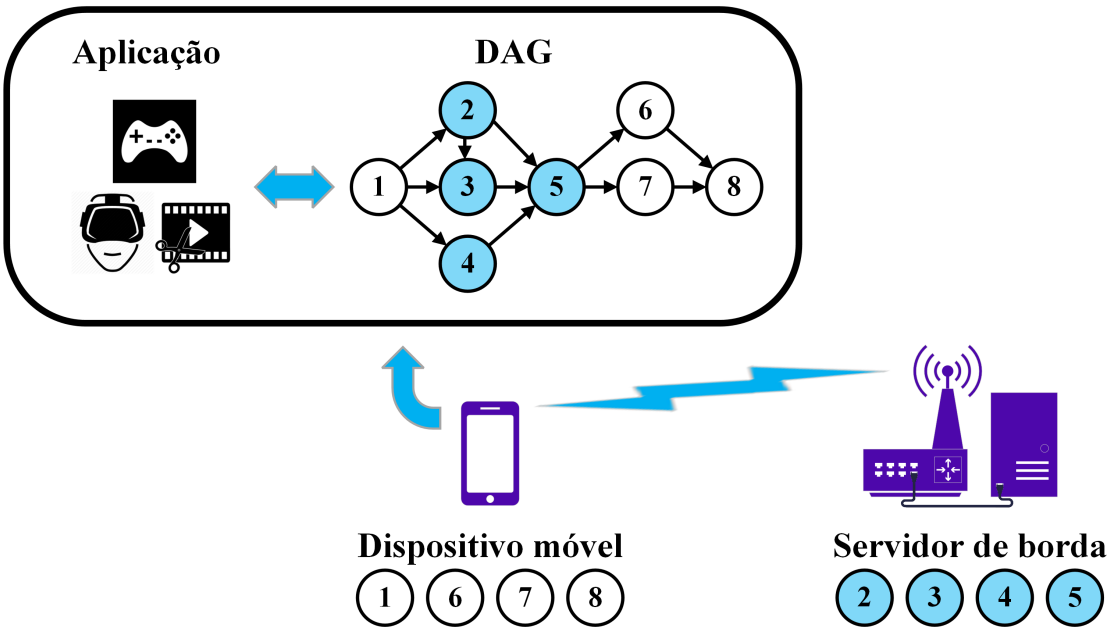


Figura 3.1: Modelagem do sistema MEC considerado no problema PODAG.

tarefa para a outra, são os parâmetros de entrada necessários, como uma variável, para executar a tarefa subsequente. Essas transmissões definem as dependências na ordem de execução das tarefas. Por exemplo, a tarefa 5 só pode ser executada quando todas as transmissões de dados originadas em suas tarefas predecessoras (tarefas 2, 3 e 4) forem concluídas.

3.2 Modelagem das Aplicações

As aplicações podem ser descritas como DAGs $G(V, E)$, nos quais o conjunto de vértices V é o conjunto de tarefas e o conjunto de arestas direcionadas E é o conjunto de transmissões de dados (dependências) entre tarefas. Uma tarefa $i \in V$ pode ser executada no dispositivo móvel ou no servidor de borda. Não há um limite estabelecido para o número de tarefas que podem ser executadas em paralelo, seja no dispositivo móvel ou no servidor de borda. Isso significa que o compartilhamento de CPU é permitido no problema PODAG, ou seja, é possível que haja mais tarefas alocadas a um dispositivo do que o número de CPUs disponíveis no mesmo. Além disso, as CPUs devem ser alocadas pelo escalonador de cada dispositivo, que é o responsável por definir qual tarefa é executada por vez, o que significa que uma CPU não necessariamente executa a mesma tarefa 100% do tempo.

A cada tarefa i, é associada uma decisão de *offloading* binária o_i, de forma que $o_i = 0$ representa a execução da tarefa no dispositivo móvel e $o_i = 1$ representa a execução no servidor de borda. O conjunto $\{o_i, \forall i \in V\}$ é o que se chama nesta tese de decisão de *offloading* D para uma dada aplicação. Como uma tarefa é considerada

a menor parte de uma aplicação, ou seja, indivisível, seu processamento não pode ser interrompido e depois migrado de um dispositivo para outro: uma tarefa deve ser executada inteiramente ou no dispositivo móvel ou no servidor de borda. Pela mesma razão, não é possível executar uma tarefa em duas ou mais CPUs ao mesmo tempo para acelerar o processamento. Em particular, para o caso de uma aplicação cujas tarefas realizem *multithreading*, cada *thread* deve ser considerada como uma tarefa individual (vértice) para a estruturação da aplicação como um DAG. Isso ocorre porque, conforme dito anteriormente, os vértices do DAG correspondem às menores partes do código da aplicação, que são as *threads* nesse caso.

Uma aresta $(i, j) \in E$ indica o envio de dados da tarefa i para a tarefa j, que são parte dos parâmetros de entrada necessários para a execução da tarefa j. Conforme dito anteriormente, uma tarefa só pode começar a ser executada quando todas as transmissões de todas as suas arestas de entrada forem recebidas. No caso de múltiplas arestas de saída, assim que uma tarefa i termina seu processamento, ela passa a transmitir em paralelo os dados correspondentes a cada uma dessas arestas, exigindo compartilhamento de largura de banda no canal de comunicação, caso o processamento das tarefas seguintes não seja feito no mesmo dispositivo onde ocorreu o processamento da tarefa i. Por outro lado, se existir uma aresta (i, j) para a qual ambas as tarefas i e j envolvidas sejam executadas no mesmo dispositivo (ou móvel ou borda), não há necessidade de utilizar o canal de comunicação e, portanto, o tempo de transmissão associado a (i, j) é considerado nulo.

Dois pressupostos são assumidos na modelagem das transmissões para fins de simplificação e sem perda de generalidade. Primeiro, as transmissões de dados não podem ser interrompidas e não há perda na transmissão. Tais características podem ser representadas simplesmente por um aumento no tempo de transmissão considerado em cada aresta, tanto pelo atraso adicional entre a pausa e a retomada de uma transmissão previamente interrompida quanto pela retransmissão no caso de perda. TALON é desenvolvida para resolver o problema PODAG para quaisquer casos, sejam as transmissões demoradas ou não, conforme a discussão dos resultados para uma ampla diversidade de cenários no Capítulo 6.

O segundo pressuposto assumido é que o grafo não dirigido subjacente é conexo. Isto é, existe pelo menos um caminho conectando cada par de vértices no DAG. Isso é razoável porque todas as tarefas do DAG decorrem da inicialização da aplicação feita pelo usuário final, representado pela tarefa inicial do DAG. De forma similar, o processamento de todas as tarefas do DAG culmina em um resultado final da execução da aplicação, que é enviado de volta para o usuário final, representado pela tarefa final do DAG. Por isso, tanto a tarefa inicial quanto a tarefa final rodam necessariamente no dispositivo móvel do usuário final [50]. Assim, nem a tarefa inicial nem a tarefa final podem ser marcadas para *offloading*. Como as decisões

de *offloading* para essas duas tarefas são definidas antecipadamente, as arestas de saída da tarefa inicial e as arestas de entrada da tarefa final são definidas aqui como âncoras. O conjunto de âncoras é um subconjunto de E.

Considerando uma aplicação com N tarefas elegíveis para *offloading*, as tarefas t_0 e t_{N+1} são as tarefas inicial e final, respectivamente. O *makespan* da aplicação é definido como o tempo total decorrido desde o início da execução de t_0 até o final da execução de t_{N+1}, quando todas as tarefas são processadas e não há mais transmissões de dados em andamento. Os tempos de processamento das tarefas e os tempos de transmissão dos dados são definidos a seguir.

3.2.1 Tempo de Processamento

O tempo de processamento é dado pelo tempo que uma tarefa leva para ser executada por completo em uma CPU. No entanto, esse tempo pode ser aumentado caso a CPU alocada for compartilhada com outras tarefas, conforme mostrado na Figura 3.2. No exemplo, a tarefa em azul leva 2 s para executar em uma CPU e a tarefa em verde leva 3 s em outra CPU. No entanto, caso haja apenas uma única CPU disponível para executar ambas as tarefas e não haja distinção em termos de prioridade entre as tarefas, a CPU é compartilhada por ambas as tarefas.

O compartilhamento de CPU reduz o tempo de ocupação que uma tarefa recebe para utilizar a CPU. Quanto menor for o tempo de ocupação, menos tempo a CPU gasta executando aquela tarefa específica e mais tempo a CPU passa executando outras tarefas. Ainda de acordo com a Figura 3.2, podendo cada tarefa utilizar apenas 50% da ocupação da CPU, o tempo de processamento necessário para a execução dobra, sendo 4 s para a tarefa em azul e 6 s para a tarefa verde. Supondo uma passagem de tempo igual a 4 s, a tarefa em azul é concluída e restam ainda 2 s para concluir a tarefa em verde com uma ocupação de 50% da CPU. Porém, como não há mais compartilhamento, a tarefa em verde pode ocupar 100% da CPU para si, o que faz com que o tempo restante para concluir seu processamento caia de 2 s para 1 s.

Não há um limite para o número de tarefas que podem estar associadas a uma mesma CPU, tanto para o dispositivo móvel quanto para o servidor de borda. Por exemplo, um *smartphone* com 8 CPUs disponíveis pode estar responsável por executar 10 tarefas em um dado momento. Como visto, cabe ao escalonador de tarefas executá-las uniformemente, tendo em vista a disponibilidade de 8 CPUs. As formulações matemáticas do tempo de processamento e da ocupação da CPU são descritas na sequência.

Uma nova tarefa inicia seu processamento imediatamente após receber todos os dados de todas as suas arestas de entrada, independentemente do número de

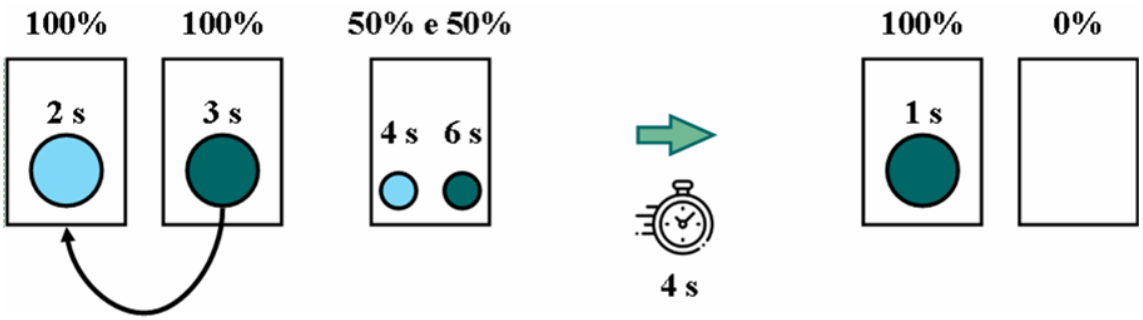


Figura 3.2: Impacto do paralelismo de tarefas no tempo de processamento.

tarefas já em execução. Quando uma tarefa i é alocada a uma CPU, na qual ela é a única tarefa em execução, ou seja, sem compartilhar essa CPU com outras tarefas concomitantes, o tempo de processamento de i é

$$t_i = \frac{n}{fc}, \tag{3.1}$$

onde n é o número de operações necessárias para executar a tarefa, f é a velocidade do *clock* (em ciclos/segundo) e c é a capacidade computacional da CPU (em operações/ciclo). Porém, devido ao compartilhamento de CPU, o tempo de processamento de uma tarefa aumenta à medida que outras tarefas começam a ser executadas na mesma CPU.

Havendo u CPUs em um dispositivo, até u tarefas podem ser executadas em paralelo sem qualquer compartilhamento e tem-se 100% de ocupação da CPU para cada tarefa. Assim que uma nova tarefa é iniciada nesse cenário, todas as tarefas em execução recebem apenas uma fração de CPU, estendendo seu tempo de processamento restante pelo fator $(u + 1)/u$, pois há mais tarefas do que CPUs disponíveis. Da mesma forma, quando o processamento de uma tarefa acaba, cada tarefa restante, ainda em execução, recebe de volta uma fração maior de ocupação da CPU, até o valor máximo de 100%, o que implica não haver mais compartilhamento de CPU, como no exemplo da Figura 3.2. Aqui, essa fração é chamada de fator de escala de CPU k_p, que é referente à ocupação da CPU por cada tarefa e é definido por

$$k_p = \max\left\{1, \frac{p}{u}\right\}, \tag{3.2}$$

no qual p é o número de tarefas executando em paralelo naquele instante. Como uma tarefa é indivisível, ela não pode ter sua execução acelerada com a alocação de mais de uma CPU para a mesma. Isso faz com que k_p nunca seja menor que 1. Quando $k_p = 1$, cada tarefa em execução recebe 100% da CPU para si, o que leva ao tempo de processamento dado pela Equação (3.1), que é o menor tempo possível para a conclusão da tarefa i.

Conforme as tarefas concomitantes a i terminam suas execuções e novas tarefas são iniciadas, o tempo de processamento restante associado a i precisa ser atualizado de acordo. O tempo de processamento restante para a conclusão de t_i, dado por Δt_i, é atualizado sempre que o fator de escala da CPU da Equação (3.2) varia e a tarefa i recebe uma parcela de utilização da CPU menor ou maior. O novo valor para o tempo de processamento restante $\Delta t'_i$ é dado por

$$\Delta t'_i = \frac{\Delta t_i k'_p}{k_p}, \tag{3.3}$$

onde k_p representa o fator de escala de CPU anterior e k'_p é o fator atualizado. Ter $k'_p > k_p$ indica o início da execução de uma nova tarefa naquele dispositivo, já $k'_p < k_p$ indica a conclusão de uma tarefa em execução até o presente momento.

O fator de escala de CPU é um ponto importante para o cálculo do tempo de processamento das tarefas e, consequentemente, para o *makespan* da aplicação como um todo e não pode ser desprezado no problema PODAG. Como o número de tarefas em execução em cada dispositivo varia, o tempo de processamento de uma tarefa no dispositivo móvel pode se tornar menor que no servidor de borda, mesmo que o *hardware* em si do servidor de borda tenha, por definição, mais recursos computacionais.

Sem o compartilhamento de CPU, o tempo de processamento mínimo de uma tarefa, que é dado pela Equação (3.1), é sempre menor no servidor de borda do que no dispositivo móvel. Porém, um número excessivo de tarefas executando paralelamente no servidor de borda pode elevar o k'_p do mesmo de modo que seja mais vantajosa a execução local. Assim, transferir tarefas para o servidor de borda sempre que possível pode resultar em um tempo de execução mais longo.

3.2.2 Tempo de Transmissão

A taxa de transmissão do canal de comunicação sem fio (em *bytes*/segundo) é denotada por r e considerada igual para transmissões de *uplink* e *downlink*. Cada aresta (i, j) representa os $d_{i,j}$ *bytes* que são passados da tarefa i como parâmetros de entrada da tarefa j. De forma análoga ao que é feito para o tempo de processamento, se não houver outra transmissão em andamento, o tempo de transmissão $T_{i,j}$ é dado por

$$T_{i,j} = \frac{d_{i,j}}{r}. \tag{3.4}$$

As transmissões de *uplink* ocorrem sempre que o processamento migra do dispositivo móvel para o servidor de borda, ou seja, a tarefa i é executada localmente e a j é executada remotamente. As transmissões de *downlink* ocorrem no sentido oposto,

com o processamento da aplicação retornando da borda de volta para o dispositivo móvel.

No entanto, como o canal de comunicação é compartilhado, podem ocorrer múltiplas transmissões simultameamente, como, por exemplo, no caso de uma tarefa com múltiplas arestas de saída, que inicia todas as suas transmissões ao mesmo tempo. A largura de banda do canal é dividida uniformemente pelo número de transmissões de dados acontecendo, dado por k_t, o que significa que todas as transmissões têm a mesma prioridade. A diferenciação dos serviços e a priorização com base em níveis de QoS pode ser facilmente implementada no modelo. No entanto, essa simplificação é feita porque nenhum pressuposto é assumido quanto à natureza das aplicações executadas pelo usuário. Além disso, é razoável que não haja priorização entre as transmissões na estrutura interna de um mesmo DAG, pois todas as transmissões de dados em um DAG fazem parte da execução da mesma aplicação, ou seja, possuem a mesma prioridade.

Sempre que uma nova transmissão começa ou uma transmissão anterior termina, o tempo restante para todos os $d_{i,j}$ *bytes* serem transmitidos, denotado por $\Delta T_{i,j}$, é atualizado. O valor atualizado do tempo de transmissão restante $\Delta T'_{i,j}$ é dado por

$$\Delta T'_{i,j} = \frac{\Delta T_{i,j} k'_t}{k_t}, \tag{3.5}$$

onde $k'_t = k_t - 1$ caso aconteça o encerramento de uma transmissão ou $k'_t = k_t + 1$ caso uma nova transmissão seja iniciada. Dessa forma, uma transmissão de dados pode utilizar 100% da largura de banda do canal somente quando ela é a única sendo realizada e essa porcentagem é reduzida conforme o canal é compartilhado. O tempo de transmissão mínimo dado pela Equação 3.4 só é alcançado quando, do início da transmissão do primeiro *byte* de $d_{i,j}$ até a recepção do último *byte*, nenhuma outra transmissão é iniciada.

Capítulo 4

Heurística TALON

Encontrar a solução ótima é reconhecidamente difícil [35, 117] para o problema PODAG, cuja formulação matemática leva a problemas de otimização não linear e não convexa presentes na literatura, os quais são provados como sendo NP-difíceis [30, 53]. As abordagens exatas conhecidas são, portanto, excessivamente demoradas [41], o que torna soluções quase ótimas, como a de uma heurística, mais adequadas para resolver o problema PODAG. Diante disso, esta tese propõe a heurística TALON para resolver o problema PODAG de forma eficiente e levando em conta as limitações de *hardware* do sistema, bem como o impacto do compartilhamento de CPU e de largura de banda do canal.

4.1 Visão Geral

Existem quatro considerações fundamentais para o raciocínio que conduz ao desenvolvimento de TALON, detalhadas a seguir.

Não monotonicidade do *makespan*: a redução do *makespan* não ocorre monotonicamente à medida que mais tarefas são *offloaded* para a borda. Algumas decisões de *offloading*, que definem em qual dispositivo cada tarefa do DAG é executada, podem gerar um *makespan* maior do que o *makespan* obtido ao se executar todas as tarefas localmente. Mesmo que o *offloading* de uma tarefa possa reduzir o seu tempo de processamento, essa redução pode ser suprimida pela adição dos tempos de transmissão de *uplink* e *downlink* referentes à tarefa *offloaded*. Por isso, TALON gera iterativamente um conjunto de decisões de *offloading* e, ao final, calcula o *makespan* de cada uma delas para selecionar aquela que resulta no menor *makespan*. TALON gera decisões de *offloading* em duas fases. A primeira emprega um critério ganancioso para escolher as tarefas que devem ser marcadas para *offloading* até que um determinado número de tarefas seja atingido. A partir disso, a segunda fase começa e inclui a geração de todas as decisões de *offloading* possíveis para as tarefas ainda não marcadas para *offloading* durante a primeira fase. Todas

essas decisões são consideradas, mesmo que sejam geradas algumas decisões ruins ao longo das iterações nas duas fases. TALON não se baseia apenas na última decisão gerada ou apenas na decisão que realiza o *offloading* do maior número de tarefas.

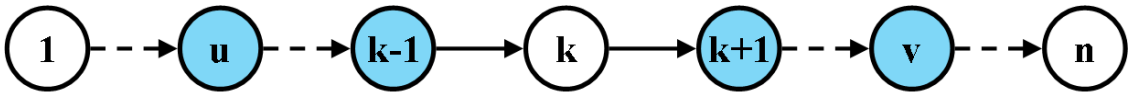
Heterogeneidade das tarefas: como as tarefas do DAG da aplicação são heterogêneas, algumas tarefas são mais preponderantes no cálculo do *makespan*, seja pelo fato de terem um tempo de processamento muito longo ou por serem tarefas muito centrais no DAG, com múltiplas arestas de entrada e/ou de saída, principalmente se tais arestas envolverem um grande número de *bytes* a serem transmitidos. A diferença de tempo (ganho) entre executar uma tarefa localmente e no servidor de borda varia de tarefa para tarefa em um DAG e, no geral, quanto maior for essa diferença, mais vantajoso se torna o *offloading* daquela tarefa para reduzir o *makespan* da aplicação.

***Offloading* aos pares de tarefas:** os tempos de transmissão tendem a ser mais representativos para o *makespan* do que os tempos de processamento, mas podem ser anulados. O tempo de processamento das tarefas só aumenta quando há mais tarefas do que CPUs disponíveis, implicando compartilhamento de CPU. Entretanto, apenas duas transmissões em paralelo já são suficientes para que cada uma tenha acesso a apenas metade da largura de banda disponível, assumindo o compartilhamento uniforme. Portanto, é desejável que transmissões mais longas sejam evitadas, buscando realizar o *offloading* de tarefas aos pares, já que isso anula o tempo de transmissão associado à aresta que liga aquele par. Em especial, isso se torna ainda mais relevante no caso de tarefas com muitas arestas de saída. Por exemplo, se uma tarefa é executada no dispositivo móvel e todas as suas sucessoras são executadas no servidor de borda, nenhuma transmissão é anulada e o compartilhamento da largura de banda por cada transmissão faz com que elas demorem ainda mais tempo para serem concluídas.

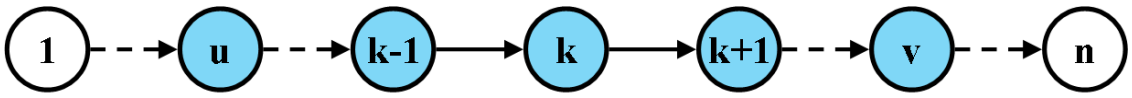
Política *one-climb*: a política *one-climb* não é uma verdade absoluta para a decisão de *offloading* ótima de um DAG genérico. Um contraexemplo é elaborado a seguir para demonstrar esse fato.

4.2 Política *one-climb* na literatura

A política *one-climb* é uma suposição recorrente na literatura sobre o *offloading* de tarefas [30, 39, 46, 47, 50, 51, 118]. Essa política se baseia nos conceitos de decisão de *offloading* contínua e descontínua para um DAG de aplicação e diz que: dada uma decisão de *offloading* descontínua $\mathcal{D}$, então existe pelo menos um caminho no DAG ao longo do qual pelo menos duas tarefas são marcadas para *offloading* por $\mathcal{D}$ e pelo menos uma tarefa entre elas não está. Isso significa que o processamento da aplicação migra para o dispositivo móvel (*downlink*) e retorna em seguida para o servidor de



(a) Decisão de *offloading* não contínua $\mathcal{D}$.



(b) Decisão de *offloading* contínua $\mathcal{D}'$.

Figura 4.1: Política *one-climb* aplicada a decisões de *offloading*.

borda (*uplink*) dentro desse caminho. Como a aplicação é iniciada e finalizada no dispositivo móvel, isso significa que ocorrem, no mínimo, duas transmissões de *uplink* e duas transmissões de *downlink* ao longo de toda a execução do DAG. Se tal decisão $\mathcal{D}$ existe, então também existe pelo menos uma decisão de *offloading* $\mathcal{D}'$ que resulte em um *makespan* menor que o da decisão $\mathcal{D}$, cujos caminhos tenham apenas sequências de *offloading* contínuas, de modo que apenas ocorram uma transmissão de *uplink* e uma de *downlink* em cada caminho do DAG, no máximo. A prova da política *one-climb* pode ser encontrada em Yang *et al.* [52] e em Zhang *et al.* [51], porém apenas para DAGs de aplicações sequenciais, como os da Figura 4.1.

A Figura 4.1 ilustra uma possível decisão de *offloading* $\mathcal{D}$ e sua respectiva versão contínua $\mathcal{D}'$. As tarefas em azul são executadas no servidor de borda, enquanto as tarefas em branco são executadas localmente no dispositivo móvel. Em $\mathcal{D}$, a sequência de *offloading* inicia em u, termina em $k-1$, recomeça em $k+1$ e termina de fato em v. Já em $\mathcal{D}'$, a sequência de *offloading* inicia em u e termina em v. É fácil notar que $\mathcal{D}'$ obedece à política *one-climb*, enquanto que, em $\mathcal{D}$, ocorrem duas transmissões de *uplink* e duas de *downlink*.

Embora a política *one-climb* seja válida para DAGs sequenciais, isso não acontece para estruturas mais complexas devido às limitações de CPU no servidor de borda e ao paralelismo de transmissão. É possível que um servidor de borda fique tão sobrecarregado com a execução concomitante das tarefas associadas a ele que o tempo de processamento de uma nova tarefa seja menor no dispositivo móvel. Ou seja, ao invés de haver uma diminuição (ganho) no tempo de processamento ao realizar o *offloading* da tarefa para o servidor de borda, o que ocorre é um aumento do tempo de processamento, sendo mais vantajosa a execução local. Além disso, forçar o *offloading* de uma tarefa específica, de modo a fazer com que uma decisão de *offloading* seja obrigatoriamente contínua, pode ocasionar novas transmissões de

dados associadas àquela tarefa. Essas transmissões de dados adicionais podem ter um impacto significativo no *makespan*.

Um dos DAGs de aplicação do *data set* usado nesta tese é mostrado na Fig. 4.2 como contraexemplo da política *one-climb*. As tarefas em azul são aquelas que devem ser *offloaded* para o servidor de borda segundo a decisão de *offloading* ótima dessa aplicação, a qual resulta em um *makespan* de 139,17 s. No entanto, há uma descontinuidade em um único caminho do DAG, destacada em vermelho. Segundo a política *one-climb*, o *offloading* da tarefa 17 é obrigatório e deve fornecer um *makespan* total menor que o obtido pela decisão mostrada. Ao marcar a tarefa 17 para *offloading*, a decisão se torna contínua e os tempos de transmissão das arestas (11, 17) e (17, 18) são devidamente anulados. Porém, essa anulação não é refletida no *makespan* total da aplicação, que é afetado negativamente, pois agora são realizadas as transmissões de dados referentes às arestas (6, 17), (7, 17), (8, 17), (9, 17), (12 , 17), (13, 17), (14, 17), (15, 17), (17, 19) e (17, 21), representadas pelas linhas tracejadas em laranja. O *makespan* dessa solução contínua é de 237,04 s, uma deterioração de 70% em relação à decisão ótima (e descontínua). Assim, uma decisão de *offloading* descontínua não é necessariamente uma decisão ruim. Na verdade, os resultados no Capítulo 6 mostram que cerca de 17% das decisões ótimas para o *data set* real utilizado nesta tese violam a política *one-climb*.

4.3 TALON

TALON é a heurística proposta nesta tese para encontrar decisões de *offloading* com baixo *makespan* e resolver o problema PODAG. A execução de TALON pode ser feita tanto no dispositivo móvel quanto no servidor de borda, porém, em ambos os casos, uma comunicação inicial entre os dois dispositivos é necessária para obter informações de *hardware*, como o número de CPUs disponíveis, para que o cálculo do *makespan* seja feito corretamente.

TALON gera um pequeno conjunto de decisões de *offloading* para reduzir o *makespan* de uma aplicação, acrescentando iterativamente decisões de *offloading* a esse conjunto. A primeira decisão inserida nesse conjunto é a decisão Sem *Offloading*, que não marca nenhuma tarefa para *offloading*, e a última decisão inserida é a de *Offloading* Total, que marca para *offloading* todas as tarefas que possam ser *offloaded*. Após a geração do conjunto, cada decisão de *offloading* é testada e aquela que fornecer o menor *makespan* é escolhida.

A geração desse conjunto é feita em duas fases e o tamanho do conjunto é flexível, sendo um parâmetro de entrada de TALON. Na primeira fase, TALON busca de maneira gananciosa pela aresta com o maior tempo de transmissão e faz o *offloading*

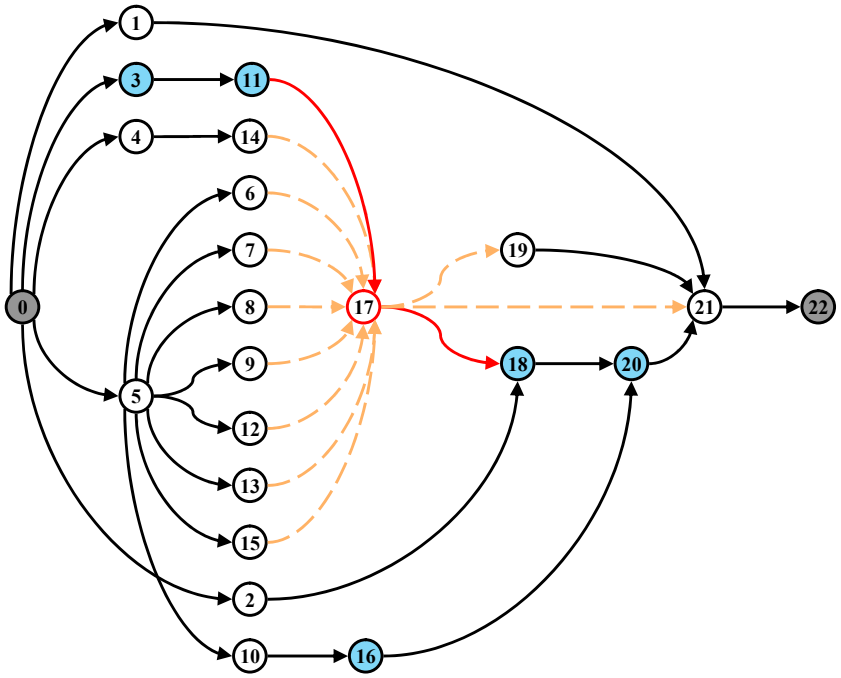


Figura 4.2: Exemplo de um DAG genérico para o qual a decisão ótima não obedece à política *one-climb*.

do par de tarefas que compõem essa aresta para anular a transmissão de dados entre elas.

É importante pontuar que, ao se observar uma aresta, se o par de tarefas que a compõe envolve uma tarefa que esteja ligada a uma âncora, o tempo de transmissão dessa âncora deve ser aplicado ao tempo de transmissão original da aresta em observação. Isso é feito porque realizar o *offloading* desse par de tarefas automaticamente aciona a transmissão de dados da âncora, seja aquela ligada à tarefa inicial ou à final. Isso reduz o benefício em termos de redução de *makespan* obtido ao anular o tempo de transmissão da aresta em observação. Se o tempo de transmissão dos dados das âncoras for muito alto ou existirem muitas âncoras ligadas ao par de tarefas *offloaded*, é possível que o benefício de anular a transmissão da aresta em observação seja inferior ao tempo de transmissão adicional associado às âncoras, o que pode levar a um aumento do *makespan* da aplicação.

Supondo que uma aresta (i, j) tenha, por exemplo, um tempo de transmissão de 50 s e que seja o maior no DAG, realizar o *offloading* das tarefas compondo essa aresta pode parecer útil para reduzir o *makespan*. De fato, o tempo de transmissão dessa aresta é anulado. No entanto, se houver, por exemplo, uma âncora entre a

tarefa inicial e a tarefa i e outra entre a tarefa j e a tarefa final, cada uma com tempo de transmissão de 35 s, realizar o *offloading* de i e de j é, na verdade, prejudicial e leva a um aumento de 20 s do *makespan*.

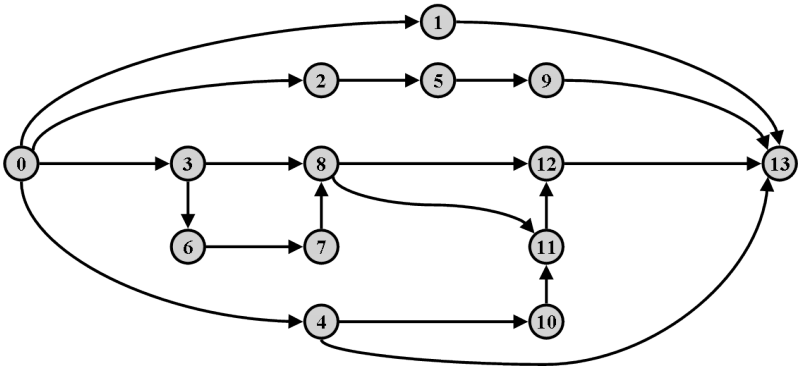
O processo da primeira fase continua para a próxima aresta com maior tempo de transmissão, priorizando agora as arestas que estejam ligadas a tarefas já marcadas para *offloading*. Caso não haja mais arestas ligadas a tarefas marcadas para *offloading*, a busca gananciosa é feita nas arestas que não tenham nenhuma tarefa marcada para *offloading* e a aresta com o maior tempo de transmissão é selecionada. À medida que cada nova aresta é selecionada dessa maneira, ela se junta às anteriores formando uma nova decisão de *offloading* para ser testada ao final.

Para um DAG de aplicação com N tarefas elegíveis para *offloading*, a primeira fase continua até que s_I tarefas sejam marcadas para *offloading*, onde s_I é um parâmetro de entrada que define o *threshold* da mudança da primeira para a segunda fase. Então, a segunda fase de TALON gera todas as $2^{s_{II}}$ decisões de *offloading* possíveis para as $s_{II} = N - s_I$ tarefas restantes, mantendo a marcação de *offloading* das s_I tarefas contempladas pela primeira fase. Após a segunda fase, TALON calcula o *makespan* associado a cada decisão e, finalmente, seleciona aquela que resultar no menor *makespan*.

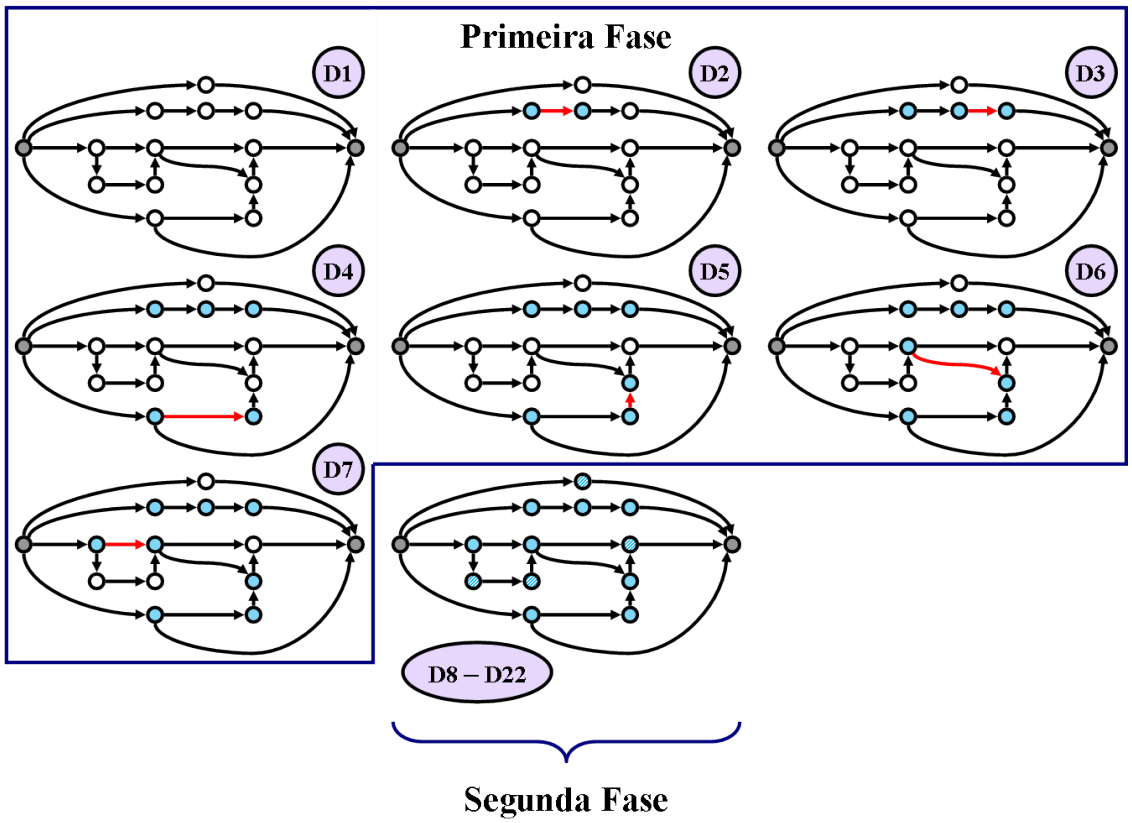
A escolha de s_I e, consequentemente, de s_{II}, uma vez que $s_I + s_{II} = N$ é feita pelo dispositivo responsável por executar TALON, seja o dispositivo móvel ou o servidor de borda, e define a quantidade de decisões de *offloading* que se deseja incluir no conjunto de decisões de TALON para as quais o *makespan* é calculado. Quanto mais decisões no conjunto, maior é o tempo de execução de TALON, porém mais próximo do *makespan* ótimo TALON tende a chegar. Em particular, quando $s_{II} = N$, o *makespan* é calculado para todas as 2^N decisões possíveis e TALON sempre alcança o ótimo, ao custo de uma execução excessivamente demorada. Esse *trade-off* deve ser avaliado conforme o que se deseja priorizar, podendo variar a depender das aplicações em execução.

Um exemplo de funcionamento da heurística TALON é mostrado na Figura 4.3. O DAG possui 14 tarefas, das quais duas são a tarefa inicial e a tarefa final. Como essas duas tarefas sempre são executadas no dispositivo móvel e não podem ser *offloaded*, elas permanecem em cinza. Então, apenas as doze tarefas restantes podem ser marcadas para *offloading*. No exemplo, considera-se que $s_I = 8$, isto é, que a primeira fase perdura até que oito tarefas tenham sido marcadas para *offloading*. Desse ponto em diante, TALON entra na segunda fase. As tarefas marcadas para serem executadas no dispositivo móvel e no servidor de borda estão em branco e em azul, respectivamente.

A primeira decisão de *offloading* gerada por TALON (D1) é a decisão Sem *Offloading*. Supondo que a aresta (2, 5) tenha o maior tempo de transmissão do



(a) Um exemplo genérico de DAG de aplicação.



(b) Decisões de *offloading* geradas em cada fase.

Figura 4.3: Ilustração do funcionamento da heurística TALON em suas duas fases.

DAG e que esse tempo seja de 77 s, esse é o ponto de partida de TALON. Como a tarefa 2 está ancorada na primeira tarefa (tarefa 0), o tempo de transmissão da aresta (0, 2) deve ser subtraído daquele da aresta (2, 5). Se a transmissão de dados na aresta (0, 2) leva 13 s, então o tempo associado à aresta (2, 5), conforme considerado pela busca gananciosa de TALON, passa a ser de 64 s. Supondo que essa ainda seja a transmissão mais demorada de todo o DAG, as tarefas 2 e 5 são marcadas para *offloading* (D2). Agora, apenas as arestas que se ligam ou à tarefa 2 ou à tarefa 5 são consideradas e, portanto, a única aresta disponível para consideração

é a aresta (5, 9), resultando na marcação para *offloading* da tarefa 9 e formando a decisão de *offloading* (D3). De forma similar, esse processo continua até que $s_I = 8$ tarefas tenham sido marcadas para *offloading*. No exemplo, isso ocorre com o *offloading* do par de tarefas 4 e 10 e, em seguida, das tarefas 11, 8, e 3, gerando mais quatro decisões (D4, D5, D6 e D7).

Nesse ponto, TALON transiciona para sua segunda fase e gera $2^4 = 16$ decisões de *offloading* adicionais, já que $s_{II} = 4$ considerando as tarefas 1, 6, 7 e 12. A última decisão gerada na primeira fase (D7) é uma delas, então apenas 15 decisões de offloading inéditas são realmente geradas, finalizando com a decisão *Offloading* Total. O *makespan* de cada uma das 22 decisões de *offloading* é calculado e aquela que resulta no *makespan* mais curto é tomada por TALON como solução para o DAG de aplicação.

Em alguns casos específicos, é possível que não haja arestas suficientes para passar de fase, ou seja, até que o número de s_I tarefas marcadas para *offloading* seja atingido. Isso acontece quando o DAG possui muitas tarefas associadas apenas às arestas de âncora, como é o caso da tarefa 1 do exemplo. Quando são esgotadas as arestas elegíveis para a busca gananciosa da primeira fase e o número de tarefas marcadas para *offloading* é inferior a s_I, um ganho de *offloading* G_k é calculado para cada tarefa k ainda não marcada para *offloading*, dado por

$$G_k = t_k^m - t_k^e - \sum_{\substack{(i,j)\in E| \\ k\in\{i,j\}}} T_{i,j}, \tag{4.1}$$

onde t_k^m e t_k^e são os tempos de processamento da tarefa k no dispositivo móvel e no servidor de borda, respectivamente, e $T_{i,j}$ é o tempo de transmissão da aresta (i, j), sendo ou $k = i$ ou $k = j$. Em seguida, as tarefas são marcadas para *offloading*, do maior para o menor ganho, o que gera decisões de *offloading* adicionais com o objetivo de atingir o número de s_I tarefas marcadas para *offloading* e iniciar a segunda fase de TALON. Por fim, o esquema completo do funcionamento de TALON é apresentado no formato de pseudocódigo conforme o Algoritmo 1.

4.4 Complexidade Computacional

A complexidade computacional de TALON varia amplamente, já que ela depende de alguns fatores associados à estrutura do DAG, como o número de tarefas N, o número de âncoras A, o número de arestas livres (ou seja, que não são âncoras) L, o parâmetro de mudança de fase s_I e um fator adicional k.

Como é possível que, ao final do primeiro *loop* Enquanto (linhas 6-12), o número de tarefas marcadas para *offloading* seja menor que s_I, o segundo *loop* Enquanto

Algoritmo 1: TALON.

1 **faça** L ser o conjunto de arestas em E que não sejam âncoras
2 **para cada** $(i, j) \in L$ **faça**
3 | **subtrair** de $T_{i,j}$ o valor de cada $T_{u,v}$, tal que $(u, v) \in E \setminus L$, sendo $u \in \{i, j\}$ ou $v \in \{i, j\}$
4 **faça** D ser a decisão de *offloading* para a qual não há tarefas marcadas para *offloading*
5 $\mathcal{D} \leftarrow \{D\}$
6 **enquanto** $L \neq \emptyset$ e s_I for maior que o número de tarefas marcadas para *offloading* em D **faça**
7 | **se** existir algum $(i, j) \in L$ tal que k seja a única tarefa de $\{i, j\}$ não marcada para *offloading* em D **então**
8 | | **marcar** k para *offloading* em D, dado que $T_{i,j}$ seja o maior tempo de transmissão dentre aqueles referentes às arestas contendo alguma tarefa já marcada para *offloading*
9 | **senão**
10 | | **marcar** ambas as tarefas i e j para *offloading* em D, dado que $T_{i,j}$ seja o maior tempo de transmissão de todas as arestas em L
11 | $\mathcal{D} \leftarrow \mathcal{D} \cup \{D\}$
12 | $L \leftarrow L \setminus \{(i, j)\}$
13 **faça** P ser o conjunto de tarefas ainda não marcadas para *offloading* em D
14 **enquanto** houver menos de s_I tarefas marcadas para *offloading* em D **faça**
15 | **marcar** a tarefa m para *offloading* em D, dado que G_m é o maior ganho dentre aqueles referentes às tarefas em P
16 | $\mathcal{D} \leftarrow \mathcal{D} \cup \{D\}$
17 | $P \leftarrow P \setminus \{m\}$
18 **para cada** uma das $2^{s_{II}}$ possibilidades conjuntas de marcações para *offloading* das s_{II} tarefas ainda não marcadas para *offloading* em D **faça**
19 | $D' \leftarrow D$
20 | **marcar** as s_{II} tarefas para *offloading* em D' de acordo com todas as possibilidades de marcações conjuntas consideradas
21 | $\mathcal{D} \leftarrow \mathcal{D} \cup \{D'\}$
22 **faça** D^* ser a decisão de *offloading* em $\mathcal{D}$ cujo cálculo do *makespan* forneça o menor tempo em relação às demais decisões em $\mathcal{D}$
23 **retorna** D^*

(linhas 13-16) garante essa condição, para sair da primeira fase e iniciar a segunda (linha 17), então k é o número de tarefas *offloaded* ao final da linha 12, de modo que

$k \leq s_I$. Em resumo, a complexidade computacional de TALON é regida por quatro partes.

1. A ordenação de arestas (por tempos de transmissão) e de vértices (por ganhos de *offloading*) durante a busca gananciosa (primeira fase) são implicitamente assumidas. Considerando que $|L|$ é $O(N^2)$, toda a ordenação necessária pode ser feita em um tempo $O(N^2 \log N)$.

2. Os *loops* Enquanto nas linhas 6 e 14 fazem juntos $s_I = O(N)$ iterações e cada uma requer um tempo $O(1)$ para ser concluída.

3. O *loop* Para na linha 18, que itera 2^{N-s_I} vezes, precisa de $O(N)$ em cada iteração. Para $N - s_I = O(\log N)$, como no Capítulo 6, esse *loop* leva um tempo $O(N^2)$.

4. Os cálculos de *makespan* na linha 22, um para uma das N decisões de *offloading*, podem ser realizados para cada decisão da seguinte forma. Uma busca em largura é feita em tempo $O(N^2)$, então uma fila de eventos de tamanho $O(N^2)$ é construída ao longo do caminho. Os eventos nessa fila são os eventos interligados de início e fim de execuções de tarefas e transmissões de dados. Ordenar essa fila com base na ordem cronológica (crescente) dos eventos, o que leva um tempo $O(N^2 \log N)$ e, em seguida, processá-la, que demora um tempo $O(N^2)$, resultam no *makespan* do DAG. No geral, tempo de $O(N^3 \log N)$ é necessário.

A soma de todas as quatro partes produz uma complexidade computacional de $O(N^3 \log N)$, que é ligeiramente maior do que a solução de Guo *et al.* [46], que é de $O(N^3)$, sendo N o número de tarefas no DAG de aplicação. No entanto, para DAGs com $|L| = O(N \log N)$, a complexidade de TALON é reduzida para $O(N^2 \log^2 N)$ e, para $|L| = O(N)$, é reduzida ainda mais para $O(N^2 \log N)$, inferior à complexidade da solução de Guo *et al.* [46].

TALON é uma heurística projetada para ser flexível. A depender do parâmetro s_I, que denota a mudança da primeira para a segunda fase, o tamanho do conjunto de decisões de *offloading* gerado pode ser muito reduzido, na ordem de aproximadamente N decisões de *offloading* quando $s_I = N$, ou muito vasto, chegando a 2^N quando $s_I = 0$. Nesse último caso, TALON funciona exatamente como uma busca exaustiva e consegue alcançar a decisão de *offloading* ótima, pois testa todas as decisões de *offloading* possíveis.

A escolha dos parâmetros s_I e s_{II} permite que TALON se adapte a vários tipos de aplicação e deve ser feita pelo dispositivo executando TALON. Novamente, no caso de $s_I = N$, o *makespan* é calculado apenas para um conjunto na ordem

de N decisões de *offloading*. Com isso, TALON consegue encontrar a decisão de menor *makespan* em um tempo significativamente menor que aquele para o caso de $s_I = 0$, no qual o *makespan* é calculado para 2^N decisões de *offloading*. Isso pode viabilizar a utilização de TALON de forma *online*, ou seja, durante a utilização das aplicações pelos usuários. Para isso, o tempo de execução de TALON, o qual está associado à escolha de s_I, deve ser pequeno em relação ao ganho (em termos de redução de *makespan*) proporcionado por uma boa decisão de *offloading*.

Capítulo 5

Análise de Dados

TALON é desenvolvida para funcionar independentemente da topologia do DAG de aplicação, com base em dados realistas. Os trabalhos mais recentes tratam de topologias simples, como DAGs sequenciais [119, 120], em árvore [48] ou *fan-in/fan-out* [44, 121]. Outros trabalhos sequer consideram as dependências entre as tarefas de uma aplicação e se destinam, ao invés disso, a outros aspectos do problema PODAG [122, 123], como tarefas atômicas modeladas como aglomerados de *bytes* enviados em lotes.

Nesta tese, os DAGs de aplicação são extraídos do *Alibaba Cluster Trace Program* de 2018[1]. Esse *data set* contém informações referentes a milhões de DAGs de aplicação submetidos a um *cluster* de mais de 4000 máquinas.

5.1 Descrição do *Data Set*

O *data set* descreve os DAGs de aplicação em dois arquivos, um para as aplicações, que inclui, por exemplo, a topologia do DAG e o *status* da execução (bem-sucedida, com falha etc.), e outro para o que o *data set* chama de instâncias, que possuem a especificação interna das tarefas. Juntos, esses arquivos contêm informações sobre tamanho da memória, porcentagem de utilização de CPU, tempo de início e tempo de finalização de cada instância, por exemplo.

Como as instâncias fazem parte de uma tarefa (ou seja, um vértice do DAG) e os detalhes sobre suas dependências internas são omitidos dos arquivos do *data set*, esta tese pressupõe que todas as instâncias de uma determinada tarefa sejam aglutinadas em uma única instância. Isso soluciona o problema de ausência de informações sobre a topologia das instâncias, pois, seja qual for a estrutura interna, elas demarcam o início e o fim da tarefa à qual estão associadas. Isso é suficiente

[1] https://github.com/alibaba/clusterdata

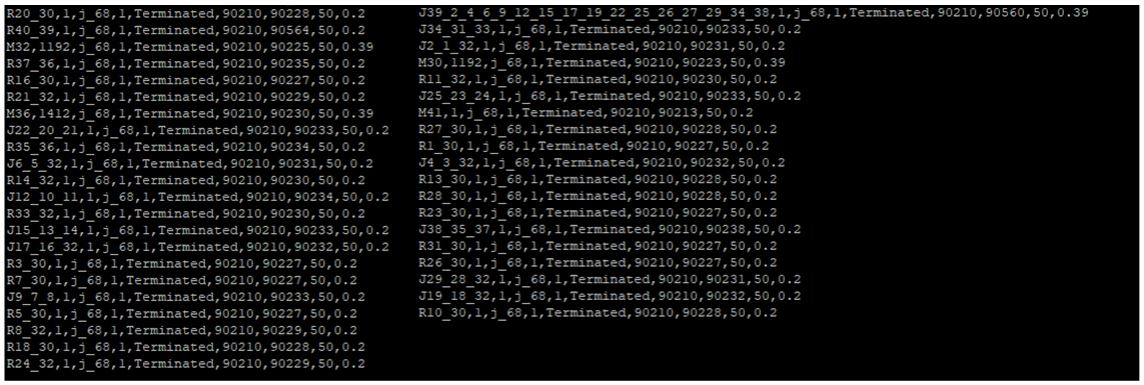


Figura 5.1: Um exemplo das informações contidas no *data set*.

para que seja respeitada a estrutura dos DAGs de aplicação, mesmo sem conhecê-la em nível de instância.

Outro ponto associado à escassez de informações sobre a topologia das instâncias é o tempo de processamento de cada tarefa. Não é possível afirmar quais instâncias executam sequencialmente ou em paralelo, o que é necessário para calcular o tempo de processamento da tarefa. Diante disso, ao comparar os dois arquivos do *data set*, percebe-se que o tempo de processamento da tarefa reportado no arquivo das aplicações não corresponde à soma do tempo de processamento de cada instância, reportado no arquivo das instâncias. Na verdade, o tempo de processamento da tarefa está mais próximo do tempo médio de execução de todas as instâncias individuais. Assim sendo, para fins de simplificação, é feita a junção das instâncias em uma única instância, que corresponde à tarefa em si, a qual é indivisível, conforme a modelagem no Capítulo 3, e seu tempo de processamento é igual à média do tempo de processamento de todas as instâncias.

A Figura 5.1 mostra um exemplo da saída em terminal das informações de um DAG de aplicação, referente à aplicação *j_68*, que possui 41 tarefas. Da esquerda para a direita, as colunas representam: nome da tarefa e suas tarefas predecessoras, número da instância executada, nome da aplicação, tipo de tarefa, *status*, *timestamp* de início da execução da tarefa (em segundos), *timestamp* do fim da execução da tarefa (em segundos), utilização de CPU e utilização de memória. A primeira linha mostra a tarefa *R20*, cuja única predecessora é a tarefa 30, isto é, existe uma aresta (30, 20). A execução de *R20* conta com apenas uma instância, a execução é bem-sucedida (*terminated*) e o tempo entre as estampas de início e fim é de 18 s. Há compartilhamento de CPU (*multithreading*), dado que a utilização de CPU é de 50%, então a CPU é ocupada por outra tarefa na metade do tempo, e a estrutura de dados da tarefa *R20* ocupa um tamanho de 0,2% da memória da máquina.

Os números exatos de utilização de CPU e memória não são disponibilizados no *data set*, mas as informações de *hardware* podem ser presumidas com base na família de instâncias SCCG5 da Alibaba Cloud, um *cluster* de supercomputação de

uso geral[2]. A família SCCG5 é voltada para aplicações de aprendizado de máquina, codificação de vídeo, análise de dados, servidores de jogos *mobile* e *online*, simulações computacionais e outros fins. A Tabela 5.1 mostra as especificações de *hardware* usadas nesta tese. Informações adicionais sobre *FLoating-point OPerations* (FLOPs) por ciclo estão disponíveis em Dolbeau [124].

Tabela 5.1: Informações de *hardware* das máquinas que compõem o *cluster* SCCG5.

Arquitetura de CPU	*Clock*	Número de CPUs	Memória	FLOPs/ciclo[a]
Intel Xeon Platinum 8163	2,5 GHz	48	384 GiB	32

[a] https://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%208163.html

Embora existam algumas informações sobre a utilização de memória por cada tarefa, não é possível inferir a partir do *data set* quantos *bits*, os quais pertencem à estrutura de dados associada a cada tarefa, são passados nas transmissões de dados entre elas. Dessa forma, uma nova suposição é feita para lidar com essa lacuna nas informações do *data set* e a quantidade de dados passados nas arestas do DAG é escolhida aleatoriamente com os seguintes critérios. Dada uma tarefa i, cuja estrutura de dados ocupa um espaço na memória de tamanho igual a d_i *bytes*, a soma dos dados transmitidos por todas as tarefas predecessoras a i em direção à mesma não pode exceder o tamanho d_i. Isso é razoável porque, conforme os dados são recebidos pela tarefa i, eles precisam ser armazenados em memória no espaço alocado à tarefa i para posteriormente serem processados. Assim, o tamanho da estrutura de dados utilizada por uma tarefa, no mínimo, deve ser equivalente ao total de *bytes* recebidos de suas tarefas predecessoras.

Como a soma dos dados transmitidos não pode exceder d_i, a quantidade de *bytes* passados em cada aresta predecessora a i é escolhida aleatoriamente entre 1, como um *char* ou um *unsigned integer*, e d_i *bytes*. Para não assumir nenhuma especificidade quanto à natureza das aplicações e suas tarefas, isto é, se é mais comum que poucos *bytes* sejam transmitidos ou o contrário, a escolha aleatória da quantidade de dados passados nas arestas do DAG segue uma distribuição uniforme.

5.2 Tratamento dos Dados das Aplicações

O *data set* de Alibaba possui informações da execução de mais de quatro milhões de DAGs de aplicação, porém alguns deles incluem execuções com erros ou campos de informação vazios. Para as simulações apresentadas nesta tese, um subconjunto de DAGs de aplicação é extraído de diferentes partes do *data set* da seguinte forma.

[2] https://www.alibabacloud.com/help/en/doc-detail/25378.htm#sccg5

- Procurar por DAGs que tenham entre 18 e 28 tarefas em sua composição. O limite inferior remove DAGs cujas estruturas sejam muito simples e com pouca diversidade, os quais são numerosos no *data set*, como os DAGs frequentemente considerados na literatura. Já o o limite superior é aplicado porque é importante comparar o desempenho de TALON com o da decisão de *offloading* ótima, cuja obtenção é excessivamente demorada [41], sendo cada vez menos viável quanto maior for o DAG. Nesta tese, a decisão ótima é encontrada atavés de busca exaustiva.

- Remover os DAGs cujas informações de topologia não estejam completas.

- Remover os DAGs sem informação de utilização de memória ou de CPU.

- Manter apenas os DAGs que tenham terminado suas execuções de forma bem-sucedida.

- Cessar a varredura do *data set* após uma hora de execução. Por se tratar de um *data set* grande, esse recorte é necessário até mesmo para que seja viável calcular o *makespan* ótimo de cada aplicação, o que é inviável caso isso seja feito para os mais de quatro milhões de DAGs do *data set*. Com isso, as informações sobre os DAGs não são conhecidas para o *data set* inteiro, mas sim para o subconjunto de DAGs extraído do mesmo.

Ao final desse processo, são extraídos 1322 DAGs de aplicação. Sabendo que o *data set* inclui DAGs de aplicação executados em *clusters* da Alibaba Cloud, é razoável supor que esses DAGs são enviados pelos usuários finais remotamente, com relação aos servidores do *cluster*, e que os resultados do processamento da aplicação são devolvidos aos usuários finais. Assim, após a obtenção dos DAGs, que possuem entre 18 e 28 tarefas, duas novas tarefas são adicionadas a cada DAG para representar o comportamento de uma aplicação rodando em um dispositivo móvel. Uma tarefa inicial é conectada às tarefas que não possuem arestas de entrada e uma tarefa final é conectada àquelas que não possuem arestas de saída, conforme mostrado na Fig. 5.2.

Como as tarefas inicial e final não podem ser *offloaded* e, portanto, seus tempos de processamento são sempre os mesmos em todas as decisões de *offloading*, essas duas tarefas não são relevantes para o cálculo do *makespan*. Sendo assim, ambas recebem tempo de processamento nulo, pois não há como elas contribuírem para a redução do *makespan*. Por outro lado, os tempos de transmissão das tarefas inicial e final geram impacto sobre o *makespan* e devem ser considerados. Como essas duas tarefas são acrescentadas aos DAGs para representar o usuário nesta tese, não há menção às mesmas no *data set*. Então, esses tempos são gerados de modo similar ao descrito na Seção 5.1. As transmissões de dados que partem da tarefa inicial

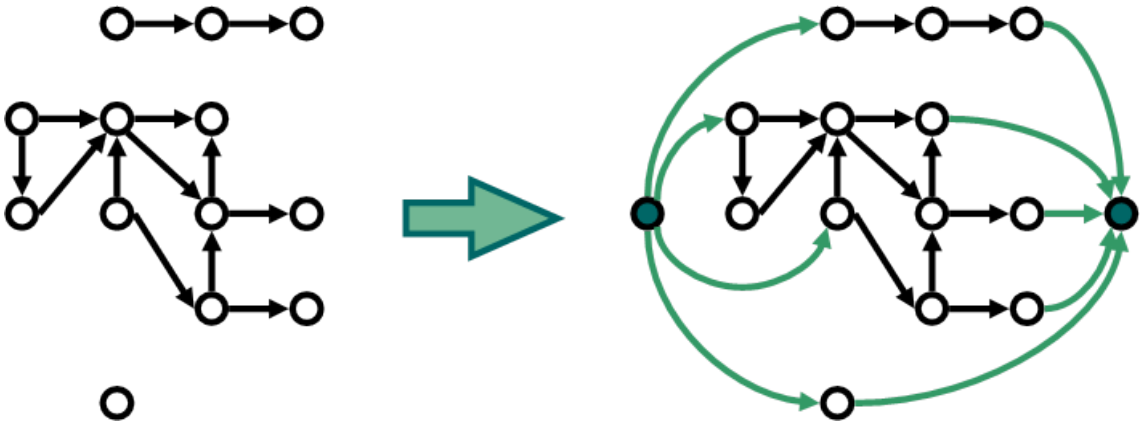


Figura 5.2: Inclusão de duas tarefas adicionais e suas respectivas arestas (em verde) para representar o usuário final.

dependem do tamanho da estrutura de dados de suas tarefas sucessoras. Já as transmissões que se destinam à tarefa final levam em consideração o tamanho das estruturas de dados das tarefas predecessoras à mesma.

5.3 Propriedades dos DAGs

Além de se trabalhar com DAGs que representem, de fato, aplicações reais, também é importante que os DAGs sejam diversificados em sua estrutura, pois é esperado que a heurística TALON tenha um bom desempenho em qualquer cenário. A Figura 5.3 mostra como os DAGs de aplicação estão distribuídos de acordo com o número de vértices, o número de arestas e a densidade, respectivamente. A densidade varia entre 0, quando um gráfico não tem arestas, e 1, quando há uma aresta para cada um dos $\frac{N(N-1)}{2}$ pares não ordenados de vértices, sendo N o número de vértices, semelhante a uma topologia de malha completa.

Com relação ao número de tarefas em cada aplicação, mesmo para o caso mais frequente, ou seja, DAGs com 23 tarefas, esses ainda são menos de 25% do total, enquanto os demais 75% do total estão distribuídos de forma mais uniforme. Quanto ao número de arestas e às densidades dos DAGs, ambos tendem a se concentrar ao redor dos valores mais intermediários, havendo poucos DAGs nos valores mais baixos e mais altos. Isso é razoável, uma vez que esses casos extremos geralmente indicam um DAG pouco estruturado: poucas dependências entre tarefas no primeiro caso, muitas no segundo.

Dada a inserção das tarefas inicial e final na Seção 5.2, aqui todos os DAGs são obrigatoriamente conexos, isto é, há pelo menos um caminho ao longo do qual é possível atravessar o DAG entre quaisquer pares de vértices i e j nele. Com isso, o menor número de arestas possível aqui é de $N - 1$ para um DAG sequencial de N tarefas. Um pequeno número de arestas e densidades, as quais são em sua maioria

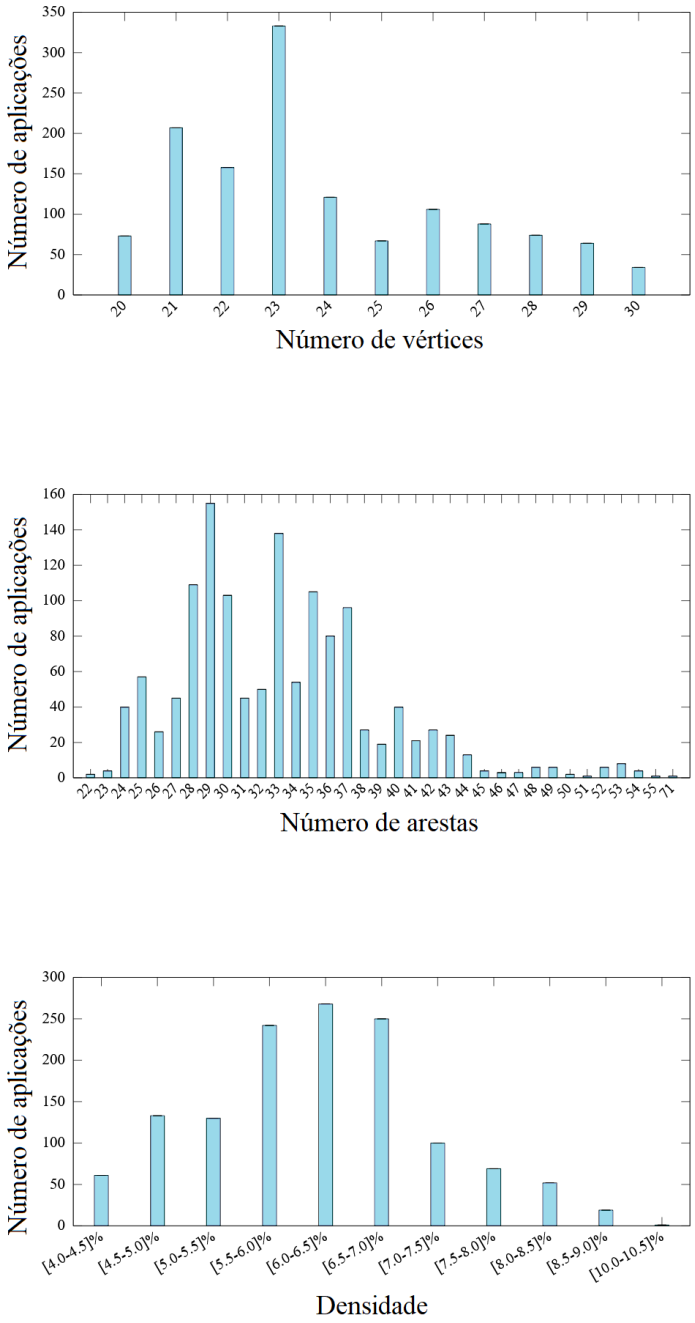


Figura 5.3: Distribuições das propriedades dos DAGs.

abaixo de 9%, como mostrado na Figura 5.3, é esperado. Isso se deve ao fato de as aplicações não apresentarem recursão de tarefas (caso contrário, em vez de DAGs, haveria gráficos cíclicos, como em Zhang *et al.* [47]), e nem todas as tarefas podem ser executadas em paralelo, como no caso de uma *thread* sequencial.

Capítulo 6

Resultados

Para avaliar o desempenho de TALON, são propostos três cenários de simulação, cada um variando um parâmetro de entrada diferente, conforme mostrado na Tabela 6.1. O primeiro cenário considera a variação do número de CPUs disponíveis no dispositivo móvel. O segundo considera essa variação, mas no servidor de borda. O terceiro considera diferentes taxas de transmissão de dados para o canal de comunicação compartilhado.

As simulações são conduzidas e os valores de *makespan* obtidos em cada caso são avaliados. Duas máquinas são utilizadas para a realização das simulações, uma Intel(R) Xeon(R) CPU E5-1650 v4 @ 3,60 GHz com 6 CPUs físicas e 128 GB de memória RAM, e outra Intel(R) Xeon(R) W-2265 CPU @ 3,50 GHz com 12 CPUs físicas e 128 GB de memória RAM. Além disso, também são medidos os valores de taxa de *offloading*, que é a proporção do número de tarefas *offloaded* em relação ao total da aplicação, similaridade entre as decisões de aproximação (heurística) e as decisões ótimas e, por fim, o grau de paralelismo no processamento de tarefas.

Em cada um dos três cenários, TALON é testada para três valores diferentes de s_{II}, a saber, $s_{II} \in \{0, \log_2 N, \log_2 N^2\}$, o que também faz com que os valores de s_I sejam definidos, já que, por definição, $s_I + s_{II} = N$. Esses valores para s_{II} definem quantas decisões de *offloading* adicionais são geradas na segunda fase do Algoritmo 1, sejam 1, N ou N^2 decisões adicionais, respectivamente. Ao longo desses experimentos, TALON é comparada com o *Heuristic Offloading Algorithm* (HOA) de Guo *et al.* [46] e também com duas soluções triviais, *Offloading* Total e Sem *Offloading*.

Todas as estratégias acima são executadas para cada um dos 1322 DAGs de aplicação, para as quais os valores de *makespan* são obtidos. Em seguida, o *makespan* ótimo é calculado para cada cenário através de uma busca exaustiva, testando cada decisão de *offloading* possível. Tendo uma aplicação com T tarefas que podem ser *offloaded*, existem 2^T decisões de *offloading* possíveis. Com relação a HOA, o modelo de cálculo do *makespan* adotado é diferente do utilizado aqui, conforme elaborado

Tabela 6.1: Parâmetros de entrada considerados nas simulações.

Nome	Número de CPUs (móvel)	Número de CPUs (borda)	Taxa de transmissão
Primeiro cenário	2; 4; 8 e Irrestrito	16	20 Mbps
Segundo cenário	Irrestrito	1; 2; 4 e 16	20 Mbps
Terceiro cenário	Irrestrito	16	10; 20 e 100 Mbps

no Capítulo 3. Com isso, toda a implementação e a modelagem do sistema de Guo *et al.* [46] são feitas fielmente para que execução de HOA possa acontecer da forma que ela é planejada para ocorrer. Isso faz com que as decisões ótimas e o *makespan* mínimo sejam diferentes nos dois modelos.

HOA sempre assume um número irrestrito de CPUs no dispositivo móvel. Se todas as CPUs no servidor de borda estiverem ocupadas, novas tarefas serão enfileiradas até que uma CPU fique disponível, ao invés de compartilhar CPUs com as tarefas já em execução. Além disso, HOA não realiza mais de uma transmissão de dados por vez, escalonando as transmissões de forma FIFO. Essas diferenças entre o modelo adotado nesta tese para TALON (e as duas soluções triviais) e o modelo adotado para HOA implicam que as decisões ótimas possam ser diferentes para o mesmo DAG. Outra particularidade de HOA é o fato de, sempre que o *makespan* da decisão de *offloading* testada é maior que o *makespan* obtido ao se executar o DAG de aplicação exclusivamente no dispositivo móvel, HOA interrompe seu procedimento e fornece como resultado a decisão de *offloading* associada à Sem *Offloading*.

Por questão de justiça, os resultados de *makespan* são, portanto, apresentados em termos da diferença relativa entre o *makespan* da heurística M_{h} (referente a TALON, HOA, Sem *Offloading* ou *Offloading* Total) e o *makespan* ótimo M_{o} (referente às decisões ótimas para TALON e para HOA). Os valores de *makespan* e outros resultados são apresentados sempre como médias, com intervalos de confiança de 99%.

Por fim, como os tempos de processamento dos DAGs de aplicação executados na Alibaba Cloud se referem à execução em um *cluster* computacional de grande escala, é esperado que esses tempos sejam menores do que aqueles que são obtidos com a execução em um servidor de borda ou em um dispositivo móvel. Então, esses tempos são redimensionados para refletirem a execução no servidor de borda e no dispositivo móvel e melhor se adequarem ao *hardware* dos servidores de borda comerciais e dos *smartphones* atuais, cujas características são mostradas na Tabela 6.2, de acordo

com

$$t_k^m = t_k^c\Big(\frac{f_c c_c}{f_m c_m}\Big) \tag{6.1}$$

e

$$t_k^e = t_k^c\Big(\frac{f_c c_c}{f_e c_e}\Big), \tag{6.2}$$

onde t_k^m, f_m e c_m são o tempo de processamento da tarefa k em segundos, o *clock* em Hz, a capacidade computacional da CPU em operações/ciclo no dispositivo móvel, respectivamente. De forma análoga, t_k^e, f_e e c_e são os valores associados ao servidor de borda e t_k^c, f_c e c_c são associados ao *cluster* da Alibaba Cloud.

Tabela 6.2: Informações de *hardware* do dispositivo móvel e do servidor de borda.

Tipo	Arquitetura de CPU	*Clock*	Número de CPUs	FLOPs/ciclo[a]
Dispositivo móvel[b]	Qualcomm Snapdragon 865	1x2,84; 3x2,42 e 4x1,8 GHz	8	8
Servidor de borda[c]	Intel Xeon D-2100	2,0 GHz	16	32

[a] https://en.wikichip.org/wiki/flops#x86
[b] https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-865-5g-mobile-platform
[c] https://www.intel.com/content/www/us/en/products/docs/processors/xeon/d-2100-brief.html

6.1 Variando o Número de CPUs do Dispositivo Móvel

Além de realizar o processamento das tarefas de um DAG, um dispositivo móvel utiliza suas CPUs para executar outras funções básicas do sistema e ser responsivo aos comandos do usuário final. Por ser um dispositivo com recursos mais restritos de *hardware* em comparação a um *cluster* de servidores de nuvem, o número de CPUs que o dispositivo móvel pode destinar à execução do DAG pode ser um fator limitante.

Esse cenário considera diferentes níveis de disponibilidade de CPU no dispositivo móvel. São considerados 2, 4, 8 ou um número irrestrito de CPUs no dispositivo móvel. Essa capacidade irrestrita é a mesma da abordagem adotada em Guo *et al.* [46], na qual sempre há uma CPU disponível para qualquer tarefa pronta para ser executada. Na prática, isso equivale a considerar que existem n CPUs disponíveis no dispositivo móvel para executar um DAG com n tarefas.

A capacidade do servidor de borda é mantida fixa, havendo a disponibilidade de todas as 16 CPUs do servidor de borda, conforme a Tabela 6.2. A taxa de transmissão do canal sem fio é de 20 Mbps, o que equivale a considerar uma rede de

legado da tecnologia *Long Term Evolution* (LTE) [125]. Uma taxa de transmissão baixa é escolhida de modo a tornar o cenário mais desafiador para a obtenção de um *makespan* baixo. Como a tendência é que os tempos de transmissão sejam mais altos, tomar boas decisões de *offloading* é ainda mais importante. Isso é feito para testar a capacidade de TALON e HOA de encontrarem as arestas mais adequadas para realizar a migração do processamento do dispositivo móvel para o servidor de borda e vice-versa.

A Figura 6.1 mostra que o desempenho de TALON permanece estável independentemente do número de CPUs no dispositivo móvel. TALON só pode ser comparada apropriadamente com HOA quando ambas consideram um número irrestrito de CPUs no dispositivo móvel, com TALON apresentando resultados variando de 4,3% a 5,7% acima do *makespan* ótimo, enquanto HOA está 15,5% acima de seu respectivo *makespan* ótimo.

Vale ressaltar que o fato de HOA sempre considerar recursos irrestritos torna seus cálculos de *makespan* em um limite superior para o *makespan* real, sendo uma aproximação otimista, pois são justamente os dispositivos móveis que possuem recursos de CPU mais limitados, ainda mais ao se tratar de modelos *budget*. Ainda assim, mesmo para o caso de TALON-0, que testa um conjunto muito limitado de decisões de *offloading*, são obtidos desempenhos melhores que os de HOA. Como mostrado na Figura 6.1, as soluções triviais levam a *makespans* inaceitavelmente altos, uma vez que essas estratégias não levam em consideração os tempos de processamento de tarefas ou tempos de transmissão.

Ainda com relação à Figura 6.1, deve-se notar que somente no caso do *Offloading* Total o desempenho piora à medida que aumenta o número de CPUs no dispositivo móvel. Como os valores do eixo vertical se referem à diferença relativa de *makespan*, um aumento na oferta de CPUs no dispositivo móvel beneficia decisões de *offloading* que as utilizem para processar algumas tarefas, o que pode ser útil para reduzir o *makespan* da decisão ótima (indicado por M_o), mas em nada afeta a decisão de *Offloading* Total (indicado por M_h).

6.2 Variando o Número de CPUs do Servidor de Borda (Cenário Multiusuário)

Mesmo com a premissa de que os recursos de *hardware* do servidor de borda sejam mais abundantes do que os do dispositivo móvel, é preciso levar em consideração que um servidor de borda pode ser responsável por atender a muitos usuários simultaneamente e que todas as suas CPUs não podem ser alocadas apenas a um usuário único para evitar, por exemplo, problemas com *starvation* associados aos demais

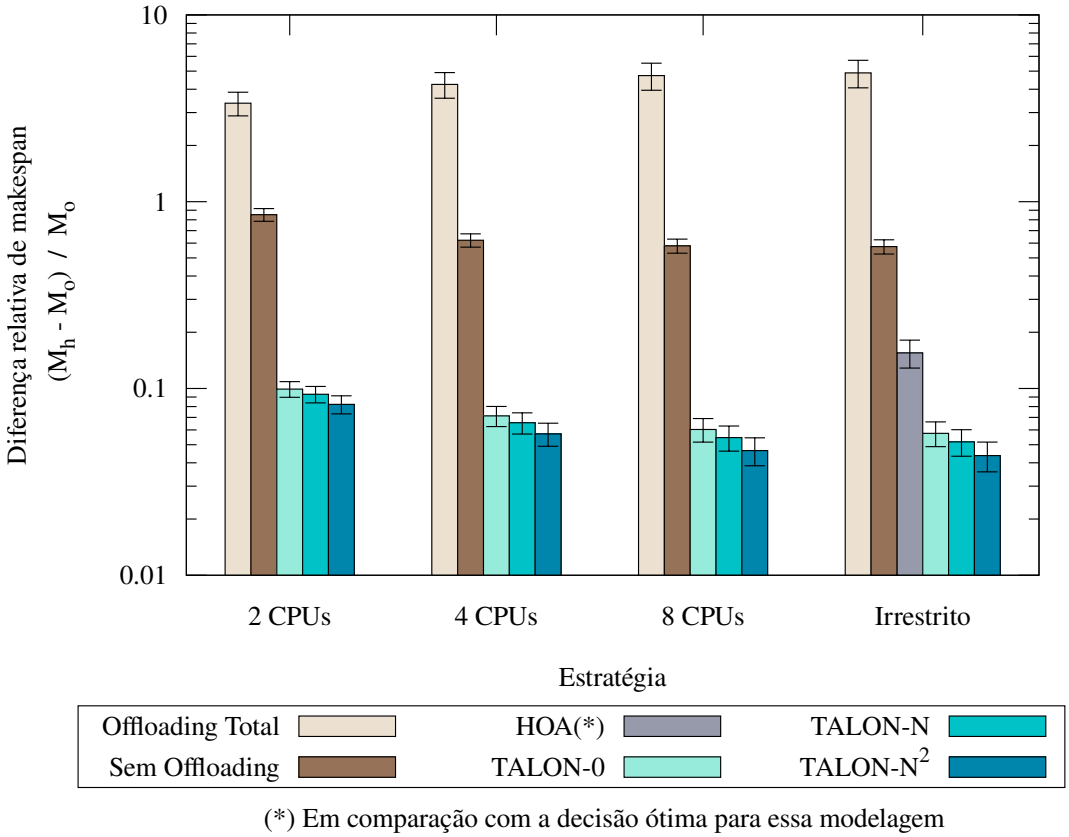


Figura 6.1: Diferença relativa de *makespan* devido à variação no número de CPUs no dispositivo móvel.

usuários. Aqui o número de CPUs disponíveis no servidor de borda varia para emular uma possível ocupação do servidor de borda atendendo a outros usuários. Essa redução é considerada para representar um cenário multiusuário real, sendo razoável presumir que o escalonador do servidor de borda permita a utilização de apenas uma quantidade reduzida de suas CPUs por cada usuário.

Como HOA sempre considera um número irrestrito de CPUs no dispositivo móvel, aqui o mesmo é assumido para TALON, de modo que a comparação entre ambas aconteça em igualdade de condições. Já no servidor de borda, há a disponibilidade de 1, 2, 4 ou 16 CPUs (das 16 CPUs existentes) disponíveis para um usuário. A taxa de transmissão é mantida em 20 Mbps, como antes. Em particular, quando todas as 16 CPUs estão disponíveis no servidor de borda, esse caso é o ponto de interseção entre as Figuras 6.1 e 6.2 (conjunto de barras mais à direita em ambos os gráficos).

A Figura 6.2 mostra a consistência das três versões de TALON, que estão 5,5% a 6,8% acima do *makespan* ótimo quando apenas uma CPU está disponível no servidor de borda. Esses resultados tornam-se ainda melhores à medida que mais CPUs ficam disponíveis, alcançando um valor de *makespan* apenas 4,3% a 5,7% acima do ótimo

quando todas as 16 CPUs estão disponíveis. Por outro lado, o desempenho do HOA piora em relação à sua solução ótima à medida que mais CPUs estão disponíveis, passando de 12,6% acima do *makespan* ótimo, no caso de uma única CPU, para 15,5%, no caso de 16 CPUs. Esse contraste se deve ao fato de que TALON tem mais sucesso na escolha de uma decisão de *offloading*, testando um conjunto de decisões e insistindo no processo de *offloading*.

A solução HOA parte do princípio que a redução do *makespan* deve ser monotônica conforme mais tarefas são marcadas para *offloading*. Quando HOA testa uma nova decisão de *offloading* e o *makespan* dela é maior do que o da execução local (como no caso de Sem *Offloading*), HOA desmarca todas as tarefas para *offloading* e fornece como resultado a execução local. Conforme se tem uma maior abundância de CPUs no servidor de borda, o que é favorável ao *offloading*, esse comportamento de HOA a afasta ainda mais do seu ótimo. Além disso, novamente, ambas as soluções triviais apresentam desempenho significativamente pior do que HOA ou TALON, pois elas não têm como objetivo reduzir *makespan*.

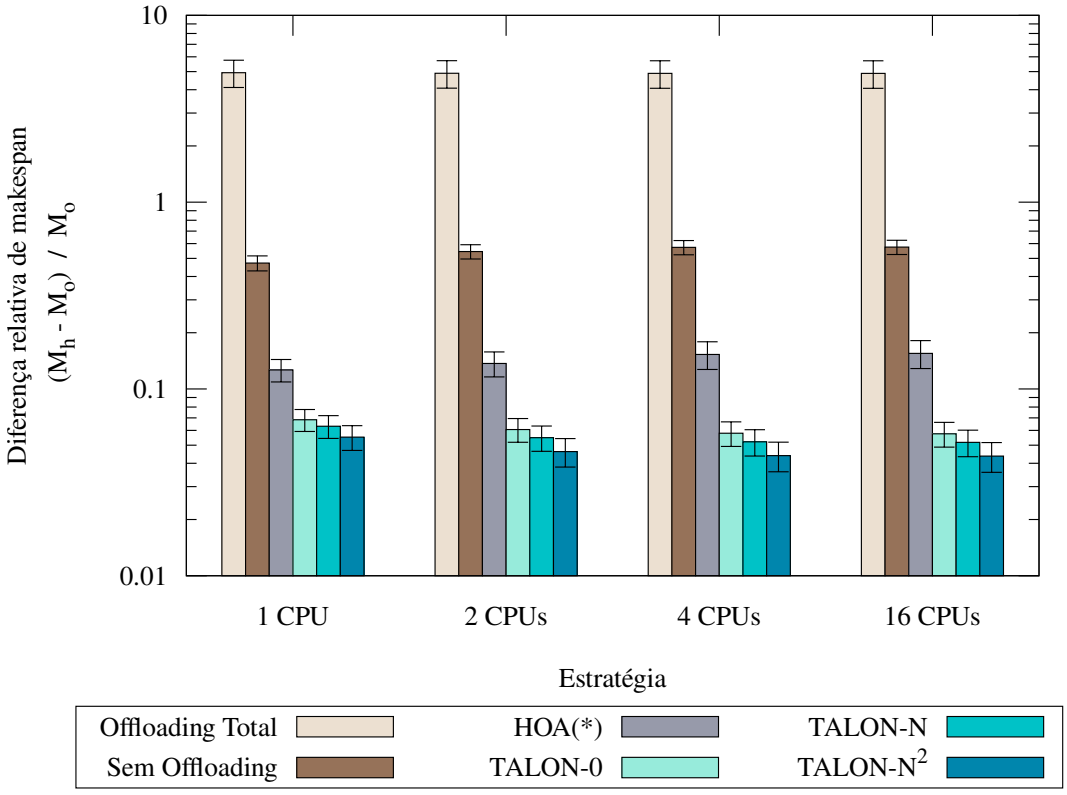


Figura 6.2: Diferença relativa de *makespan* devido à variação no número de CPUs no servidor de borda.

Além dos resultados de diferença relativa de *makespan*, os valores de makespan absolutos revelam outros detalhes importantes para o caso multiusuário, no qual a

comparação entre TALON e HOA pode ser feita de maneira plena para todos os valores do eixo horizontal da Figura 6.2.

A Tabela 6.3 mostra o *makespan* médio para todos os 1322 DAGs de aplicação. Esses DAGs são bastante diversos, variando de aplicações bastante simples a aplicações com uso intensivo de processamento e transmissão. Portanto, é esperado um desvio padrão suficientemente alto a ponto de haver alguma sobreposição entre os intervalos de confiança. Porém, é possível perceber que, TALON consegue alcançar valores de *makespan* menores que os de HOA mesmo em circunstâncias mais restritivas. Isso pode ser entendido como uma maior eficiência por parte de TALON na utilização das CPUs do servidor de borda do que HOA. Por exemplo, Com 16 CPUs, HOA tem um *makespan* médio de 348,53 s, enquanto o makespan médio de TALON, em suas três versões, se situa entre 333,78 e 341,95 s com apenas duas CPUs. Isso mostra que, recebendo oito vezes menos CPUs do servidor de borda para cada usuário, TALON ainda assim entrega um desempenho equivalente e ligeiramente menor em média do que HOA.

Tabela 6.3: *Makespan* absoluto (em segundos) para TALON e HOA no cenário multiusuário.

	1 CPU	2 CPUs	4 CPUs	16 CPUs
HOA	385,95 ± 41,48	354,52 ± 36,49	349,05 ± 35,73	348,53 ± 35,68
TALON-0	370,60 ± 40,33	341,95 ± 36,72	334,12 ± 35,86	333,31 ± 35,78
TALON-N	366,27 ± 39,35	337,67 ± 35,91	329,77 ± 35,04	329,05 ± 34,96
TALON-N^2	361,53 ± 38,55	333,78 ± 35,48	326,37 ± 34,70	325,79 ± 34,65

6.3 Variando a Taxa de Transmissão de Dados

As taxas de transmissão podem ser reduzidas devido a condições do canal [30], uso de equipamentos legados e alta demanda em uma célula lotada na rede móvel. É importante que TALON seja capaz de encontrar oportunidades, através da estrutura do DAG, para realizar o *offloading* mesmo que os tempos de transmissão sejam elevados com a escassez de largura de banda disponível. Por outro lado, são esperadas taxas de transmissão mais elevadas nas redes 5G em diante. Portanto, a variação na transmissão de dados tem potencial para impactar o processo de tomada de decisão de *offloading*.

Para analisar o efeito da taxa de transmissão no *makespan* dos DAGs de aplicação, são considerados um número irrestrito de CPUs no dispositivo móvel e a disponibilidade de todas as 16 CPUs no servidor de borda. Essa disponibilidade plena de recursos de processamento é tomada aqui para que o foco seja o impacto das taxas de transmissão no *makespan*. As taxas de transmissão utilizadas nesse

cenário são de 10 Mbps, 20 Mbps e 100 Mbps. Essas taxas representam desde conexões bastante limitadas em termos de largura de banda, como em uma célula de rede móvel congestionada ou utilizando uma tecnologia de rede mais antiga, até conexões de 5G com taxas de dados mais altas [15].

Como um dos princípios de TALON é evitar as transmissões de dados mais longas em sua primeira fase, marcando para *offloading* o par de tarefas compondo tais arestas, o que anula o tempo de transmissão, TALON desempenha bem para diferentes taxas de transmissão. Conjuntamente, TALON prioriza o *offloading* de tarefas que formem arestas com tarefas já marcadas para *offloading*, explorando ainda mais o aspecto de evitar transmissões. Isso faz com que TALON geralmente migre o processamento do dispositivo móvel para a borda, e vice-versa, nas arestas com menor quantidade de *bytes* a transmitir.

A Figura 6.3 mostra o quão perto TALON chega da decisão ótima. Para uma taxa de 10 Mbps, TALON está 3,9% a 4,7% acima do *makespan* ótimo. Com 100 Mbps, esse desempenho permanece estável, ficando apenas 5,2% a 8,9% acima do *makespan* ótimo. Para HOA, por outro lado, a tendência é afastar-se cada vez mais do *makespan* ótimo com o aumento da taxa de transmissão, desviando-se em até 24,4% do *makespan* ideal com uma taxa de transmissão de 100 Mbps. Novamente, todas as três versões de TALON fornecem resultados estatisticamente melhores do que HOA e as soluções triviais para a redução do *makespan*.

Embora as soluções heurísticas se distanciem mais de seus respectivos ótimos com o aumento da taxa de transmissão, é importante que isso não seja tomado como uma degradação nos valores absolutos de *makespan*. De fato, o aumento da taxa de transmissão leva a *makespans* menores tanto para as heurísticas quanto para os ótimos. No entanto, a proporção com a qual isso acontece não é a mesma para ambas. Por essa razão, caso a redução do *makespan* ótimo aconteça em uma proporção maior do que aquela que acontece para *makespan* da heurística, o valor da diferença relativa de *makespan* aumenta.

Curiosamente, ao se levar em consideração a evolução das redes futuras e o advento do 6G em termos de largura de banda do canal, essa tendência por parte de HOA de se afastar do *makespan* ótimo pode levar a decisões de *offloading* ruins. Isso se torna pior pela busca de HOA por uma redução monotônica do *makespan* e, ao não cumprir esse quesito, optar por não fazer *offloading*, ao invés de aproveitar as taxas de transmissão maiores para que a migração do processamento entre dispositivo móvel e servidor de borda aconteça mais favoravelmente.

A respeito das soluções triviais, ambas continuam fornecendo os piores resultados em termos de *makespan*. A solução Sem *Offloading* piora com o aumento da taxa de transmissão, pois isso não gera benefício para essa decisão, mas beneficia a decisão ótima com a qual é comparada, de forma semelhante ao que acontece na Seção 6.1. O

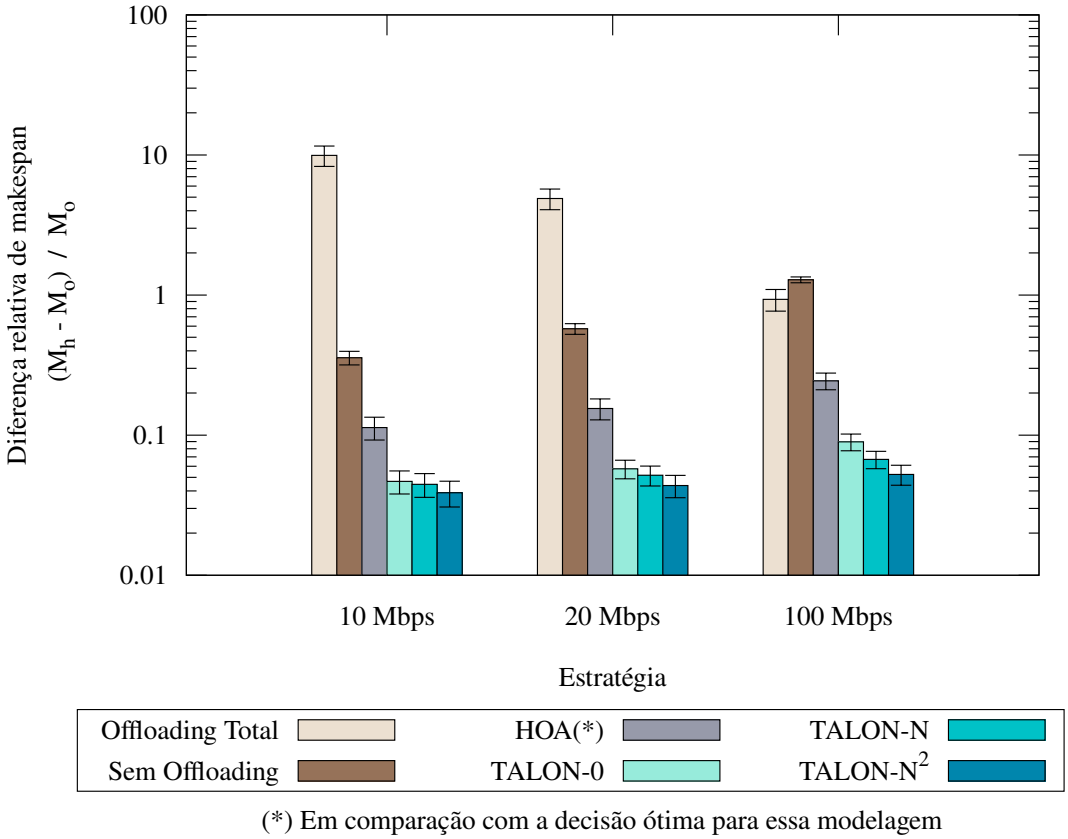


Figura 6.3: Diferença relativa de *makespan* devido à variação na taxa de transmissão de dados.

efeito contrário acontece para a solução *Offloading* Total, que em muito se beneficia do aumento da taxa de transmissão, mas seu desempenho ainda não se compara ao de TALON.

6.4 Comparação entre as Taxas de *Offloading*

Além de comprovar a eficiência de TALON para reduzir o *makespan* das aplicações, também é importante saber como são as decisões de *offloading* geradas pela mesma. Isso é importante para identificar se existe uma tendência a onerar mais o servidor de borda ou o dispositivo móvel. No primeiro caso, isso pode representar uma sobrecarga do servidor de borda, principalmente se múltiplos usuários possuírem a mesma tendência. No segundo, é possível que haja um baixo aproveitamento dos recursos de *hardware* disponíveis no servidor de borda, podendo levar a um problema de subutilização. Para isso, define-se o conceito de taxa de *offloading*, que é a a relação entre número de tarefas *offloaded* e o número total de tarefas elegíveis para *offloading* no DAG.

São selecionados três dos cenários utilizados nas Seções 6.1 e 6.2, explorando os limites inferior e superior de número de CPUs no dispositivo móvel e no servidor de borda, para investigar como TALON se comporta em relação à taxa de *offloading*. Esses cenários podem ser vistos na Tabela 6.4 e também são usados nas Seções 6.5 e 6.6. Em particular, como as soluções triviais Sem *Offloading* e *Offloading* Total têm, por definição, taxas de *offloading* de 0% e 100%, elas são omitidas dos resultados.

A Figura 6.4 mostra as taxas de *offloading* de TALON e HOA, bem como de suas respectivas decisões ótimas. TALON é capaz de refletir a disponibilidade de CPUs em suas decisões de *offloading*. Por exemplo, o primeiro cenário é o mais restritivo por parte do dispositivo móvel enquanto tem abundância de recursos no servidor de borda. O contrário acontece para o terceiro caso. TALON realiza o *offloading* de mais tarefas no primeiro cenário do que no terceiro, desafogando o dispositivo que esteja com menor disponibilidade de CPUs. No segundo caso, como ambos os dispositivos têm CPUs o bastante, as taxas de *offloading* de TALON se situam entre os dois extremos.

Mesmo com o aumento correspondente da taxa de *offloading* no primeiro cenário, os valores da taxa permanecem relativamente próximos entre si, sugerindo que TALON não sobrecarrega o servidor de borda mesmo com a escassez de recursos no dispositivo móvel. Mesmo para o primeiro cenário, TALON realiza o *offloading* de, no máximo, 42% das tarefas. Já para o terceiro cenário, as limitações do servidor de borda levam TALON a fazer *offloading* de apenas 29% a 30% das tarefas, ajustando-se às condições do dispositivo móvel e do servidor de borda.

Conforme mencionado anteriormente, HOA sempre considera que os dispositivos móveis possuem um número irrestrito de CPUs para executar qualquer DAG de aplicação, portanto HOA não possui resultados no primeiro cenário. Além disso, HOA não explora tantas oportunidades de *offloading* quantas forem possíveis e realiza o *offloading* aproximadamente de apenas 15% das tarefas, o que sugere uma possível justificativa para que seus valores de *makespan* sejam superiores em média aos

Tabela 6.4: Os três cenários considerados na comparação das decisões de *offloading*.

Nome	Número de CPUs (móvel)	Número de CPUs (borda)	Taxa de transmissão
Cenário 1	2	16	20 Mbps
Cenário 2	Irrestrito	16	20 Mbps
Cenário 3	Irrestrito	1	20 Mbps

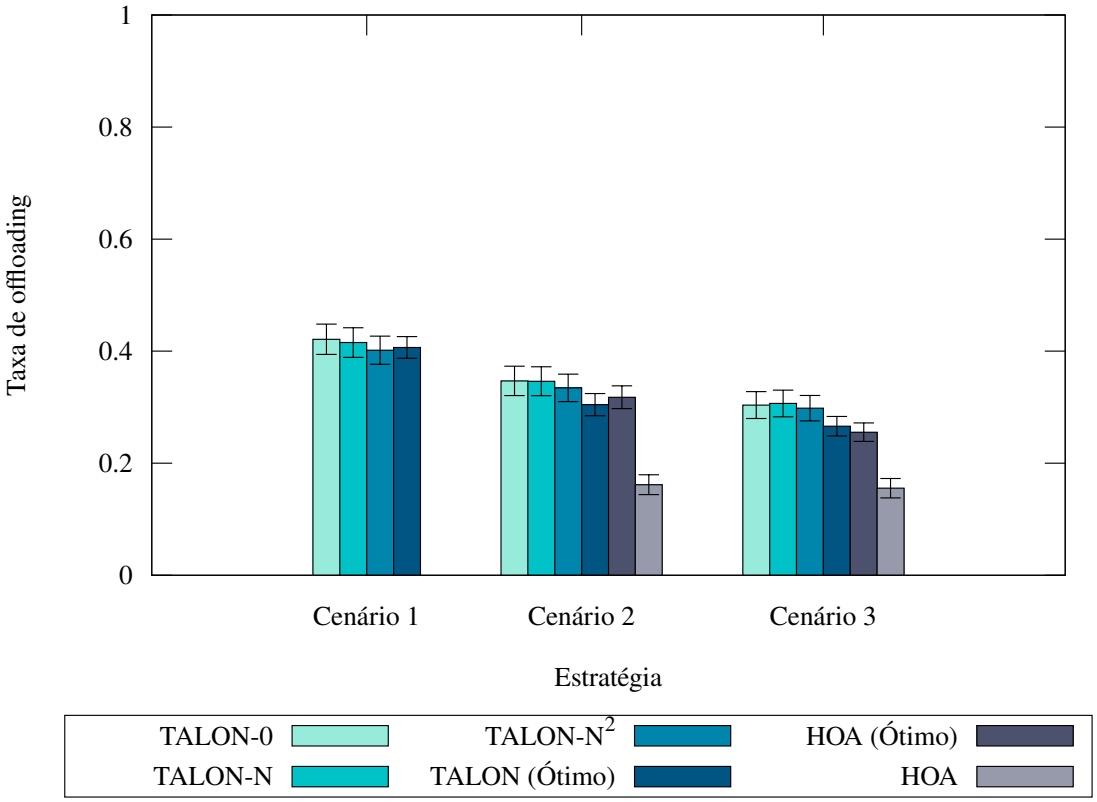


Figura 6.4: Taxas de *offloading* para os cenários na Tabela 6.4.

obtidos por TALON e também indique que HOA recorra à decisão Sem *Offloading* desnecessariamente.

Embora as modelagens de sistema utilizadas nesta tese e em Guo *et al.* [46] sejam diferentes, bem como seus cálculos de *makespan*, o que significa que as decisões ótimas possam ser diferentes entre as duas, a Figura 6.4 mostra que a diferença as taxas de *offloading* ótimas das duas modelagens estão separadas por apenas 0,01. Dito isso, outro ponto importante aqui é que a diferença entre a taxa de *offloading* entre HOA e a de sua decisão ótima (HOA Ótimo) é de 0,1 a 0,15, em nítido contraste com o que TALON consegue em relação a TALON Ótimo, que é, no máximo, 0,04.

Numericamente, TALON se aproxima muito mais de seu valor ótimo do que HOA, porém, não basta apenas saber quantas tarefas são marcadas para *offloading*, mas também quais delas, uma vez que reconhecer a heterogeneidade das tarefas compondo um DAG e, consequentemente, que algumas contribuem mais do que outras para a redução do *makespan*, também é um dos princípios considerados no desenvolvimento de TALON, conforme visto no Capítulo 4.

6.5 Análise de Similaridade

O grau de semelhança para verificar como as decisões tomadas pelas heurísticas se relacionam, tarefa por tarefa, com aquelas tomadas pelas suas respectivas decisões ótimas é chamado, no contexto desta tese, de similaridade. Assim, um índice de similaridade é formulado com base na distância de Hamming normalizada, dada por

$$S_A = 1 - \frac{H_A}{N_A}, \tag{6.3}$$

onde S_A é o índice de similaridade para o DAG de aplicação A, H_A é a distância de Hamming entre uma determinada decisão de *offloading* e a decisão ótima para A, e N_A é o número de tarefas elegíveis para *offloading* em A. Nesse contexto, a distância de Hamming é o número de modificações que devem ser feitas nas marcações de *offloading* de uma decisão de *offloading* para que elas se igualem às marcações de *offloading* da decisão ótima. Isto é, H_A é o número de tarefas para as quais um algoritmo realiza uma marcação de *offloading* errada em relação à decisão ótima. Naturalmente, quando $S_A = 1$, a heurística é capaz de acertar todas as marcações de *offloading* feitas pela decisão ótima para A.

Como cada DAG de aplicação A tem um valor diferente de N_A, a distância de Hamming é normalizada em relação a N_A. Por exemplo, uma distância de Hamming de 3 indica três marcações errôneas em relação ao ótimo, mas se isso ocorre em um DAG com 100 tarefas, o impacto desse erro não é o mesmo de errar três das marcações em um DAG com 5 tarefas.

Conforme mostrado na Figura 6.5, todas as versões de TALON atingem um índice de similaridade superior a 80% em todos os casos, chegando até 87%. Novamente, ambas as soluções triviais têm desempenho significativamente pior do que HOA ou TALON. Mesmo que a solução Sem *Offloading* apresente índices de similaridade que cheguem a 60% ou 70%, isso apenas indica que a decisão ótima realiza *offloading* precisamente e para poucas tarefas, haja vista as taxas de *offloading* ótimas por volta de 30% e 40% na Seção 6.4. Individualmente, o indíce de similaridade é uma métrica pouco reveladora, por isso deve ser analisado em conjunto com as métricas apresentadas anteriormente, que mostram o desempenho ruim em termos de redução de *makespan* das soluções triviais, por exemplo.

Todas as versões de TALON superam HOA no Cenário 2, enquanto TALON e HOA estão empatadas no Cenário 3 quanto à similaridade de cada uma com as respectivas decisões ótimas. Isso possivelmente se deve à natureza muito restritiva desse cenário, no qual apenas uma CPU está disponível no servidor de borda, uma vez que HOA, por padrão, deixa de aproveitar oportunidades de *offloading* enquanto TALON se adapta em cada cenário para não sobrecarregar o servidor de borda. Por-

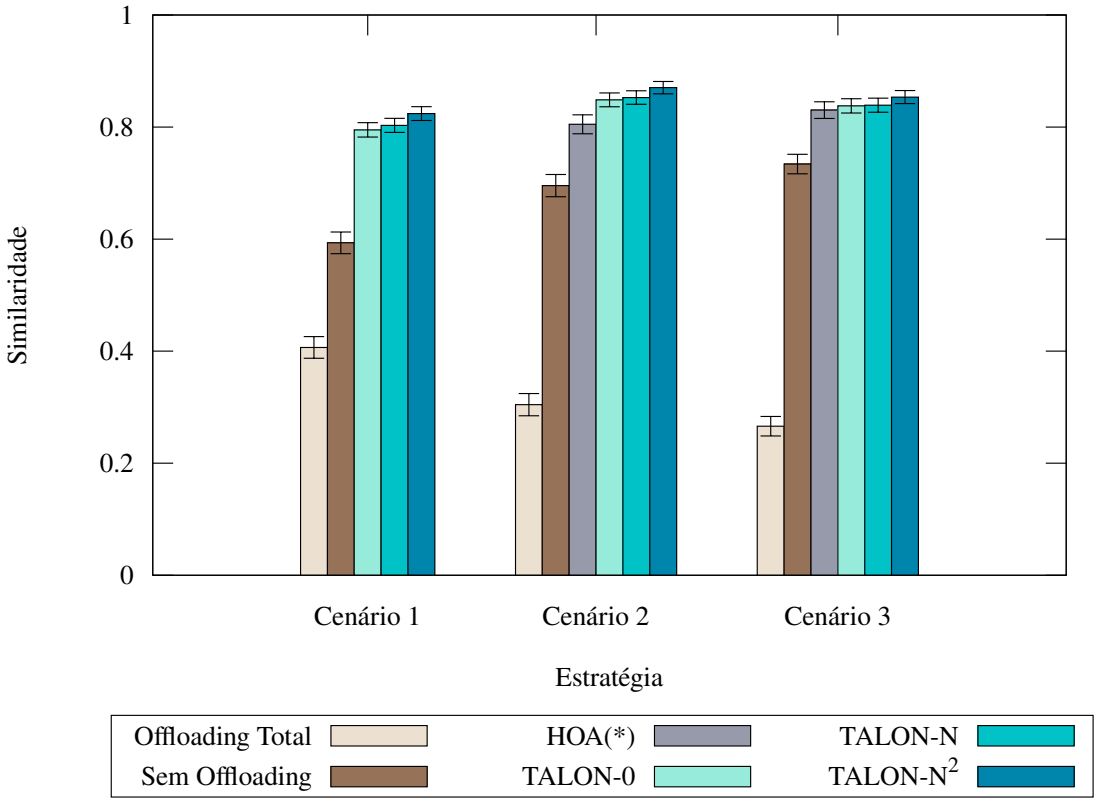


Figura 6.5: Similaridade entre as decisões de *offloading* das heurísticas e as de suas decisões ótimas correspondentes.

tanto, em média, tanto TALON quanto HOA marcam poucas tarefas para *offloading*, bem como suas respectivas decisões ótimas.

6.6 Análise do Paralelismo de Tarefas

Tanto o *makespan*, quanto a taxa de *offloading* e o índice de similaridade são úteis para avaliar uma decisão de *offloading*. No entanto, essas métricas não mostram uma noção temporal que reflita a execução da aplicação do início ao fim. Por exemplo, um DAG de aplicação com 30 tarefas elegíveis para *offloading* e uma taxa de *offloading* de 90% podem causar a impressão de que a decisão de *offloading* em questão é muito onerosa para servidor de borda. No entanto, se essas 30 tarefas formarem uma topologia sequencial (como um caminho), não mais do que uma única CPU do servidor de borda é utilizada a qualquer momento, deixando as restantes livres para executar outras funções.

A última métrica avaliada neste Capítulo é, portanto, o máximo paralelismo de processamento de tarefas alcançado ao longo da execução do DAG, ou simplesmente paralelismo de tarefas. Pretende-se investigar como cada decisão de *offloading* ex-

Tabela 6.5: Paralelismo de tarefas no dispositivo móvel.

	Cenário 1	Cenário 2	Cenário 3
Offloading Total	1,0 ± 0,0	1,0 ± 0,0	1,0 ± 0,0
Sem *Offloading*	8,24 ± 0,26	8,24 ± 0,26	8,24 ± 0,26
HOA	-	7,66 ± 0,28	7,69 ± 0,28
HOA (Ótimo)	-	6,65 ± 0,28	7,06 ± 0,27
TALON-0	5,72 ± 0,29	6,49 ± 0,31	6,79 ± 0,30
TALON-N	5,78 ± 0,28	6,50 ± 0,31	6,76 ± 0,30
TALON-N^2	5,89 ± 0,28	6,61 ± 0,30	6,86 ± 0,29
TALON (Ótimo)	5,62 ± 0,24	6,77 ± 0,28	6,99 ± 0,27

Tabela 6.6: Paralelismo de tarefas no servidor de borda.

	Cenário 1	Cenário 2	Cenário 3
Offloading Total	3,68 ± 0,23	3,68 ± 0,23	3,80 ± 0,23
Sem *Offloading*	0,0 ± 0,0	0,0 ± 0,0	0,0 ± 0,0
HOA	-	0,90 ± 0,10	0,40 ± 0,04
HOA (Ótimo)	-	1,88 ± 0,14	0,75 ± 0,03
TALON-0	1,94 ± 0,16	1,62 ± 0,15	1,43 ± 0,13
TALON-N	1,95 ± 0,16	1,62 ± 0,15	1,43 ± 0,13
TALON-N^2	1,92 ± 0,16	1,60 ± 0,15	1,40 ± 0,12
TALON (Ótimo)	2,15 ± 0,13	1,70 ± 0,13	1,41 ± 0,10

plora a utilização das CPUs disponíveis, tanto no dispositivo móvel quanto no servidor de borda. Esse paralelismo é dado pelo maior número de tarefas do DAG sendo executadas concomitantemente em cada dispositivo ao longo da execução da aplicação. As médias dos valores de paralelismo para os 1322 DAGs de aplicação são dadas pelas Tabelas 6.5 e 6.6.

Por definição, a solução *Offloading* Total sempre tem paralelismo de tarefas igual a 1 no dispositivo móvel, pois apenas as tarefas inicial e final são executadas localmente. Da mesma forma, a solução Sem *Offloading* sempre tem paralelismo zero no servidor de borda, pois nenhuma tarefa é *offloaded*. O paralelismo de tarefas de HOA confirma sua tendência de depender mais do dispositivo móvel do que do servidor de borda, tendo, em média, mais de 7 tarefas sendo executadas ao mesmo tempo no dispositivo móvel e apenas 0,90 e 0,40 no servidor de borda para os cenários 2 e 3, respectivamente. Quanto mais o paralelismo se aproxima de 0, mais isso indica que HOA opta por não realizar *offloading*, recorrendo à execução local para uma parcela significativa de DAGs de aplicação.

Com relação a TALON, o paralelismo de tarefas está notavelmente bem alinhado com o da decisão ótima e, em comparação com HOA, há uma diferença menor

entre o paralelismo de tarefas no dispositivo móvel e no servidor, sugerindo uma distribuição mais uniforme do processamento das tarefas entre ambos. Vale lembrar que a estrutura dos DAGs propriamente dita permite níveis altos de paralelismo, isto é, não são tratados DAGs unicamente sequenciais, isso é comprovado na Tabela 6.5, cujos valores de paralelismo podem chegar a 8 tarefas concomitantes.

Uma descoberta importante aqui é que, mesmo com taxas de *offloading* em torno de 40%, conforme a Seção 6.4, TALON é muito eficiente em atingir valores baixos de *makespan* e não sobrecarregar o servidor de borda, visto que o paralelismo de tarefas que TALON atinge não passa de cerca de 2, em média, para todos os cenários.

Capítulo 7

Conclusão e Trabalhos Futuros

Com o crescimento das redes móveis e o surgimento de novas aplicações, como telemedicina, jogos *mobile* e *online*, AI, *streaming* de vídeo ao vivo, VR e AR, novos paradigmas de rede devem ser definidos para sustentar os requisitos dessas aplicações. Tais aplicações podem ser sensíveis a latência, ou fazer uso intensivo de memória, processamento e rede. A computação de borda e, mais especificamente, o *offloading* computacional da execução dessas aplicações para a borda surgem como uma possível solução capaz de atender a esses critérios. São utilizados os recursos tanto do dispositivo móvel quanto do servidor de borda para particionar a aplicação e executá-la por partes com o objetivo de reduzir, por exemplo, o tempo de conclusão (*makespan*) da aplicação.

Aplicações de rede, ao invés de apresentar uma estrutura monolítica, são frequentemente compostas por múltiplas tarefas dependentes e suas transmissões de dados, que se referem aos dados gerados e passados de uma tarefa para suas sucessoras. Isso leva à estruturação de aplicações de rede como DAGs, através do particionamento das aplicações em tarefas, para as quais uma decisão de *offloading* é tomada. Portanto, essas tarefas podem ser executadas localmente em um dispositivo móvel ou *offloaded* para um servidor de borda. Este é o problema PODAG proposto nesta tese.

Tomar uma decisão sobre quais tarefas devem ser *offloaded* para atingir o *makespan* mínimo é um problema complexo. Portanto, esta tese propõe a heurística TALON para gerar um conjunto de decisões de *offloading* em duas etapas, combinando uma fase gananciosa com uma fase que gera todas as decisões possíveis para as tarefas ainda não marcadas para *offloading*. Com isso, TALON resolve de forma eficiente o problema PODAG para aplicações genéricas em computação de borda, um desafio antes inexplorado na literatura, a qual conta apenas com soluções simplificadas e cenários específicos.

TALON é comparada com a solução da principal referência da literatura (HOA) e duas outras soluções triviais, obtendo um *makespan* reduzido para aplicações realis-

tas, que são extraídas de um *data set* de um grande *cluster* computacional. TALON supera todas as soluções em diferentes cenários, os quais incluem a variação do número de CPUs disponíveis tanto no dispositivo móvel quanto no servidor de borda e a variação da taxa de transmissão utilizada no canal de comunicação sem fio. Em números, TALON está consistentemente 3,9% a 8,9% acima do *makespan* mínimo da decisão ótima em todos os cenários de teste. Em especial, para o cenário multiusuário, o desempenho de TALON, quando estão disponíveis apenas duas CPUs no servidor de borda, já é comparável (sendo ligeiramente melhor em termos de média do *makespan*) ao desempenho obtido por HOA, mesmo com 16 CPUs disponíveis para HOA.

TALON também é avaliada quanto às suas decisões de *offloading* e seu impacto no servidor de borda. TALON realiza *offloading* de, no máximo, 42% das tarefas, reduzindo esse número quando os recursos de CPU no servidor de borda se tornam mais escassos, e está muito próxima em número de tarefas *offloaded* da decisão ótima. De fato, além do aspecto quantitativo, as decisões tomadas por TALON e pela decisão ótima tendem a ser semelhantes, com valores de similaridade sempre maiores que 80%, mostrando que são tomadas as mesmas decisões, tarefa por tarefa. Quanto à sobrecarga no servidor de borda, TALON não o onera demasiadamente. Em média, o paralelismo de tarefas que TALON atinge, ou seja, o número de CPUs sendo utilizadas concomitantemente no servidor de borda para a execução do DAG de aplicação, não passa de aproximadamente 2 e está bem alinhado com o paralelismo associado à decisão ótima.

Outra conclusão desta tese é a demonstração de que a política *one-climb* não deve ser uma imposição para encontrar uma decisão de *offloading* eficiente para DAGs de aplicação gerais. TALON não se limita a essa política e, na verdade, para o conjunto de 1322 DAGs, cerca de 17% das decisões ótimas de *offloading* não obedecem a essa política.

Dados os resultados, fica claro o benefício de não limitar TALON à política *one-climb* ou à redução monotônica do *makespan*. Inclusive, buscar realizar o *offloading* das tarefas aos pares, ao invés de individualmente, também tem um resultado positivo para reduzir o *makespan*. Mesmo que TALON não calcule, durante suas duas fases de geração de decisões de *offloading*, o número de tarefas sendo executadas simultaneamente e o número de transmissões sendo realizadas simultaneamente, esses fatores são devidamente considerados durante o teste do conjunto de decisões de *offloading*. Além disso, TALON também tenta marcar para *offloading* as tarefas que mais contribuem para a redução do *makespan*. Essa contribuição é dada pela diferença entre os tempos de processamento no dispositivo móvel e no servidor de borda. Por fazer o *offloading* das tarefas principalmente aos pares, TALON também cancela os tempos mais altos de transmissão de dados. Todos esses aspectos

remetem às considerações fundamentais sobre as quais TALON está firmada, na forma como apresentada no Capítulo 4, destacando a importância de, na medida do possível, considerar todos os aspectos que possam influenciar no *makespan*.

As direções futuras de pesquisa incluem algumas oportunidades. A transmissão em meio sem fio é conhecida pela grande atenuação e pela aleatoriedade no canal, essa degradação impacta na intensidade do sinal transmitido e pode gerar perdas. Inclusive, o canal pode se tornar indisponível por um determinado tempo devido a ruído e desvanecimento. Aliar a tomada de decisão de *offloading* de TALON a um mecanismo novo para lidar com perdas é muito relevante para aplicações sensíveis a latência. De fato, o atraso adicional incorrido por cada retransmissão pode representar uma degradação que certas aplicações não podem tolerar. Além disso, também é possível que a retransmissão não ocorra tão logo haja a perda. Caso novas tarefas já estejam prontas para serem *offloaded*, é conveniente priorizar as retransmissões para diminuir o atraso adicional associado às mesmas antes de iniciar novas transmissões.

Além do *makespan*, outro aspecto essencial para aplicações móveis é o consumo de energia dos dispositivos de usuário, como *smartphones* e óculos de AR/VR. Decisões de *offloading* associadas a um *makespan* baixo, mas que levem a um processamento intensivo no dispositivo de usuário, podem levar a um alto consumo de energia. Um consumo elevado pode esgotar a bateria do dispositivo em pouco tempo ou levar a um aquecimento do *hardware*, prejudicando a QoE do usuário. Incorporar restrições de consumo de energia máximo ao problema PODAG, bem como limitar o número de CPUs em utilização no dispositivo de usuário, podem reduzir o consumo de energia enquanto se alcança um *makespan* baixo. Adicionalmente, medir o consumo de energia por tarefa e processar as de consumo elevado no servidor de borda ou considerar um mecanismo de redução de *duty cycle* no dispositivo de usuário também podem contribuir para a redução do consumo de energia.

Por fim, aplicações de execução em tempo real, nas quais haja recursão de tarefas, podem ser estruturadas como grafos cíclicos e formam um problema à parte do problema PODAG, com outros desafios. Essas aplicações incluem atividades interativas, como metaverso e jogos, cujas tarefas a serem executadas dependem de ações realizadas naquele exato momento pelos usuários finais. Como as informações do grafo não são conhecidas previamente, é necessário que TALON funcione de forma dinâmica, tomando a decisão de *offloading* ao longo da execução da aplicação. As incertezas quanto ao número de vezes que um ciclo é realizado ou qual a próxima tarefa a ser executada, caso uma tarefa possa acionar uma aresta de saída ou outra, são um fator a ser considerado. Assim, mecanismos de aprendizado de máquina podem ser implementados para prever a execução do grafo no longo prazo e tomar a decisão de *offloading* que seja provavelmente a mais benéfica.

Referências Bibliográficas

[1] AL-FUQAHA, A., GUIZANI, M., MOHAMMADI, M., et al. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", *IEEE Communications Surveys & Tutorials*, v. 17, pp. 2347–2376, 2015.

[2] ZHANG, H., LI, J., MOTEGI, K., et al. "IoT Framework of Wearable Device for Telemedicine Application". In: *2020 59th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pp. 1616–1621, 2020.

[3] ZHANG, T., CHIASSERINI, C. F., GIACCONE, P. "TAME: An Efficient Task Allocation Algorithm for Integrated Mobile Gaming", *IEEE Systems Journal*, v. 13, pp. 1546–1557, 2019.

[4] ZHANG, X., CHEN, H., ZHAO, Y., et al. "Improving Cloud Gaming Experience Through Mobile Edge Computing", *IEEE Wireless Communications*, v. 26, pp. 178–183, 2019.

[5] ORLOSKY, J., KIYOKAWA, K., TAKEMURA, H. "Virtual and augmented reality on the 5G highway", *Journal of Information Processing*, v. 25, pp. 133–141, 2017.

[6] ALBALAWI, U., JOSHI, S. "Secure and trusted telemedicine in Internet of Things IoT". In: *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp. 30–34, 2018.

[7] ELEMAM, E., BAHAA-ELDIN, A. M., SHAKER, N. H., et al. "A Secure MQTT Protocol, Telemedicine IoT Case Study". In: *2019 14th International Conference on Computer Engineering and Systems (ICCES)*, pp. 99–105, 2019.

[8] ACETO, G., PERSICO, V., PESCAPÉ, A. "A Survey on Information and Communication Technologies for Industry 4.0: State-of-the-Art, Taxonomies, Perspectives, and Challenges", *IEEE Communications Surveys Tutorials*, v. 21, pp. 3467–3501, 2019.

[9] HU, X., CHU, T. H. S., LEUNG, V. C. M., et al. "A Survey on Mobile Social Networks: Applications, Platforms, System Architectures, and Future Research Directions", *IEEE Communications Surveys & Tutorials*, v. 17, pp. 1557–1581, 2015.

[10] HUANG, Q., ANG, P., KNOWLES, P., et al. "SVE: Distributed Video Processing at Facebook Scale". In: *26th Symposium on Operating Systems Principles (SOSP)*, pp. 87–103, 2017.

[11] ALIYU, A., ABDULLAH, A. H., KAIWARTYA, O., et al. "Towards video streaming in IoT Environments: Vehicular communication perspective", *Computer Communications*, v. 118, pp. 93–119, 2018.

[12] YAQOOB, A., BI, T., MUNTEAN, G.-M. "A Survey on Adaptive 360° Video Streaming: Solutions, Challenges and Opportunities", *IEEE Communications Surveys & Tutorials*, v. 22, pp. 2801–2838, 2020.

[13] JEDARI, B., PREMSANKAR, G., ILLAHI, G., et al. "Video Caching, Analytics, and Delivery at the Wireless Edge: A Survey and Future Directions", *IEEE Communications Surveys & Tutorials*, v. 23, pp. 431–471, 2021.

[14] WANG, Y., SU, Z., ZHANG, N., et al. "A Survey on Metaverse: Fundamentals, Security, and Privacy", *IEEE Communications Surveys & Tutorials*, v. 25, pp. 319–352, 2023.

[15] PARVEZ, I., RAHMATI, A., GUVENC, I., et al. "A Survey on Low Latency Towards 5G: RAN, Core Network and Caching Solutions", *IEEE Communications Surveys & Tutorials*, v. 20, pp. 3098–3130, 2018.

[16] CHEN, Z. *An Application Platform for Wearable Cognitive Assistance*. Tese de Doutorado, Carnegie Mellon University, 2018.

[17] LIN, L., LIAO, X., JIN, H., et al. "Computation Offloading Toward Edge Computing", *Proceedings of the IEEE*, v. 107, pp. 1584–1607, 2019.

[18] SATYANARAYANAN, M. "The emergence of edge computing", *Computer*, v. 50, pp. 30–39, 2017.

[19] YICK, J., MUKHERJEE, B., GHOSAL, D. "Wireless sensor network survey", *Computer Networks*, v. 52, pp. 2292–2330, 2008.

[20] DINH, H. T., LEE, C., NIYATO, D., et al. "A survey of mobile cloud computing: Architecture, applications, and approaches", *Wireless Communications and Mobile Computing*, v. 13, pp. 1587–1611, 2013.

[21] FERNANDO, N., LOKE, S. W., RAHAYU, W. "Mobile cloud computing: A survey", *Future Generation Computer Systems*, v. 29, pp. 84–106, 2013.

[22] WEICHBROTH, P. "Usability of Mobile Applications: A Systematic Literature Study", *IEEE Access*, v. 8, pp. 55563–55577, 2020.

[23] WU, C., YANG, B., ZHU, W., et al. "Toward High Mobile GPU Performance Through Collaborative Workload Offloading", *IEEE Transactions on Parallel and Distributed Systems*, v. 29, pp. 435–449, 2018.

[24] SALAHT, F. A., DESPREZ, F., LEBRE, A. "An Overview of Service Placement Problem in Fog and Edge Computing", *ACM Computing Surveys*, v. 53, pp. 1–35, 2020.

[25] KUNTSCHKE, R., STEGMAIER, B., KEMPER, A., et al. "StreamGlobe: Processing and Sharing Data Streams in Grid-Based P2P Infrastructures". In: *Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 1259–1262, 2005.

[26] SADASHIV, N., KUMAR, S. "Cluster, grid and cloud computing: A detailed comparison". In: *2011 6th International Conference on Computer Science & Education (ICCSE)*, pp. 477–482, 2011.

[27] MA, X., ZHAO, Y., ZHANG, L., et al. "When mobile terminals meet the cloud: computation offloading as the bridge", *IEEE Network*, v. 27, pp. 28–33, 2013.

[28] BAROLLI, L., XHAFA, F. "JXTA-Overlay: A P2P Platform for Distributed, Collaborative, and Ubiquitous Computing", *IEEE Transactions on Industrial Electronics*, v. 58, pp. 2163–2172, 2010.

[29] SINGH, M., PRASANNA, V. K., ROLIM, J., et al. "Collaborative and Distributed Computation in Mesh-Like Wireless Sensor Arrays". In: *Personal Wireless Communications*, pp. 1–11, 2003.

[30] MAO, Y., YOU, C., ZHANG, J., et al. "A Survey on Mobile Edge Computing: The Communication Perspective", *IEEE Communications Surveys & Tutorials*, v. 19, pp. 2322–2358, 2017.

[31] CISCO. "Cisco annual Internet report (2018–2023) white paper". 2020. Disponível em: <`https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`>. Online. White Paper.

[32] ROMAN, R., LOPEZ, J., MAMBO, M. "Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges", *Future Generation Computer Systems*, v. 78, pp. 680–698, 2018.

[33] SHI, W., CAO, J., ZHANG, Q., et al. "Edge Computing: Vision and Challenges", *IEEE Internet of Things Journal*, v. 3, pp. 637–646, 2016.

[34] JIANG, X., YU, F. R., SONG, T., et al. "A survey on Multi-Access Edge Computing Applied to Video Streaming: Some Research Issues and Challenges", *IEEE Communications Surveys & Tutorials*, v. 23, pp. 871–903, 2021.

[35] FENG, C., HAN, P., ZHANG, X., et al. "Computation offloading in mobile edge computing networks: A survey", *Journal of Network and Computer Applications*, v. 202, pp. 103366, 2022.

[36] CHIANG, M., ZHANG, T. "Fog and IoT: An Overview of Research Opportunities", *IEEE Internet of Things Journal*, v. 3, pp. 854–864, 2016.

[37] WANG, B., WANG, C., HUANG, W., et al. "A survey and taxonomy on task offloading for edge-cloud computing", *IEEE Access*, v. 8, pp. 186080–186101, 2020.

[38] XU, Z., LIANG, W., JIA, M., et al. "Task Offloading with Network Function Requirements in a Mobile Edge-Cloud Network", *IEEE Transactions on Mobile Computing*, v. 18, pp. 2672–2685, 2019.

[39] JIA, M., CAO, J., YANG, L. "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing". In: *33rd IEEE International Conference on Computer Communications (INFOCOM) Workshop on Mobile Cloud Computing*, pp. 352–357, 2014.

[40] ZANNI, A., YU, S., BELLAVISTA, P., et al. "Automated Selection of Offloadable Tasks for Mobile Computation Offloading in Edge Computing". In: *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 388–392, 2017.

[41] LIN, H., ZEADALLY, S., CHEN, Z., et al. "A survey on computation offloading modeling for edge computing", *Journal of Network and Computer Applications*, v. 169, pp. 102781, 2020.

[42] GUO, K., YANG, M., ZHANG, Y. "Computation offloading over a shared communication channel for mobile cloud computing". In: *IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, 2018.

[43] MAHN, T., BECKER, D., AL-SHATRI, H., et al. "A Distributed Algorithm for Multi-Stage Computation Offloading". In: *7th IEEE International Conference on Cloud Networking (CLOUDNET)*, pp. 47–52, 2018.

[44] MAHMOODI, S. E., SUBBALAKSHMI, K., UMA, R. N. *Spectrum-Aware Mobile Computing*. Cham, Switzerland, Springer, 2019.

[45] WANG, Y., SHENG, M., WANG, X., et al. "Mobile-Edge Computing: Partial Computation Offloading Using Dynamic Voltage Scaling", *IEEE Transactions on Communications*, v. 64, pp. 4268–4282, 2016.

[46] GUO, K., YANG, M., ZHANG, Y., et al. "Efficient resource assignment in mobile edge computing: A dynamic congestion-aware offloading approach", *Journal of Network and Computer Applications*, v. 134, pp. 40–51, 2019.

[47] ZHANG, W., WEN, Y., WU, D. O. "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing". In: *32nd IEEE International Conference on Computer Communications (INFOCOM)*, pp. 190–194, 2013.

[48] DUAN, Y., WU, J. "Computation Offloading Scheduling for Deep Neural Network Inference in Mobile Computing". In: *29th IEEE/ACM International Symposium on Quality of Service (IWQoS)*, pp. 1–10, 2021.

[49] YANG, T., CHAI, R., ZHANG, L., et al. "Worst Case Latency Optimization-based Joint Computation Offloading and Scheduling for Interdependent Subtasks". In: *12th International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1010–1015, 2020.

[50] SHU, G., ZHENG, X., XU, H., et al. "Cloudlet-Assisted Heuristic Offloading for Mobile Interactive Applications". In: *5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pp. 66–73, 2017.

[51] ZHANG, Y., LIU, H., JIAO, L., et al. "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing". In: *1st IEEE International Conference on Cloud Networking (CLOUDNET)*, pp. 80–86, 2012.

[52] YANG, S., BEI, X., ZHANG, Y., et al. "Application Offloading Based on R-OSGi in Mobile Cloud Computing". In: *4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pp. 46–52, 2016.

[53] AN, X., FAN, R., HU, H., et al. "Joint Task Offloading and Resource Allocation for IoT Edge Computing With Sequential Task Dependency", *IEEE Internet of Things Journal*, v. 9, pp. 16546–16561, 2022.

[54] DE QUEIROZ, G. F. C., DE REZENDE, J. F., BARBOSA, V. C. "A flexible algorithm to offload DAG applications for edge computing", *Journal of Network and Computer Applications*, v. 222, pp. 1–13, 2023.

[55] TANAKA, H., YOSHIDA, M., MORI, K., et al. "Multi-access Edge Computing: A Survey", *Journal of Information Processing*, v. 26, pp. 87–97, 2018.

[56] ABBAS, N., ZHANG, Y., TAHERKORDI, A., et al. "Mobile Edge Computing: A Survey", *IEEE Internet of Things Journal*, v. 5, pp. 450–465, 2018.

[57] ETSI. "Mobile-Edge Computing — Introductory Technical White Paper". 2014. Disponível em: <https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf>. Online. White Paper.

[58] ETSI. "Developing Software for Multi-Access Edge Computing". 2017. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp20_MEC_SoftwareDevelopment_FINAL.pdf>. Online. White Paper.

[59] ETSI. "Cloud RAN and MEC: A Perfect Pairing". 2018. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp23_MEC_and_CRAN_ed1_FINAL.pdf>. Online. White Paper.

[60] ETSI. "MEC Deployments in 4G and Evolution Towards 5G". 2018. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp24_MEC_deployment_in_4G_5G_FINAL.pdf>. Online. White Paper.

[61] ETSI. "MEC in 5G networks". 2018. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf>. Online. White Paper.

[62] ETSI. "MEC in an Enterprise Setting: A Solution Outline". 2018. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp30_MEC_Enterprise_FINAL.pdf>. Online. White Paper.

[63] ETSI. "Harmonizing standards for edge computing - A synergized architecture leveraging ETSI ISG MEC and 3GPP specifications". 2020. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/

ETSI_wp36_Harmonizing-standards-for-edge-computing.pdf>. Online. White Paper.

[64] ETSI. "Enhanced DNS Support towards Distributed MEC Environment". 2020. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/etsi-wp39-Enhanced-DNS-Support-towards-Distributed-MEC-Environment.pdf>. Online. White Paper.

[65] ETSI. "MEC federation: deployment considerations". 2022. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/ETSI_WP_49_MEC-Federation-Deployment-considerations.pdf>. Online. White Paper.

[66] ETSI. "MEC security: Status of standards support and future evolutions". 2022. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/ETSI-WP-46-2nd-Ed-MEC-security.pdf>. Online. White Paper.

[67] ETSI. "MEC support towards Edge Native Design". 2023. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/ETSI-WP55-MEC_support_towards_Edge_native.pdf>. Online. White Paper.

[68] ETSI. "Enabling Multi-access Edge Computing in Internet-of-Things: how to deploy ETSI MEC and oneM2M". 2023. Disponível em: <https://www.etsi.org/images/files/ETSIWhitePapers/ETSI-WP59-Enabling-Multi-access-Edge-Computing-in-iot.pdf>. Online. White Paper.

[69] YI, S., LI, C., LI, Q. "A Survey of Fog Computing: Concepts, Applications and Issues". In: *Workshop on Mobile Big Data (Mobidata)*, pp. 37–42, 2015.

[70] BECK, M. T., WERNER, M., FELD, S., et al. "Mobile Edge Computing: A Taxonomy". In: *AFIN 2014, The Sixth International Conference on Advances in Future Internet*, pp. 48–54, 2014.

[71] EUSUFZAI, F., HAQ, T., CHOWDHURY, S., et al. "An Overview of Cognitive Internet of Things". In: *Secure Edge Computing: Applications, Techniques and Challenges*, CRC Press, cap. 5, pp. 65–82, Boca Raton, United States, 2022.

[72] SAPIENZA, M., GUARDO, E., CAVALLO, M., et al. "Solving Critical Events through Mobile Edge Computing: An Approach for Smart Cities". In: *2nd IEEE Workshop on Sensors and Smart Cities (SSC 2016)*, pp. 287–291, 2016.

[73] SANGAMITHRA, A., MARY, T. M., CLINTON, G. "Overview of Edge Computing and Its Exploring Characteristics". In: *Cases on Edge Computing and Analytics*, IGI Global, cap. 4, pp. 73–94, Hershey, United States, 2021.

[74] DOS SANTOS, G. B., TRINTA, F. A. M., REGO, P. A. L. "Impactos do Offloading de Processamento no Tempo de Execução e Consumo Energético de Dispositivos Móveis". In: *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pp. 1271–1284, 2018.

[75] SATYANARAYANAN, M., BAHL, P., CACERES, R., et al. "The Case for VM-Based Cloudlets in Mobile Computing", *IEEE Pervasive Computing*, v. 8, pp. 14–23, 2009.

[76] KITANOV, S., JANEVSKI, T. "State of the art: Fog computing for 5G networks". In: *2016 24th Telecommunications Forum (TELFOR)*, pp. 86–89, 2016.

[77] VAQUERO, L. M., RODERO-MERINO, L. "Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing", *ACM SIGCOMM Computer Communication Review*, v. 44, pp. 27–32, 2014.

[78] BONOMI, F., MILITO, R., ZHU, J., et al. "Fog Computing and Its Role in the Internet of Things". In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, pp. 13–16, 2012.

[79] DOLUI, K., DATTA, S. K. "Comparison of Edge Computing Implementations: Fog Computing, Cloudlet and Mobile Edge Computing". In: *2017 Global Internet of Things Summit (GIoTS)*, pp. 19–24, 2017.

[80] AI, Y., PENG, M., ZHANG, K. "Edge computing technologies for Internet of Things: a primer", *Digital Communications and Networks*, v. 4, pp. 77–86, 2018.

[81] YOUSEFPOUR, A., FUNG, C., NGUYEN, T., et al. "All one needs to know about fog computing and related edge computing paradigms: A complete survey", *Journal of Systems Architecture*, v. 98, pp. 289–330, 2019.

[82] ELAZHARY, H. "Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions", *Journal of network and computer applications*, v. 128, pp. 105–140, 2019.

[83] REN, J., ZHANG, D., HE, S., et al. "A Survey on End-Edge-Cloud Orchestrated Network Computing Paradigms: Transparent Computing, Mobile Edge Computing, Fog Computing, and Cloudlet", *ACM Computing Surveys*, v. 52, pp. 1–36, 2019.

[84] HAQUE, M. E., TARIQ, F., KHANDAKER, M. R. A., et al. "A Survey of Scheduling in 5G URLLC and Outlook for Emerging 6G Systems", *IEEE Access*, v. 11, pp. 34372–34396, 2023.

[85] LIU, Y., CLERCKX, B., POPOVSKI, P. "Network Slicing for eMBB, URLLC, and mMTC: An Uplink Rate-Splitting Multiple Access Approach", *IEEE Transactions on Wireless Communications*, Early Access, pp. 1–14, 2023.

[86] KHAN, B. S., JANGSHER, S., AHMED, A., et al. "URLLC and eMBB in 5G Industrial IoT: A Survey", *IEEE Open Journal of the Communications Society*, v. 3, pp. 1134–1163, 2022.

[87] BROWN, G. "Ultra-Reliable Low-Latency 5G for Industrial Automation". Heavy Reading, Qualcomm Inc. 2022. Disponível em: <https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/ultra-reliable-low-latency-5g-for-industrial-automation.pdf>. Online. White Paper.

[88] CHIH-PING LI, JING JIANG, CHEN, W., et al. "5G Ultra-Reliable and Low-Latency Systems Design". In: *2017 European Conference on Networks and Communications (EuCNC)*, pp. 1–5, 2017.

[89] SCHULZ, P., MATTHE, M., KLESSIG, H., et al. "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture", *IEEE Communications Magazine*, v. 55, pp. 70–78, 2017.

[90] TAN, H., HAN, Z., LI, X., et al. "Online Job Dispatching and Scheduling in Edge-Clouds". In: *36th IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1548–1556, 2017.

[91] BROWN, G. "Exploring 5G New Radio: Use Cases, Capabilities & Timeline". Heavy Reading, Qualcomm Inc. 2016. Disponível em: <https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/heavyreading_whitepaper_-_exploring_5g_new_radio_use_cases_capabilities_and_timeine.pdf>. Online. White Paper.

[92] PARK, J., CHOI, J., KIM, S.-L., et al. "Enabling the Wireless Metaverse via Semantic Multiverse Communication". In: *2023 20th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pp. 85–90, 2023.

[93] KASGARI, A. T. Z., SAAD, W., MOZAFFARI, M., et al. "Experienced Deep Reinforcement Learning With Generative Adversarial Networks (GANs) for Model-Free Ultra Reliable Low Latency Communication", *IEEE Transactions on Communications*, v. 69, pp. 884–899, 2021.

[94] BANDI, A., ADAPA, P. V. S. R., KUCHI, Y. E. V. P. K. "The Power of Generative AI: A Review of Requirements, Models, Input-Output Formats, Evaluation Metrics, and Challenges", *Future Internet*, v. Special Issue: State-of-the-Art Future Internet Technology in USA 2022–2023, pp. 1–60, 2023.

[95] KARAPANTELAKIS, A., ALIZADEH, P., ALABASSI, A., et al. "Generative AI in mobile networks: a survey", *Annals of Telecommunications*, Early Access, pp. 1–19, 2023.

[96] WANG, L., JIAO, L., HE, T., et al. "Service Entity Placement for Social Virtual Reality Applications in Edge Computing". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 468–476, 2018.

[97] BERTÉNYI, B. "5G standards in 3GPP". 2017. Disponível em: <`https://www.3gpp.org/ftp/information/presentations/Presentations_2017/2017_03_Bertenyi_5G_3GPP.pdf`>. Online. 3GPP Presentations.

[98] WU, J., TAN, R., WANG, M. "Streaming High-Definition Real-Time Video to Mobile Devices with Partially Reliable Transfer", *IEEE Transactions on Mobile Computing*, v. 18, pp. 458–472, 2019.

[99] RICO, D., MERINO, P. "A Survey of End-to-End Solutions for Reliable Low-Latency Communications in 5G Networks", *IEEE Access*, v. 8, pp. 192808–192834, 2020.

[100] PHAM, Q.-V., FANG, F., HA, V. N., et al. "A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art", *IEEE Access*, v. 8, pp. 116974–117017, 2020.

[101] KATSALIS, K., PAPAIOANNOU, T. G., NIKAEIN, N., et al. "SLA-Driven VM Scheduling in Mobile Edge Computing". In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pp. 750–757, 2016.

[102] LEE, K., LEE, J., YI, Y., et al. "Mobile Data Offloading: How Much Can WiFi Deliver?" *IEEE/ACM Transactions on Networking*, v. 21, pp. 536–550, 2013.

[103] YU, Y. "Mobile edge computing towards 5G: Vision, recent progress, and open challenges", *China Communications*, v. 13, pp. 89–99, 2016.

[104] MIETTINEN, A. P., NURMINEN, J. K. "Energy efficiency of mobile clients in cloud computing", *HotCloud*, v. 10, pp. 19, 2010.

[105] VERBRAEKEN, J., WOLTING, M., KATZY, J., et al. "A Survey on Distributed Machine Learning", *ACM Computing Surveys*, v. 53, pp. 1–33, 2020.

[106] CAO, K., LIU, Y., MENG, G., et al. "An Overview on Edge Computing Research", *IEEE access*, v. 8, pp. 85714–85728, 2020.

[107] KHAN, A. U. R., OTHMAN, M., MADANI, S. A., et al. "A Survey of Mobile Cloud Computing Application Models", *IEEE Communications Surveys & Tutorials*, v. 16, pp. 393–413, 2014.

[108] MACH, P., BECVAR, Z. "Mobile Edge Computing: A Survey on Architecture and Computation Offloading", *IEEE Communications Surveys & Tutorials*, v. 19, pp. 1628–1656, 2017.

[109] SHAKARAMI, A., GHOBAEI-ARANI, M., SHAHIDINEJAD, A. "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective", *Computer Networks*, v. 182, pp. 107496, 2020.

[110] MELENDEZ, S., MCGARRY, M. P. "Computation offloading decisions for reducing completion time". In: *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 160–164, 2017.

[111] YOU, C., HUANG, K., CHAE, H., et al. "Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading", *IEEE Transactions on Wireless Communications*, v. 16, pp. 1397–1411, 2017.

[112] CHEN, X., JIAO, L., LI, W., et al. "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing", *IEEE/ACM Transactions on Networking*, v. 24, pp. 2795–2808, 2016.

[113] WANG, W., ZHOU, W. "Computational offloading with delay and capacity constraints in mobile edge". In: *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2017.

[114] ZHANG, G., ZHANG, W., CAO, Y., et al. "Energy-Delay Tradeoff for Dynamic Offloading in Mobile-Edge Computing System With Energy Harvesting Devices", *IEEE Transactions on Industrial Informatics*, v. 14, pp. 4642–4655, 2018.

[115] LIAO, Z., PENG, J., XIONG, B., et al. "Adaptive offloading in mobile-edge computing for ultra-dense cellular networks based on genetic algorithm", *Journal of Cloud Computing*, v. 10, pp. 15, 2021.

[116] KUANG, Z., LI, L., GAO, J., et al. "Partial Offloading Scheduling and Power Allocation for Mobile Edge Computing Systems", *IEEE Internet of Things Journal*, v. 6, pp. 6774–6785, 2019.

[117] HAN, P., LIU, Y., GUO, L. "Interference-Aware Online Multicomponent Service Placement in Edge Cloud Networks and its AI Application", *IEEE Internet of Things Journal*, v. 8, pp. 10557–10572, 2021.

[118] YAN, J., BI, S., ZHANG, Y. J., et al. "Optimal Task Offloading and Resource Allocation in Mobile-Edge Computing With Inter-User Task Dependency", *IEEE Transactions on Wireless Communications*, v. 19, pp. 235–250, 2020.

[119] NING, Z., DONG, P., KONG, X., et al. "A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things", *IEEE Internet of Things Journal*, v. 6, pp. 4804–4814, 2019.

[120] WANG, Z., ZHAO, D., NI, M., et al. "Collaborative Mobile Computation Offloading to Vehicle-Based Cloudlets", *IEEE Transactions on Vehicular Technology*, v. 70, pp. 768–781, 2021.

[121] MAZOUZI, H., ACHIR, N., BOUSSETTA, K. "Elastic Offloading of Multitasking Applications to Mobile Edge Computing". In: *22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pp. 307–314, 2019.

[122] TRAN, T. X., POMPILI, D. "Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks", *IEEE Transactions on Vehicular Technology*, v. 68, pp. 856–868, 2019.

[123] PENG, J., QIU, H., CAI, J., et al. "D2D-Assisted Multi-User Cooperative Partial Offloading, Transmission Scheduling and Computation Allocating for MEC", *IEEE Transactions on Wireless Communications*, v. 20, pp. 4858–4873, 2021.

[124] DOLBEAU, R. “Theoretical peak FLOPS per instruction set: A tutorial”, *The Journal of Supercomputing*, v. 74, pp. 1341–1377, 2018.

[125] SALIBA, D., IMAD, R., HOUCKE, S., et al. “WiFi Dimensioning to offload LTE in 5G Networks”. In: *9th IEEE Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 521–526, 2019.