Alocação Adaptativa de Tarefas na Névoa em Ambientes de Saúde Inteligente

Robertson Lima[1], **Anand Subramanian**[1], **Fernando Matos**[1]

[1]Centro de Informática - Universidade Federal da Paraíba (UFPB)

robertsonlima@ppgi.ci.ufpb.br,{anand,fernando}@ci.ufpb.br

Abstract. *Integrating IoT devices with Fog and Cloud Computing enables the deployment of smart pervasive healthcare environments to aid in disease prevention and diagnosis. However, management solutions are necessary to perform processing offload in such environments. Thus, this paper proposes an approach to optimize the allocation and reallocation of processing requests on machines in the Fog and Cloud using mathematical modeling. The aim is to minimize infrastructure's hardware utilization cost while ensuring the application requirements. Results show that for instances with up to 200 patients the optimization method generates near optimal solutions and the reoptimization generates optimal solutions in less than 7 and 2.8 seconds on average, respectively.*

Resumo. *A integração de IoT com computação em Névoa e Nuvem permite a implantação de ambientes de saúde pervasivos para auxiliar na prevenção e diagnóstico de doenças. Contudo, é necessário um gerenciamento mais inteligente para realizar o offload de tarefas computacionais nestes ambientes. Este trabalho propõe uma otimização para alocar e re-alocar requisições de processamento em máquinas na Névoa e Nuvem utilizando modelagem matemática. O objetivo é minimizar o custo de utilização da infraestrutura enquanto garante os requisitos das aplicações. Resultados apontam que a otimização gera soluções próximas do ótimo e a reotimização gera soluções ótimas em menos de 7 e 2,8 segundos em média, respectivamente, para instâncias com até 200 pacientes.*

1. Introdução

A integração da Computação em Nuvem com dispositivos IoT em ambientes de saúde pervasivos facilita o desenvolvimento e auxilia na implantação de diversas aplicações operacionais e médicas, como monitoramento remoto (telemedicina), armazenamento de registros e compartilhamento de imagens médicas [Wang and Jin 2019]. Nestes ambientes, a capacidade de processamento limitada e restrições energéticas que alguns dispositivos IoT possuem [Peralta et al. 2017] motivam o *offload* de tarefas computacionais para servidores na nuvem sem tais restrições [Li et al. 2019, Gross and Geyer 2020]. Contudo, muitas vezes a transmissão de dados ocorre para equipamentos geograficamente distantes, aumentando assim a latência de resposta, o que não é desejável em aplicações no contexto de *healthcare* que demandam por respostas em tempo real, como por exemplo monitoramento de sinais vitais. A Computação em Névoa minimiza este problema ao trazer poder de processamento para mais perto dos dispositivos, em equipamentos gerenciados localmente. Porém, esta abordagem aumenta a complexidade do gerenciamento da infraestrutura, pois é necessário tomar decisões de alocação das tarefas nos recursos

disponíveis considerando diversos fatores, como latência, eficiência energética e custo associado à utilização de recursos sob demanda na Névoa/Nuvem [Liu et al. 2019].

Uma alocação inteligente de tarefas deve sempre garantir que os requisitos das aplicações sejam garantidos ao realizar o *offload* de processamento. Além disso, eventos no ambiente, como chegadas de demandas de novas aplicações ou um servidor na Nuvem que está sendo subtilizado devido ao término de aplicações pode gerar a necessidade de realocações de tarefas. Neste contexto, muitos trabalhos propõe soluções de otimização para minimizar a latência de resposta às aplicações [Souza et al. 2016, Rezazadeh et al. 2019], minimizar o custo e aumentar a utilização dos servidores [Wen et al. 2017] ou para otimizar o consumo energético [N. Jayasena and Thisarasinghe 2019]. Apesar dos resultados positivos, os trabalhos não levam em consideração eventos que podem gerar uma re-adaptação das tarefas já alocadas.

Assim, este trabalho propõe uma abordagem dinâmica e adaptativa baseada em modelagem matemática para otimizar a alocação de tarefas de processamento em dispositivos na Névoa e Nuvem focando em reduzir o custo da infraestrutura. Através de uma reotimização, a abordagem também auxilia na adaptabilidade do sistema, permitindo que uma nova alocação seja calculada quando determinados eventos ocorrerem, como violação dos requisitos das aplicações ou subutilização de recursos. Foram considerados aspectos como capacidade computacional dos servidores, latência entre os dispositivos e a redução do impacto das migrações de tarefas de processamento durante a implantação da solução. Resultados mostram que a otimização gera soluções próximas do ótimo e a reotimização gera soluções ótimas em menos de 7 e 2,8 segundos em média, respectivamente, para instâncias com até 200 pacientes.

Este artigo está organizado como a seguir: a Seção 2 apresenta os trabalhos relacionados. A Seção 3 detalha os modelos matemáticos da abordagem proposta. A Seção 4 descreve os cenários de teste e discute os resultados e a Seção 5 conclui o artigo.

2. Trabalhos Relacionados

A alocação inteligente de recursos na Névoa e Nuvem é não determinístico e NP-difícil, mas de extrema importância na realização da orquestração desses recursos pois simplifica manutenções, reduzindo custos e aumentando a segurança e confiabilidade do sistema. Os trabalhos listados a seguir são soluções para a alocação inteligente de tarefas em ambientes da Névoa. [Souza et al. 2016, Zhao and Huang 2020] apresentam soluções para realizar o *offload* de tarefas na Névoa e Nuvem focando em reduzir o impacto da latência nas aplicações. [N. Jayasena and Thisarasinghe 2019, Wen et al. 2017] apresentam soluções baseadas metaheurísticas para decidir em quais dispositivos as tarefas de processamento deverão ser alocadas. [Rahabri and NICKRAY 2019, Rahbari and Nickray 2017] apresentam abordagens baseadas no problema da mochila (Knapsack Problem), sendo uma gulosa e outra metaheurística, e que foram submetidas a estudos de caso em ambientes *healthcare*. Finalmente, [Rezazadeh et al. 2019] apresenta um algoritmo exploratório para alocação de módulos na Névoa ou nuvem com objetivo de minimizar a latência.

O trabalho proposto se difere de alguns dos trabalhos relacionados pois assume que as tarefas executarão enquanto os dispositivos IoT estiverem em funcionamento, o que não pode ser pré-determinado para todas as aplicações. Os trabalhos relacionados também utilizam entradas pequenas na avaliação, o que não necessariamente representa

o comportamento real de um ambiente inteligente com uma ampla gama de dispositivos. Além disso, todos eles tratam apenas de novas demandas, sem considerar cenários em que os nós de processamento já estejam lidando com demandas antigas.

3. Otimização de alocação de tarefas

A abordagem de otimização proposta opera em um ambiente de saúde inteligente composto por 3 camadas (Figura 1). A camada IoT contém diversos tipos de dispositivos médicos (e.g. oxímetros, eletrocardiogramas, bombas de insulina) e de uso geral (e.g. câmeras de segurança, termômetros, detectores de vazamento de gás e poluição) que, para economizar energia, são capazes de realizar o *offload* do processamento dos dados coletados para nós presentes nas camadas superiores para executar as tarefas associadas ao dispositivo. Chamaremos de *tarefa* o conjunto de ações realizadas a partir de um dado coletado: armazenamento, processamento e qualquer outra ação disparada a partir deste processamento.

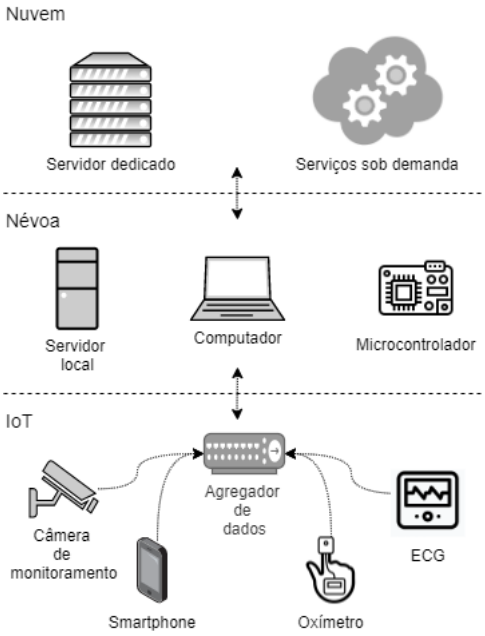


Figura 1. Ambiente de saúde inteligente

Por exemplo, a tarefa associada a um eletrocardiograma é ler os dados de frequência cardíaca, armazená-los e, em caso de anormalidade, alertar a equipe médica. A camada da Névoa é composta por equipamentos geograficamente próximos da camada IoT e que têm a capacidade de armazenar e processar dados, o que minimiza problemas de latência causados por transmissões de longa distância. Por fim, a camada da Nuvem contém servidores dedicados previamente alocados ou utilizando infraestrutura sob demanda. Chamaremos de nó todo dispositivo físico ou virtual presente tanto na Névoa quanto na Nuvem capaz de executar as tarefas dos dispositivos IoT e a solução de otimização. As tarefas podem ser alocadas e realocadas em qualquer nó da Névoa/Nuvem, desde que haja recursos computacionais suficientes.

3.1. Modelo Matemático

Seja T = {1, 2, ..., n} o conjunto de tarefas, D = {1, 2, ..., m} o conjunto de nós físicos ou virtuais disponíveis na Névoa ou Nuvem e R = {1, 2, ..., s } o conjunto de recursos computacionais em cada nó. A capacidade de um dado recurso $k \in R$ disponível no nó $j \in D$ é dado por C_{kj}, porém o modelo proposto assume que uma tarefa pode demandar uma quantidade diferente de um recurso dependendo da arquitetura do nó, de modo que r_{kij} representa quanto o recurso $k \in R$ é utilizado caso a tarefa $i \in T$ seja alocada ao nó $j \in D$. Cada tipo de tarefa tem diferentes valores para a latência máxima esperada para garantir o QoS, aqui dados por q_i, $i \in T$, que serão comparados com a latência entre o sensor associado à tarefa $i \in T$ e o nó $j \in D$, expressa por l_{ij}. Dado o custo w_j de utilizar o nó $j \in D$, o objetivo é minimizar o custo total de locação das tarefas.

A variável binária x_{ij} expressa se uma tarefa $i \in T$ está alocada à um nó $j \in D$, enquanto a variável y_j denota quando um nó $j \in D$ é utilizado na solução ou não. O modelo pode ser expresso da seguinte forma:

$$\min \sum_{j=1}^{m} w_j y_j \tag{1}$$

sujeito a:

$$l_{ij} x_{ij} \leq q_i, \qquad i \in T, j \in D \tag{2}$$

$$\sum_{i=1}^{n} r_{kij} x_{ij} \leq C_{kj} y_j, \qquad j \in D, y \in R \tag{3}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j \in D \tag{4}$$

$$x_{ij}, y_j \in \{0, 1\} \qquad i \in T, j \in R. \tag{5}$$

A Função Objetivo (FO) (1) minimiza o custo geral da infraestrutura, garantindo os requisitos de QoS. As restrições (2) garantem que a latência esperada (l_{ij}) entre o sensor e o nó que está executando a tarefa deve ser menor ou igual à latência máxima esperada (q_i) pelo tipo de tarefa. As restrições (3) asseguram, para cada tipo de recurso, que a soma total dos recursos consumidos pelas tarefas alocadas no nó deve ser menor que a capacidade total daquele nó. As restrições (4) determinam que uma tarefa i só pode estar alocada a um único nó. Finalmente, as restrições (5) definem o domínio das variáveis.

Como corporações podem adquirir hardware em lote, é provável que os nós na Névoa sejam máquinas com a mesma especificação. Já na Nuvem, as máquinas virtuais (Virtual Machine - VM) são precificadas baseado nos recursos selecionados pelo usuário. Uma VM com 2 CPUs e 4GB de RAM provavelmente custará o mesmo que duas VMs com 1 CPU e 2GB de RAM cada. Devido à esta simetria entre o hardware da Névoa e o preço por VM na Nuvem, múltiplas soluções serão obtidas com o mesmo valor na FO. Como cada tarefa é um *container*, devemos levar em consideração que existe um custo de migração associado, visto que para transferir uma tarefa entre nós precisamos instanciar um *container*, redirecionar o tráfego e transferir os dados em cache, fazer a migração e por fim parar a execução do *container* original. Esse processo gera uma sobrecarga na infraestrutura se for executado múltiplas vezes, especialmente se os recursos estiverem sendo super-utilizados pois dificultaria a migração paralela de *containers*.

3.2. Modelo de reotimização

Como o primeiro modelo matemático não leva em consideração a existência de tarefas já em execução, a alocação obtida nele pode acarretar em numa mudança completa na disposição das tarefas. Para contornar este problema, um segundo modelo matemático é proposto utilizando o resultado da FO (α) obtido na execução do primeiro modelo e a disposição corrente das tarefas (S). O objetivo deste segundo modelo é encontrar uma solução com um valor similar a α, mas que mantenha o máximo de tarefas possíveis nos nós em que já estão alocadas, ou seja, minimizar a quantidade de migrações.

Dado que x'_{ij} é a matriz de alocação das tarefas em execução na infraestrutura, definimos novamente x_{ij} como a alocação obtida por este segundo modelo. O modelo de reotimização pode ser escrito da seguinte maneira:

$$\min \sum_{(i,j)\in S} (x'_{ij} - x_{ij}) + \sum_{(i,j)\in \overline{S}} x_{ij} \tag{6}$$

sujeito a:

(2)–(5)

$$-\epsilon \leq \sum_{j=1}^{m} w_j y_j - \alpha \leq \epsilon. \tag{7}$$

A FO (6) consiste em minimizar o número de tarefas que precisam ser transferidas de um nó para outro. A restrição (7) garante que a solução obtida tenha o custo aproximadamente igual ao obtido utilizando o primeiro modelo, permitindo um pequeno desvio, especificado por ϵ. Note que apesar dessa nova restrição ser uma desigualdade, ela se comporta praticamente como uma igualdade, pois ϵ é consideravelmente menor que qualquer valor que w_j possa assumir, uma vez que o custo associado ao uso de um equipamento não é proporcional ao uso de seus recursos. Assim sendo, caso um nó esteja sendo utilizado, mesmo que subutilizado, consideramos o custo em sua totalidade.

4. Avaliação

A abordagem de otimização consiste na implementação dos dois modelos matemáticos apresentados na Seção 3.1 utilizando as interfaces do *solver* CPLEX para C++. Para avaliar a solução, foram geradas diversas instâncias para simular como os sensores de um hospital inteligente poderiam fazer o *offload* para nós na Nuvem ou Névoa. Todo nó possui a *engine* do Docker instalado, sendo responsável pela implantação dos *containers* e monitoramento dos recursos da máquina. Consideramos 3 recursos nos testes: RAM, armazenamento e CPU (em porcentagem). Como nós da Névoa podem ser de diferentes fabricantes e arquiteturas, executamos todas as tarefas previamente em cada tipo de máquina para coletar quanto de cada recurso é utilizado por cada tipo de tarefa.

Foram utilizados 3 computadores físicos, um Raspberry PI e uma VM na Nuvem para representar os nós na Névoa. Já na Nuvem, utilizamos 3 VMs com diferentes especificações. Definimos que o custo de um nó físico é equivalente ao custo de mantê-lo ligado operando em sua total capacidade, enquanto que o custo das VMs é estabelecido pelo provedor do serviço. As especificações dos nós podem ser vistos na Tabela 1.

Tabela 1. Especificações dos nós utilizados.

Alias	Tipo	CPU	Núcleos	RAM	Armazenamento	Custo
DESK1	Névoa	Intel i5-3450	4	16GB	500GB	0,27
DESK2	Névoa	Intel i5-4210U	2	12GB	250GB	0,06
DESK3	Névoa	Intel G2030	2	8GB	500GB	0,23
RPI2B	Névoa	ARM Cortex A7	4	1GB	16GB	0,002
N1S2	Névoa	Intel 2-6Gen	2	7,5GB	1024GB	0,4
N1S4	Nuvem	Intel 2-6Gen	4	15GB	1024GB	0,8
N1S8	Nuvem	Intel 2-6Gen	8	30GB	1024GB	1,60

Consideramos como ambiente de simulação um hospital inteligente, onde adaptamos 3 dispositivos simulados da plataforma OpenICE [Arney et al. 2017] e transformamos em aplicações rodando em *containers*, sendo elas um eletrocardiograma (ECG),

um oxímetro de pulso (OXI) e um monitor multiparamétrico (MULTI). Também criamos um *container* com uma aplicação web rodando o sistema de detecção de objetos em tempo real YOLO [Redmon and Farhadi 2018], com o objetivo de simular um serviço que classifica imagens faciais de um paciente para detectar desconforto e emitir alertas. Por fim, criamos uma aplicação que criptografa *streams* de dados para simular a proteção dos dados sensíveis gerados por uma máquina de ressonância magnética. Os requisitos computacionais exigidos por cada tarefa foram coletados através do monitor de recursos do Docker executando-as na frequência máxima de funcionamento. A Tabela 2 mostra quanto cada tarefa consome de recursos e a latência máxima exigida. A Tabela 3 apresenta a porcentagem de CPU consumida por cada tarefa em cada tipo de nó.

Tabela 2. Recursos exigidos por cada tipo de tarefa

Tarefa	RAM(MB)	Armaz.(MB)	Latn.(ms)
ECG	88 (8*)	90 (125*)	60
OXI	70 (8*)	70 (125*)	70
MULTI	60 (9*)	50 (125*)	50
FACIAL	200 (190*)	900	130
CRYPTO	250	600	160

Tabela 3. CPU necessária por cada tarefa no nó em que foi alocada

Nó	ECG	OXI	MULTI	FACIAL	CRYPTO
DESK1	0,1%	0,1%	0,44	12%	27%
DESK2	0,58%	0,56%	1,16%	41%	41%
DESK3	0,34%	0,32%	1%	50%	69%
RPI2B	0,36%	0,34%	1,16%	52%	26%
N1S2	0,71%	0,73%	0,92%	37%	59,4%
N1S4	0,12%	0,12%	0,48%	18%	28,2%
N1S8	0,06%	0,06%	0,18%	9,25%	14,84%

4.1. Geração das instâncias

Para avaliar a eficiência da alocação, levamos em consideração cenários de um hospital inteligente dispondo de diferentes números de leitos, onde cada paciente admitido irá utilizar mais de um tipo de sensor. Este hospital possui capacidade computacional suficiente para processar todos os dados coletados, portanto não existe limitação na disponibilidade de sensores. As instâncias de testes foram geradas associando aleatoriamente os tipos de sensores para os pacientes e gerando uma latência entre cada sensor e os nós dentro das faixas de 20 à 80 ms para nós na Névoa e de 40 à 160 ms para nós na Nuvem. Os recursos disponíveis em cada nó não serão amplamente utilizados pelas tarefas, pois assumimos que eles estarão dedicados à outras atividades como rotinas do sistema operacional, processamento dedicado ou demais atividades dos funcionários do hospital. Recursos na Névoa são fixos em cada instância do problema e recursos na Nuvem são ilimitados dado que podemos alocar VMs sob demanda. Então para cada 50 tarefas (N) na entrada acima da capacidade padrão da infraestrutura (500), alocamos 10 VMs. A Tabela 4 mostra a quantidade de cada nó usado nas instâncias e os valores de CPU disponíveis.

Tabela 4. Disponibilidade por tipo de nó.

	Desk1	Desk2	Desk3	RPI2B	N1S2	N1S4	N1S8
Tipo	Névoa	Névoa	Névoa	Névoa	Névoa	Nuvem	Nuvem
CPU (%)	70	70	60	60	90	90	90
Quantidade	5	5	5	20	5	$10 + N \times 10$	$10 + N \times 10$

Para os cenários de teste, consideramos 50, 100, 150 e 200 leitos em um hospital inteligente totalmente conectado [Vishnu et al. 2020]. Dado $P = [50, 100, 150, 200]$ o conjunto de cenários de ocupação de leitos, foram geradas 20 instâncias base (IB), 5 para cada elemento neste conjunto. Essas instâncias foram submetidas ao primeiro modelo sem restrições no tempo de execução a fim de obter o valor ótimo das soluções. Cenários

como este são dinâmicos, de modo que a solução em execução em um dado momento pode não ser a adequada no futuro. Por exemplo, múltiplos pacientes podem ter tido alta e seus sensores foram desligados, bem como tantos outros podem ter dado entrada no hospital. Ou seja, se o contexto muda, o sistema pode alocar o conjunto de novas demandas e realocar de maneira inteligente, se necessário, as demandas antigas. Para simular este cenário, alteramos aleatoriamente 20% das tarefas nas instâncias base, porém mantivemos o mesmo número de tarefas por questões de simplicidade. Chamaremos tais instâncias de instâncias derivadas (ID).

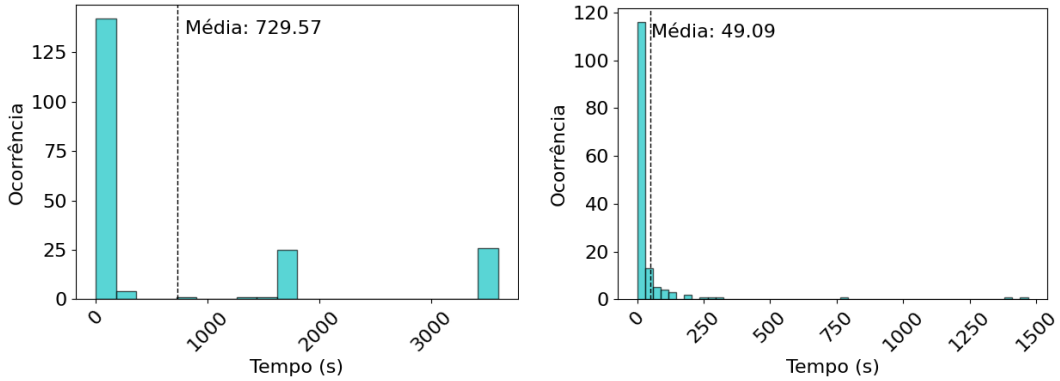
Como o problema de alocação é NP-difícil, o tempo de execução das instâncias base cresceu exponencialmente conforme a entrada. Na Tabela 5, os valores na coluna *ótimo* mostram o tempo de execução, em segundos, do primeiro modelo até atingir o valor ótimo pela primeira vez, enquanto que a coluna *Prova* consiste no tempo total de execução, ou seja, quanto tempo demorou para provar a otimalidade da solução.

Tabela 5. Tempo de execução (s) do primeiro modelo para as instâncias base

	50 pacientes		**100 pacientes**		**150 pacientes**		**200 pacientes**	
Instância	Ótimo	Prova	Ótimo	Prova	Ótimo	Prova	Ótimo	Prova
1	0,53	1,65	1,51	92	9,9	9,9	5	1753
2	0,57	0,57	1,97	77,2	1,7	17,65	1,91	1,91
3	2,16	2,16	3,5	3,5	2,3	2,3	7,7	9558
4	0,9	0,9	2,47	9,43	1,14	1,14	4,8	788,38
5	0,4	0,4	8,48	140	1,34	1,34	5,35	115,9

4.2. Instâncias derivadas

Todas as instâncias derivadas foram submetidas ao primeiro modelo com um limite de 60 minutos, exceto as instâncias 3, 4 e 5 com 200 pacientes na entrada. Durante a execução dessas instâncias o consumo de memória RAM cresceu além da capacidade da máquina, então a restrição de tempo foi reduzida para 30 minutos.



a) Todas as instâncias b) Com solução ótima

Figura 2. Distribuição do tempo de execução das IDs

A Figura 2(a) mostra a distribuição do tempo de execução de todas as IDs, incluindo as que foram pausadas pela restrição no tempo de execução. Considerando apenas as que tiveram a otimalidade confirmada (Figura 2(b)), vemos que o modelo pode demorar substancialmente para provar a otimalidade das soluções. Conforme a Tabela 6, a solução ótima foi confirmada em 74,5% (149) das instâncias, mas a execução mais

Tabela 6. Tempo t_j para confirmar a otimalidade.

# P	n	c <= 10 s	c >10s	min(t_j)	max(t_j)	$\bar{t_j}$
50	50	49(98%)	1(2%)	0,3s	19,24s	1,49s
100	38	9(23,68%)	29(76,32%)	2,3s	1469s	110s
150	50	35(70%)	15(30%)	0,95s	89,19s	12,76s
200	11	6(54,54%)	5(45,46%)	1,99s	1388s	220s
All	149	99(66,44%)	50(33,55%)	0,3s	1469s	49,09s

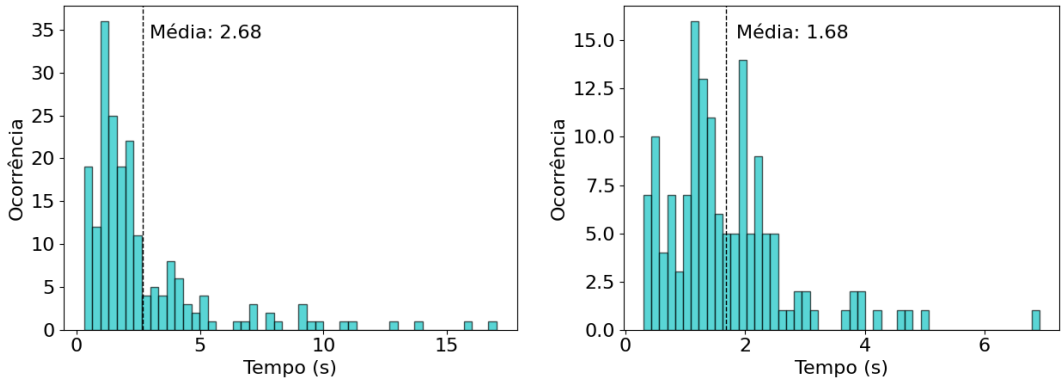
$$c = t_j - t_i$$

demorada durou 24 minutos aproximadamente. Também podemos notar que a diferença (c) entre o tempo da prova e a primeira ocorrência é menor que 10s em 66,44% dos casos.

Tabela 7. Tempo t_i para encontrar a primeira ocorrência da melhor solução

#P	n	t_i <= 5 s	t_i >5 s	min(t_i)	max(t_i)	$\bar{t_i}$
50	50	50 (100%)	0 (0%)	0,3s	2,24s	1,03s
100	50	50 (100%)	0 (0%)	0,99s	4s	1,47s
150	50	50 (100%)	0 (0%)	0,95s	4s	2,03s
200	50	29 (58%)	21 (42%)	1,99s	17,01s	6,16s
All	200	179 (89.5%)	21 (10,5%)	0,3s	17,01s	2,67s

A Tabela 7 apresenta o tempo necessário para obter pela primeira vez a melhor solução (não necessariamente a ótima) de cada execução,incluindo todas as IDs. Podemos perceber que estas soluções são obtidas em menos de 5 segundos para 89,5% dos casos e em menos de 17 segundos para os demais (10,5%). Se considerarmos apenas as 149 instâncias com otimalidade confirmada, a melhor solução aparece em menos de 5 segundos em 99,32% dos casos e em menos de 7 segundos para os demais (0,68%). Nas Figuras 3(a) e 3(b) pode-se visualizar o tempo para obter a melhor solução entre todas as IDs e entre apenas as com otimalidade comprovada, respectivamente.



a) Todas as instâncias b) Com solução ótima

Figura 3. Tempo para obter a melhor solução

Tabela 8. Análise do tempo necessário para obter *gap* com valor menor que 1%.

Parâmetro	100 p.	200 p.	All
Número de instâncias	12	39	51
Tempo médio para encontrar melhor solução	1,75 s	6,77 s	5,59 s
Tempo médio para atingir *gap* abaixo de 1%	10,14 s	5,74 s	6,77 s
Tempo médio para encontrar *gap* mínimo	44,21 s	258,18 s	207,84 s

Apesar do tempo de execução total em grande parte dos casos ser proibitivo em cenários reais, a taxa de 89% em que o valor ótimo aparece nos primeiros 5 segundos

de execução indica que não é necessário executar a abordagem de alocação por muito tempo para obter uma solução de excelente qualidade, ao menos em cenários semelhantes aos apresentados nesse trabalho. Mesmo considerando o maior tempo de execução das instâncias, vemos que o valor ótimo foi obtido em 17 segundos, o que não é proibitivo de ser utilizado em um ambiente real, já que a alocação de tarefas não deve ser executada em tempo real, mas sim para otimizar a alocação corrente quando eventos forem acionados para fazer uso eficiente de recursos. A Tabela 8 contém os resultados obtidos analisando a saída do resolvedor (CPLEX) para as instâncias em que não foi possível provar a otimalidade dentro da janela definida, sendo o *gap* a diferença entre a melhor solução encontrada até o momento e o valor limite da melhor solução possível. Vemos que o *gap* se aproxima de 1% rapidamente, e na maioria dos casos, a melhor solução já foi encontrada. Isto indica que em cenários semelhantes, os modelos utilizados conseguem gerar soluções de alta qualidade, com *gaps* próximos à 1% sem executar por longos períodos de tempo.

O primeiro modelo encontra soluções com um custo de infraestrutura reduzido, mas não leva em consideração a alocação corrente das tarefas. Devido à quantidade reduzida de tipos diferentes de nós e tarefas, é esperado que exista um grande número de soluções com o mesmo valor ótimo. Isso acontece porque dado que temos dois nós N_a e N_b com especificações idênticas e portanto o custo de utilização é também idêntico, se em uma solução ótima (S_a) N_a é utilizado e N_b não, existe outra solução (S_b) onde N_b é utilizado e N_a não. Isso explica porque o módulo rapidamente obtém a solução ótima, mas demora consideravelmente para confirmá-la.

4.3. Etapa de reotimização da solução

O segundo modelo é executado para encontrar uma solução com o custo da infraestrutura obtido no primeiro modelo, mas que mantenha o máximo de alocações já existentes executando nos mesmos nós. Fazemos isso para evitar implantar o resultado obtido inicialmente e gerar grande quantidade de migrações de tarefas entre os nós. Conforme vemos na Figura 4, o tempo de execução em relação ao valor da entrada cresce menos neste segundo modelo. A solução ótima foi obtida em todas as instâncias, sendo em menos de 5 segundos para 85,5% dos casos e em menos de 27 para os demais (14,5%). Como o resultado obtido pela execução desse segundo modelo irá gerar uma economia por não precisar reconfigurar o comportamento de sensores e equipamentos de rede, o tempo adicional desprendido em sua execução é justificado.

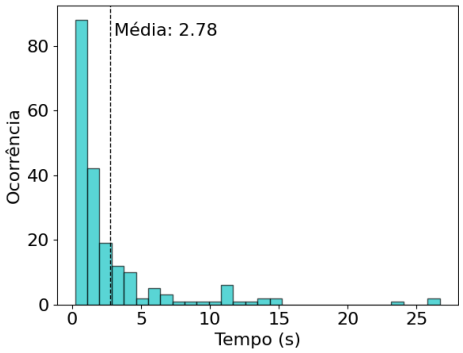


Figura 4. Tempo de execução do segundo modelo

5. Conclusão

Este trabalho propôs uma abordagem de alocação baseada em dois modelos matemáticos para auxiliar na alocação de recursos computacionais em ambientes de saúde inteligente, com o objetivo de garantir os requisitos das aplicações e reduzir o custo da infraestrutura. Os testes mostraram que a abordagem é capaz de entregar soluções de excelente qualidade, com grandes chances de serem soluções ótimas, em um curto período de tempo. Diferente de outros trabalhos, as soluções obtidas levam em consideração o estado corrente

do ambiente de execução, reduzindo o número de migrações necessárias para implantar a solução obtida. Estas migrações só irão ocorrer quando forem estritamente necessárias para reduzir o custo da infraestrutura.

Referências

Arney, D., Plourde, J., and Goldman, J. (2017). Openice medical device interoperability platform overview and requirement analysis. *Biomedical Engineering / Biomedizinische Technik*, 63.

Gross, J. and Geyer, C. F. (2020). A cost efficient model for minimizing energy consumption and processing time for iot tasks in a mobile edge computing environment. In *Anais do XII Simpósio Brasileiro de Computação Ubíqua e Pervasiva*, pages 41–50.

Li, Q., Zhao, J., Gong, Y., and Zhang, Q. (2019). Energy-efficient computation offloading and resource allocation in fog computing for internet of everything. *China Communications*, 16(3):32–41.

Liu, C., Xiang, F., Wang, P., and Sun, Z. (2019). A review of issues and challenges in fog computing environment. In *2019 IEEE DASC/PiCom/CBDCom/CyberSciTech*, pages 232–237.

N. Jayasena, K. P. and Thisarasinghe, B. S. (2019). Optimized task scheduling on fog computing environment using meta heuristic algorithms. In *2019 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 53–58.

Peralta, G., Iglesias-Urkia, M., Barcelo, M., Gomez, R., Moran, A., and Bilbao, J. (2017). Fog computing based efficient iot scheme for the industry 4.0. In *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, pages 1–6.

Rahabri, D. and NICKRAY, M. (2019). Low-latency and energy-efficient scheduling in fog-based iot applications. *TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES*, 27:1406–1427.

Rahbari, D. and Nickray, M. (2017). Scheduling of fog networks with optimized knapsack by symbiotic organisms search. In *2017 21st Conference of Open Innovations Association (FRUCT)*, pages 278–283.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv*.

Rezazadeh, Z., Rezaei, M., and Nickray, M. (2019). Lamp: A hybrid fog-cloud latency-aware module placement algorithm for iot applications. In *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, pages 845–850.

Souza, V. B. C., Ramírez, W., Masip-Bruin, X., Marín-Tordera, E., Ren, G., and Tashakor, G. (2016). Handling service allocation in combined fog-cloud scenarios. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–5.

Vishnu, S., Ramson, S. J., and Jegan, R. (2020). Internet of medical things (iomt) - an overview. In *2020 5th International Conference on Devices, Circuits and Systems (ICDCS)*, pages 101–104.

Wang, X. and Jin, Z. (2019). An overview of mobile cloud computing for pervasive healthcare. *IEEE Access*, 7:66774–66791.

Wen, Z., Yang, R., Garraghan, P., Lin, T., xu, J., and Rovatsos, M. (2017). Fog orchestration for internet of things services. *IEEE Internet Computing*, 21:16–24.

Zhao, X. and Huang, C. (2020). Microservice based computational offloading framework and cost efficient task scheduling algorithm in heterogeneous fog cloud network. *IEEE Access*, 8:56680–56694.