Alocação de Tarefas com Simulated Annealing na Borda da Rede para Internet Industrial

Vitor Gabriel Reis Lux Barboza, Janine Kniess

Pós-Graduação em Computação Aplicada
Universidade do Estado de Santa Catarina - Joinville, Brasil

vitor.barboza@edu.udesc.br, janine.kniess@udesc.br

Abstract. *Industrial Internet application bring requirements to meet critical tasks with low response time. Edge Computing offers an alternative to process the application data with low latency. In this work, we propose an approach for task allocation of industrial vehicles in the edge. The edge node receives tasks from different vehicles to process. Thus, is needed to define the best task processing order that meets the deadline and priority requirements. The solution was modeled based on the heuristic Simulated Annealing and the results demonstrate that the Task Allocation Approach was able to select the best task processing combination that obeys the requirements.*

Resumo. *Aplicações de Internet Industrial requerem o atendimento de tarefas críticas com baixo tempo de resposta. A Computação na Borda oferece uma alternativa ao processamento de dados na nuvem computacional, pois possibilita reduzir a latência. Este trabalho traz uma abordagem para a alocação de tarefas de veículos industriais. O nó de borda recebe as tarefas dos veículos e deve estabelecer a melhor sequência de atendimento considerando o tempo limite e a prioridade de cada tarefa. A solução foi modelada com base na heurística Simulated Annealing, e os resultados demonstram que a abordagem pôde selecionar combinações de processamento que minimizam o tempo de resposta.*

1. Introdução

A Internet das Coisas Industriais (IIoT) é uma seção de dispositivos conectados à Internet usados em fábricas inteligentes, manutenções remotas a linhas de produção, ou vigilância. Os dispositivos IIoT possuem poucos recursos computacionais, como memória, processamento e energia. A computação em nuvem é uma tecnologia utilizada pela indústria para fornecer recursos, como armazenamento de longo prazo e poder de processamento [Xue et al. 2020]. No entanto, o tempo de resposta das aplicações IIoT é um requisito essencial nesses ambientes críticos, e o envio de dados para a nuvem acrescenta atrasos que podem impactar no tempo de resposta exigido pela aplicação.

De acordo com [Hou et al. 2022] devido à heterogeneidade dos dispositivos industriais e à sua limitação de recursos computacionais, como energia, processamento e armazenamento, parte das aplicações que os utilizam, requerem ambientes computacionais que ofereçam mais recursos. A Computação em Nuvem surgiu como opção para processar e armazenar os dados gerados pelas aplicações industriais, pois centraliza estes serviços. Contudo, as aplicações industriais comumente geram volumes de dados massivos, que podem criar congestionamentos e gargalos nos canais de comunicação para

a nuvem. Este conjunto de características associado ao elevado grau de criticidade das aplicações, conduz a requerimentos por recursos computacionais que com latência reduzida, bem como ofereçam redundâncias nos enlaces, ou outras técnicas que evitem quaisquer perdas de comunicação. Estes são requisitos difíceis de alcançar na indústria apenas com o uso de Computação em Nuvem. A Computação na borda é uma solução para este problema, porque leva os recursos computacionais para a borda da rede, próximo aos dispositivos IIoT e, como consequência, reduz a latência no tempo de resposta para as aplicações [Shi et al. 2016].

Para ilustrar o uso de computação na borda, considere o cenário de aplicação da Figura 1. O ambiente representa uma planta fabril de uma indústria composta por dispositivos IIoT, como veículos industriais conectados à rede, nós de borda *(edges)* e nuvem computacional. No cenário os veículos industriais são autônomos e atuam em funções distintas no ambiente da fábrica. Por exemplo, um primeiro modelo abastece com matéria prima e insumos as máquinas das linhas de produção, um segundo modelo atua como empilhador de caixas na logística interna, e outros modelos atuam em outras funções.

No ambiente, cada veículo é equipado com uma câmera que pode tirar fotos do compartimento de carga das máquinas de produção, buscando abastecê-las. Há também a necessidade de detectar se no seu percurso há algum obstáculo à frente; como uma pessoa, ou outro objeto, esta detecção é feita por meio da câmera, que tira fotos durante a locomoção. O veículo envia as imagens para a camada de computação na borda, que processa as imagens e devolve os resultados. Tais requisições no formato de imagens, ou dados de outros sensores do veículo, são as tarefas propriamente ditas, que precisam ser processadas na borda. O nó de borda recebe múltiplas solicitações dos veículos e precisa alocá-las de acordo com o limite de tempo e a prioridade de cada solicitação. Para realizar a alocação de tarefas, este artigo apresenta como contribuição, o TASϵC, uma abordagem para a alocação de tarefas de dispositivos IIoT baseado na heurística *Simulated Annealing* (SA) [Kirkpatrick et al. 1983], que seleciona a combinação otimizada para atender aos prazos e a prioridade das tarefas da aplicação.

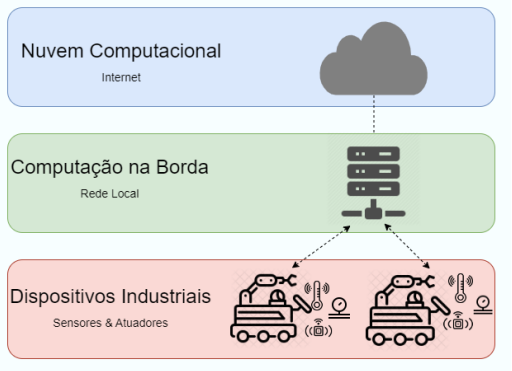

Figura 1. Cenário de Aplicação (Elaborado pelos autores).

A alocação de tarefas na nuvem computacional não será considerada como uma possibilidade neste trabalho, apenas alocações na borda serão avaliadas. Este artigo está organizado como segue: Na Seção 2, apresenta-se os trabalhos relacionados. Na Seção 3, descreve-se a abordagem para a Alocação de Tarefas, que detalha a formulação do pro-

blema, a modelagem do algoritmo com o SA e os mecanismos para lidar com as falhas na entrega das mensagens, por exemplo, decorrentes da mobilidade dos veículos. Na Seção 4, mostra-se os resultados obtidos através das simulações no simulador iFogSim [Gupta et al. 2017]. Por fim, a Seção 5 traz as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Alguns estudos propõem soluções para o escalonamento de tarefas, apresentando diferentes estratégias e heurísticas na busca por soluções otimizadas ao problema.

No trabalho de [He 2022] o autor propõe uma abordagem com uso de computação de borda, para prover a alocação de tarefas de dispositivos industriais. Em sua abordagem cada tarefa possui um prazo para completar o processamento. Tarefas sensíveis a atrasos são priorizadas pelo mecanismo para que recebam o processamento com antecedência. A meta-heurística selecionada pelo autor para definir as decisões do processo de alocação de tarefas é o *Genetic Algorithm (GA)* [Davis 1991]. Utilizando o simulador CloudSim, os autores compararam a solução de escalonamento com dois algoritmos exatos, dentre eles o de busca gulosa. O estudo propõem cenários variando de 15 a 40 tarefas. Os resultados apresentam o GA atendendo satisfatoriamente a minimização do tempo de resposta e com resultados melhores que os outros dois algoritmos comparados.

No trabalho de [Bai 2020], os autores apresentam um cenário real de indústria de energia, onde a queima de carvão é o principal agente de geração de energia e precisa ser monitorada por câmeras que analisam o nível de resíduos liberados no ar durante o processo de queima. Os dispositivos IIoT, neste caso são câmeras fixas em determinados pontos do processo, que enviam seus dados por rede sem fio. Os autores propõem um cenário em que as tarefas geradas pelos dispositivos não possuem dependência entre si, ou seja não têm uma sequência padrão de execução. Os autores elaboraram um esquema de priorização no processamento das tarefas, são efetuados cálculos relacionados aos requerimentos de latência em função do tamanho de cada tarefa e dos recursos computacionais disponíveis no *edge*.

Foi codificado um algoritmo híbrido, que une a otimização Lyapunov, de [Salle and Lefschetz 1961] com o algoritmo *Matching* apresentado por [Gale and Shapley 1962], a fim de estabilizar e controlar a alocação de tarefas. Os autores concluem que a estratégia de alocação de tarefas obteve melhores resultados, pois ela considera os limites de atraso de cada tarefa ao realizar a otimização e portanto pode ser aplicada em ambientes industriais críticos.

No trabalho relacionado de [Sun et al. 2021], os autores citam um cenário de monitoramento de indústria química com processos críticos de monitoramento de poluentes, no qual os dispositivos IIoT possuem sensores para monitoramento do ambiente. As tarefas são alocadas na borda tendo por objetivo principal o de reduzir o tempo de processamento de todas as tarefas geradas a partir dos sensores. A solução apresenta preocupação com a prioridade de cada tarefa para alcançar a ordem otimizada, gerenciando e otimizando as prioridades de cada tarefa. As tarefas não possuem dependências entre si, mas carregam um índice de prioridade que determina parte do fluxo de execução adequado que deve ser seguido pelo servidor de borda.

Para alcançar este objetivo de otimizar a estratégia de alocação de tarefas, o estudo propõe o uso de um algoritmo híbrido, com uso de *Bayesian Networks*, concebido por

[Pearl 1985] em conjunto com um algoritmo populacional, o selecionado neste caso é o *Particle Swarm Optimization (PSO)*. Os resultados da simulação apontam que a solução ultrapassou os resultados em relação aos algoritmos comparados, genético híbrido, um algoritmo PSO híbrido e um algoritmo de evolução diferencial híbrido.

No trabalho [You and Tang 2021] é proposta uma estratégia de otimização para alocação de tarefas. A heurística utilizada foi o *Particle Swarm Optimization*. Considera-se mais de um nó de borda para processar as tarefas. Os resultados mostram melhor desempenho com o PSO quanto ao tempo de resposta quando comparado ao Algoritmo Genético e ao *Simulated Annealing*. Os autores concluem que a parametrização do PSO num cenário de múltiplos nós *edge*, favoreceu a redução do tempo.

No artigo de [Matrouk 2023], propõe-se alocar tarefas de dispositivos IIoT na borda da rede. O algoritmo de otimização para determinar a decisão de alocação é o *Animal Migration Optimization*. Foi utilizado o simulador iFogSim para comparações com o *Improved Firefly Algorithm* e *Moving Target Search*. Os resultados demonstram a superioridade desta solução proposta em comparação aos demais algoritmos.

O Algoritmo TASϵC proposto, agrega os seguintes aspectos aos trabalhos relacionados: (i) realiza a alocação considerando o tempo limite da tarefa; (ii) contabiliza o tempo que as tarefas aguardam na fila no nó de borda; (iii) considera a prioridade de cada tarefa; e (iv) considera a mobilidade dos dispositivos industriais. A meta-heurística selecionada para este cenário é o *Simulated Annealing (SA)* [Kirkpatrick et al. 1983]. Seu papel é apresentar para o *edge* a decisão otimizada sobre a ordem de execução das tarefas. A escolha dessa meta-heurística em relação as apresentadas, justifica-se por ela apresentar como principal característica o fato dela evitar mínimos locais, empregando uma busca aleatória que, por vezes, aceita vizinhos cuja energia seja mais elevada.

Dois algoritmos serão utilizados adicionalmente para fins de comparação: (i) o Quicksort, [Hoare 1961], é um algoritmo do tipo dividir e conquistar. Ele funciona selecionando um elemento inicial do conjunto e particionando os outros elementos em dois subconjuntos, conforme sejam menores ou maiores que o elemento inicial, proporcionando a partir destes subconjuntos a solução de ordenação do conjunto maior, as tarefas recebidas, nas comparações, este algoritmo (Quicksort) permitirá avaliar as tarefas ordenadas por exemplo, com as mais prioritárias à frente das menos prioritárias, ou as com menores tempo de limite à frente das com maiores tempo de limite; (ii) já o primeiro a entrar, primeiro a sair, do inglês, *first in first out (FIFO)* [Tanenbaum and Bos 2015], é um método para organizar a manipulação de uma estrutura de dados, no qual o mais antigo (primeiro) a dar entrada na fila, é processado primeiro, a seguir o processamento é sequenciado até que o último a entrar seja processado.

3. Modelagem do Sistema e Definição do Problema

A formulação do problema de alocação de tarefas considera o cálculo do tempo de resposta para cada solicitação enviada pelos veículos. O cenário inclui um nó de borda, que possui uma fila de entrada para as tarefas recebidas e veículos industriais, $V = \{v1, v2, \ldots, vn\}$, sendo um veículo representado por $v \in V$. Cada veículo envia múltiplas solicitações para serem processadas. As tarefas são representadas por $M = \{i1, i2, \ldots, in\}$, onde $i \in M$. Cada tarefa corresponde a uma tupla com os atributos $T_i = (i, T_i^{limit}, D_i, C_i, P_i, v)$, em que T_i^{limit} é o prazo para conclusão da tarefa, em segun-

dos; D_i, é o tamanho total do pacote, em bits, ele corresponde ao tamanho do conteúdo da tarefa acrescido do cabeçalho do pacote, por exemplo, o tamanho de uma imagem obtida pela câmera, ou os dados lidos de um sensor, ou seja, é o tamanho dos dados que necessitam ser processados; C_i é a quantidade de ciclos de CPU para processar um bit de dados; P_i corresponde ao nível de prioridade da tarefa, 1, refere-se a uma tarefa de alta prioridade e 2 baixa prioridade. A formulação do problema é tratada como uma permutação de inteiros, e cada ordem de combinação para execução da tarefa tem um custo associado.

O tempo total de resposta de uma tarefa é dado pela RT_i, Equação 3, que corresponde à soma do atraso de transmissão RT_{trans}, do tempo de processamento RT_{exec} e o atraso de fila no nó de borda RT_{queue}. A função objetivo para a minimização do tempo de resposta é representada pela Equação 4. Executar as tarefas dentro do limite de tempo constitui uma restrição na função objetivo, Equação 5. O tempo de resposta deve ser menor ou igual ao tempo limite da tarefa. Uma função de penalidade é adicionada à $f(x)$, Equação 6. Quando o escalonador se depara com uma infração de tempo, a penalidade adicionará mais tempo ao RT_i, levando o escalonador a selecionar soluções que atendam à restrição. Definimos um coeficiente penalidade, $g \in [0, 1]$, onde 0 representa nenhuma penalidade e 1 o pior caso.

$$RT_{trans} = \frac{D_i}{R_j} \tag{1}$$

$$RT_{exec} = \frac{D_i \times C_i}{f_j} \tag{2}$$

$$RT_i = RT_{trans} + RT_{exec} + RT_{queue} \tag{3}$$

$$min \sum_{v=1}^{V} \sum_{i=1}^{M} RT_{v,i}^{j} \tag{4}$$

$$s.t.: \ RT_i \leq T_i^{limit} \tag{5}$$

$$f(x) = \sum_{v=1}^{V} \sum_{i=1}^{M} RT_{i,v}^{j} + g \times \sum_{v=1}^{V} \sum_{i=1}^{M} (RT_{v,i}^{j} - T_i^{limit}) \tag{6}$$

No Algoritmo 1, o procedimento *Task Allocation* representa o SA atuando na otimização da alocação de tarefas. As entradas são as tuplas de cada tarefa recebidas dos veículos, e parâmetros nativos do SA. A solução inicial (linha 2), é gerada a partir do recebimento das tarefas que são analisadas no procedimento *Initial Solution*. Em seguida o algoritmo efetua os cálculos do tempo de resposta (linha 3), aplicando a fórmula da $f(x)$, por meio do procedimento *Calculate RT Total* (Algoritmo 2). O SA efetua buscas por soluções vizinhas (linhas 4 a 18), utilizando técnicas estocásticas que proporcionem uma solução otimizada e que atendam aos requisitos e restrições de prazo e prioridade das tarefas.

No Algoritmo 1, os dados de entrada são as tuplas de cada tarefa e os dois parâmetros nativos do SA: o número total de iterações N e o coeficiente de resfriamento α. A solução inicial, S (linha 2), é gerada através da leitura de todas as tarefas na fila do *edge*. O tempo de resposta total para a completude do processamento pelo *edge* (linha 3) é obtido pelo procedimento que executa a Função de Fitness no Algoritmo 2, linhas 1

Algorithm 1 Task Allocation with SA

```
1:  function TASK_ALLOCATION (i, D_i, C_i, T_i^limit, P_i, v, N, α)
2:      S ← INITIAL_SOLUTION(i, D_i, C_i, T_i^limit, P_i, v;)            ▷ Generate initial solution S
3:      RT_total ← CALCULATE_RT_TOTAL(S); S* ← S;                        ▷ Set initial solution S as best solution
4:      for (Iter = 1; Iter < N; Iter + +) do
5:          S' ← NEIGHBOR_SOLUTION(S);                                   ▷ Generate neighbor solution S'
6:          RT_neighbor ← CALCULATE_RT_TOTAL(S');
7:          if (RT_neighbor < RT_total) then
8:              S ← S'; RT_total ← T_neighbor;
9:          if (RT_total < RT_best) then
10:             S* ← S;                                                  ▷ Set current solution S' as best solution
11:             Δ = RT_neighbor − RT_total;                              ▷ Calculate the delta over the solutions
12:             r ← generate r ∈ [0, 1];                                 ▷ Select randomly a float between 0 and 1
13:         if (r < e^{−Δ/Temp}) then
14:             S ← S';                                                  ▷ Set solution S' as the current solution
15:             RT_total ← RT_neighbor;
16:         Temp = (1 − (Iter/N))^α;                                     ▷ Update temperature
17:     S* ← PRIORITIZATION(S*);
18:     return (S*, RT_best);
19: function INITIAL_SOLUTION
20:     new Task(Task[0] = i, Task[1] = D_i, Task[2] = C_i, Task[3] = T_i^limit, Task[4] = P_i, Task[5] = v);
21:     RT_trans = D_i/R_j;
22:     RT_exec = (D_i × C_i)/f_j;
23:     RT_i = RT_trans + RT_exec + RT_queue;
24:     S ← new ArrayList < Tasks >;
25:     for (j = 1; j ≤ get_size(ArrayList < Tasks >); j + +) do
26:         task ← get_int_random(get_size(ArrayList < Tasks >));
27:         if (S[j] ⊅ get_value(task, ArrayList < Tasks >)) then
28:             S[j] ← get_value(task, ArrayList < Tasks >);
29:     return S;
```

a 16, onde a variável RT_{queue} inicialmente recebe o valor zero para cada tarefa. A cada iteração do algoritmo, o valor de RT_{queue} é incrementado pela soma do tempo de resposta das tarefas anteriores com o tempo para a execução das próximas tarefas em sequência na RT_{queue}.

No Algoritmo 1, linha 5, as soluções vizinhas são geradas através de trocas (*swaps*) randômicos das posições das tarefas em relação a sequência anterior para encontrar a melhor solução (linhas 7 a 18). Se uma solução melhor for encontrada, o algoritmo a seleciona como a melhor; caso contrário, usará o vizinho encontrado na busca estocástica como a melhor solução. Após isso, seguindo a semântica do SA, a temperatura muda (linha 16).

A Figura 2, exemplifica o comportamento do TASϵC ao realizar o escalonamento das tarefas nó de borda. Na Figura 2 (a), tem-se o momento inicial em que a fila está recebendo as tarefas enviadas simultaneamente pelos veículos. As tarefas enviadas pelos veículos receberão um índice na ordem em que chegarem, então a primeira tarefa recebeu o índice T1. Depois disso, T2, T3 e assim por adiante.

A transição de (a) para (b), representa o momento em que o TASϵC está calculando os tempos de resposta das tarefas, enquanto executa as primeiras trocas de posição da fila. No cenário, algoritmo identificou que as tarefas T4 e T5 estão fora do limite de tempo. Para atender aos requisitos da aplicação, determinou-se que T4 deve ser processada primeiro e T5 logo depois. O algoritmo busca melhores posições para T4 e T5 pelo *swap*. Após, decide que T1 será movido para a posição de T5, T4 será movido para a posição de T1 e T5 será movido para a posição de T4. Na transição de (b) para (c) o re-

Algorithm 2 Total Response Time

```
1:  function CALCULATE_RT_TOTAL(S)
2:      RT_queue ← 0;
3:      for (j = 0; j ≤ get_size(S); j++) do
4:          fromTask = getTask(j);
5:          if (j + 1 < get_size(S)) then
6:              destTask = getTask(j + 1);
7:          else
8:              destTask = getTask(j);
9:          if ((fromTask.RT_i(destTask) + RT_queue) ≥ T_i^limit) then
10:             if ((fromTask.RT_i(destTask) + RT_queue) ≥ 2 × T_i^limit then
11:                 g = 1;                                   ▷ Set penalty coefficient
12:             else
13:                 g = T_i^limit − fromTask.RT_i(destTask);
14:         penalty+ = g × (T_i^limit − fromTask.RT_i(destTask));   ▷ Calculate the penalty
15:         RT_queue = fromTask.RT_i(destTask) + RT_queue;
16:     RT_total = RT_queue + penalty; return RT_total;      ▷ Set the RT_total
17: function NEIGHBOR_SOLUTION
18:     S' ← new Array[ ];
19:     x1 ← int_random(size(S));
20:     x2 ← int_random(size(S));
21:     if (x1 == x2) then NEIGHBOR_SOLUTION(S);
22:     else
23:         if (x1 > x2) then
24:             if (destTask.P_i(fromTask) < fromTask.P_i(destTask)) then
25:                 S'[x1] ← S[x2];
26:                 S'[x2] ← S[x1];
27:             else
28:                 if (destTask.P_i(fromTask) == fromTask.P_i(destTask)) then
29:                     if (destTask.T_i^limit < fromTask.T_i^limit) then
30:                         S'[x1] ← S[x2];
31:                         S'[x2] ← S[x1];
32:     return S';
```

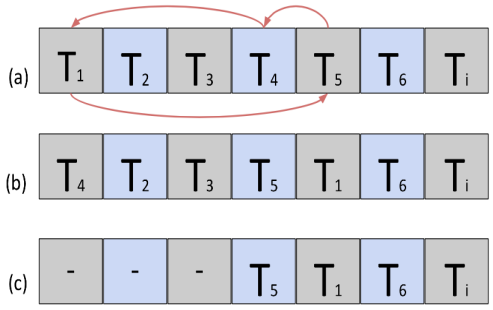

Figura 2. Escalonamento com TASϵC no *edge*.

sultado da alocação atende aos limites T4 e T5 e os demais limites das outras tarefas. Em (c) o nó de borda começa a processar cada uma das tarefas na ordem dada. Ao finalizar o processamento, as respostas são devolvidas aos veículos, finalizando a execução.

No contexto de uma planta fabril na indústria com veículos industriais, falhas podem ocorrer no envio e recebimento das mensagens devido a mobilidade dos veículos e interferências dos ativos da indústria. No processo de escalonamento de tarefas, foi desenvolvido um temporizador para gerenciar o envio e recebimento de mensagens pelo dispositivo no veículo. Por exemplo, o veículo, após enviar a primeira mensagem de requisição ao *edge*, aciona o temporizador, denominado $\Delta_{request}$ definido na Equação 7. O temporizador $\Delta_{request}$ define o tempo total em que o requisitante espera receber uma

mensagem de resposta do *edge* indicando que a solicitação foi recebida. Se o dispositivo no veículo não receber uma confirmação durante $\Delta_{request}$, envia um novo pedido.

$$\Delta_{request} = K \tag{7}$$

Na Equação 7, o valor de *K* especifica o atraso de ida e volta para 1 (um) salto na rede entre o veículo e o *edge*.

4. Avaliação de Desempenho

Foram realizados experimentos utilizando o Simulador iFogSim [Gupta et al. 2017]. Três cenários distintos foram configurados, (ver a Tabela 1). Na Tabela 2, apresenta-se a descrição dos grupos de tarefas submetidas ao SA, bem como os parâmetros de entrada do algoritmo. Um número de 5 veículos foi utilizado e a principal diferença entre os cenários é o número de tarefas M de cada veículo. Para cada cenário, o Algoritmo 1 foi executado 30 vezes. Além do TASϵC foram realizadas comparações com os algoritmos, *First-in-First-out (FIFO)* e *Quicksort*. Dois atributos foram usados no escalonamento das tarefas: prioridade e tempo limite.

O computador utilizado nos experimentos possui processador Intel Core i7 de 1,8 GHz, 16 GB de RAM e sistema operacional Windows 10. Definiu-se que a planta fabril tem uma área de 500m $\times$ 500m (quinhentos metros de comprimento, por quinhentos metros de largura). Considerou-se também que a planta fabril possui uma rede sem fio do tipo infra estruturada operando no padrão IEEE 802.11ac, com antenas omnidirecionais e o sinal de alcance das antenas em 300m (trezentos metros). A taxa de transmissão da rede sem fio, é de 100 Mbps.

Tabela 1. Cenários de Configuração

Cenário	Grupo de Tarefas	Veículo 1	Veículo 2	Veículo 3	Veículo 4	Veículo 5	Total
S1	Grupo 1	0	1	2	0	4	7
	Grupo 2	4	1	2	3	1	11
	Grupo 3	3	2	2	5	3	15
	Grupo 4	2	3	4	2	4	15
	Total	9	7	10	10	12	**48**
S2	Grupo 1	2	1	5	3	3	14
	Grupo 2	1	7	5	1	7	21
	Grupo 3	6	6	2	11	5	30
	Grupo 4	9	6	5	4	7	31
	Total	18	20	17	19	22	**96**
S3	Grupo 1	2	3	11	5	2	23
	Grupo 2	9	4	8	7	11	39
	Grupo 3	9	7	9	9	6	40
	Grupo 4	8	8	11	8	7	42
	Total	28	22	39	29	26	**144**

No cenário S1, Figuras 3 e 4, os algoritmos avaliados executaram as solicitações recebidas dentro do prazo para o Grupo 1, porém nos Grupos 2, 3 e 4, o FIFO extrapolou o limite. O algoritmo TASϵC apresentou comportamento semelhante ao Quicksort quando ordenado por T_i^{limit}, por outro lado o TASϵC teve desempenho melhor que o Quicksort

Tabela 2. Descrição dos Parâmetros do Sistema

Grupo	Tarefa	T_i^{limit}	D_i	C_i	P_i	RT_i
1	Fotografar uma máquina industrial	12	66.355.200	180	Alta ou Baixa	2.1565
2	Detectar movimentos ao redor do veículo	6	16.588.800	44	Alta ou Baixa	0.2571
3	Detectar obstáculos à frente do veículo	3	7.372.800	20	Alta ou Baixa	0.0921
4	Enviar a posicionamento atual do veículo	1,5	400.000	100	Alta ou Baixa	0.0090

quando ordenado pela prioridade das tarefas. O TASϵC leva em consideração no escalonamento os dois atributos, o tempo limite da solicitação e a prioridade. O Quicksort é executado separadamente, considerando a prioridade e tempo limite.

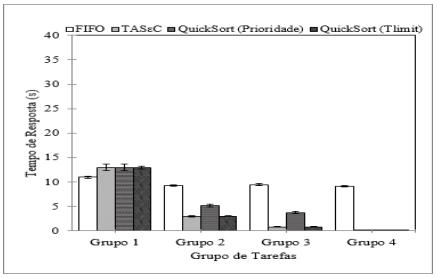

Figura 3. Tempo Resposta (S1)

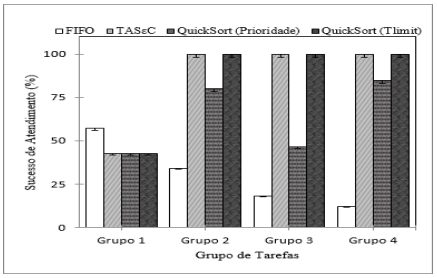

Figura 4. Atendimento (S1)

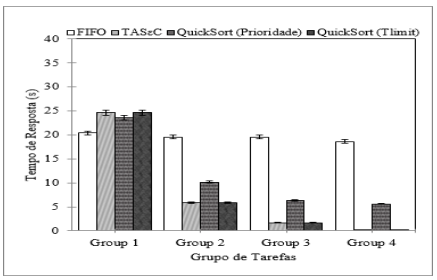

Figura 5. Tempo Resposta (S2)

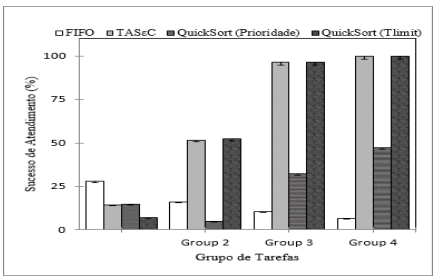

Figura 6. Atendimento (S2)

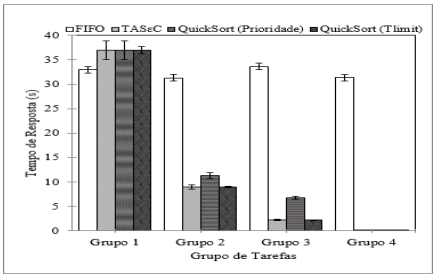

Figura 7. Tempo Resposta (S3)

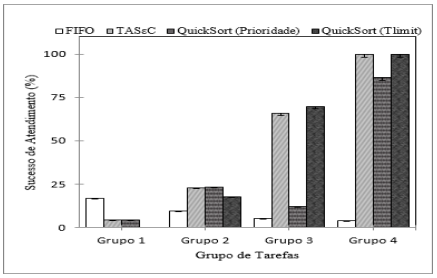

Figura 8. Atendimento (S3)

As Figuras 5, 6, 7, 8, mostram os resultados com os cenários S2 e S3. Em ambos, observou-se que o alto volume de tarefas afetou o tempo de resposta. O TASϵC atendeu todas as tarefas com prioridade alta antes de elas terem seus tempos limites alcançados. Na Tabela 3 é mostrado que todas as tarefas de alta prioridade foram atendidas pelo TASϵC nos cenários S1, S2 e S3. O Quicksort não conseguiu lidar com todas as tarefas urgentes nesses cenários.

Pode-se concluir pelos experimentos que o TASϵC teve um desempenho melhor nas tarefas mais urgentes do que o Quicksort à medida que o número de solicitações

aumentou. No cenário S2, o número de solicitações é 96, e no cenário S3, o número é 144. Nestes casos, o Quicksort atendeu em média 98% das solicitações no cenário S2 e 99% no cenário S3. Ressalta-se que em ambientes industriais, a falha na conclusão de tarefas urgentes pode resultar em perdas financeiras e riscos para a vida dos funcionários.

Tabela 3. Sucesso de Atendimento das Tarefas

Percentual de Atendimento (%)		S1		*S2*		*S3*	
		Alta	Baixa	Alta	Baixa	Alta	Baixa
TASϵC	Grupo 1	100	33	100	0	100	0
	Grupo 2	100	100	100	52	100	0
	Grupo 3	100	100	100	95	100	61
	Grupo 4	100	100	100	100	100	100
QuickSort	Grupo 1	100	33	100	0	100	0
	Grupo 2	100	70	100	0	100	0
	Grupo 3	100	0	96	0	99	0
	Grupo 4	97	0	97	0	100	0

Com o objetivo de avaliar o temporizador $\Delta_{request}$ diante das perdas de mensagens, novos experimentos foram gerados agregando o temporizador na implementação do escalonador. Dois cenários foram propostos, um com 20 tarefas e um segundo com 40 tarefas, ambos usando 5 veículos. Os mesmos grupos da Tabela 1 foram mantidos, assim como os demais parâmetros de configuração do cenário. Os resultados demonstram as medianas de cada execução.

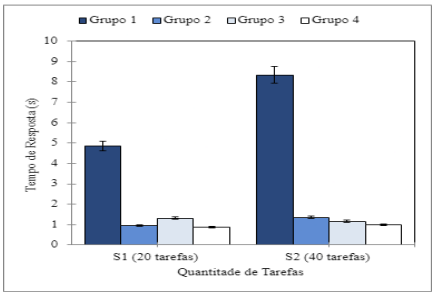

Figura 9. Tempo Resposta S1, S2.

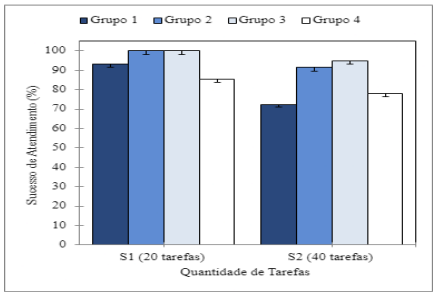

Figura 10. Atendimento por Tempo S1, S2

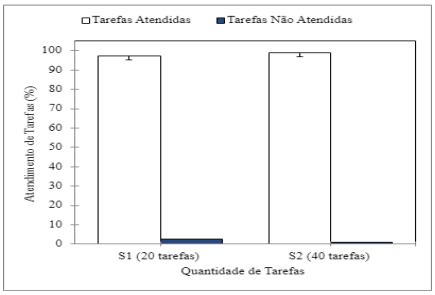

Figura 11. (%) Tarefas não Atendidas S1, S2

Com a adição do temporizador, contatou-se que o percentual de atendimento melhorou significativamente, por exemplo, o percentual de tarefas não atendidas para o

cenário S1 foi de 2,8% e para o cenário S2 de 1,0% (Figura 11). Na Figura 9, os resultados comprovam que o uso do temporizador não afetou negativamente o tempo de resposta.

5. Conclusões

Neste trabalho apresentamos a abordagem TASϵC para alocação de tarefas em ambiente industrial. O objetivo é atender tarefas em termos de prioridade e prazo. O *Simulated Annealing* foi utilizado como base para a definição do escalonamento das tarefas. A partir dos resultados, concluímos que o TASϵC priorizou o atendimento das tarefas mais urgentes, foi capaz de considerar a composição de atributos, tempo limite e prioridade de tarefas ao executar o agendamento. Atendendo dentro dos prazos limites, 100% das tarefas com urgência alta e cerca de 62% das tarefas com urgência baixa, nos três cenários de teste avaliados. Com a adição de um temporizador para tratar perdas de mensagens na rede, o percentual de requisições não atendidas ficou abaixo de 2,8% nos cenários avaliados sem impactar negativamente no tempo de resposta das tarefas.

Em trabalhos futuros, proporemos melhorias no TASϵC em relação a falhas na mensagem de resposta do *edge* com informações sobre o tempo de resposta total para a execução da tarefa. Também, em curto prazo, pretende-se comparar a proposta com outra meta-heurística de trabalhos da literatura apresentados na Seção 2, e expandir os experimentos para cenários maiores que os avaliados até o momento, como 200, 300 e 400 tarefas simultâneas a fim de observar o comportamento.

Referências

Bai, Y. (2020). 5g industrial iot and edge computing based coal slime flotation foam image processing system. *IEEE Access*, 8:137606–137615.

Davis, L. (1991). Handbook of genetic algorithms.

Gale, D. and Shapley, L. S. (1962). College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15.

Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., and Buyya, R. (2017). ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*.

He, J. (2022). Optimization of edge delay sensitive task scheduling based on genetic algorithm. In *2022 International Conference on Algorithms, Data Mining, and Information Technology (ADMIT)*, pages 155–159.

Hoare, C. A. R. (1961). Algorithm 64: Quicksort. *Commun. ACM*, 4(7):321.

Hou, W., Wen, H., Zhang, N., Wu, J., Lei, W., and Zhao, R. (2022). Incentive-driven task allocation for collaborative edge computing in industrial internet of things. *IEEE Internet of Things Journal*, 9(1):706–718.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Matrouk, K. e. (2023). Mobility aware-task scheduling and virtual fog for offloading in iot-fog-cloud environment. *Wireless Personal Communications*, 130:801–836.

Pearl, J. (1985). *Bayesian Networks: A Model of Self-activated Memory for Evidential Reasoning*. Report (University of California, Los Angeles. Computer Science Dept.). UCLA Computer Science Department.

Salle, L. and Lefschetz (1961). *Stability by Liapunov's Direct Method: With Applications*. Academic Press.

Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646.

Sun, L., Wang, J., and Lin, B. (2021). Task allocation strategy for mec-enabled iiots via bayesian network based evolutionary computation. *IEEE Transactions on Industrial Informatics*, 17(5):3441–3449.

Tanenbaum, A. and Bos, H. (2015). *Modern Operating Systems*. Always learning. Pearson.

Xue, Y., Wu, X., and Yue, J. (2020). An offloading algorithm of dense-tasks for mobile edge computing. In *ACM*, New York, USA. Association for Computing Machinery.

You, Q. and Tang, B. (2021). Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things. *Cloud Computing*.