Um Classificador de Prioridade de Requisições para Alocação de Recursos na Computação em Borda

Guilherme A. de Araújo[1], **Sandy F. C. Bezerra**[1], **Atslands R. da Rocha**[1]

[1] Programa de Pós-Graduação em Engenharia de Teleinformática (PPGETI), Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brazil,

{guialves, sandycosta}@alu.ufc.br, atslands@ufc.br

Abstract. *Edge Computing allows data processing and tasks to be performed close to end devices. However, due to the limited resources of Edge nodes, prioritizing the tasks to be executed is an efficient technique for managing resources. We implemented and compared three rule-based priority classifiers to determine the types of priority services so that Edge node resources can be optimally allocated. Our proposal explores how available resources can be distributed among services in a simulated environment. The results show that the best-performing classifier achieved an accuracy of 94% and a general error of 0.22, as evidenced by the CPU consumption of the Edge nodes.*

Resumo. *A Computação em Borda permite o processamento de dados e tarefas próximo aos dispositivos finais. No entanto, devido aos recursos limitados dos nós de Borda, priorizar as tarefas que serão executadas é uma técnica eficiente de gerenciar recursos. Implementamos e comparamos três classificadores de prioridade baseados em regras, para determinar os tipos de serviços prioritários, de modo que os recursos dos nós de Borda possam ser alocados de forma otimizada. A proposta explora como os recursos disponíveis podem ser distribuídos entre os serviços em um ambiente simulado. Os resultados mostram que o classificador com melhor desempenho obteve uma acurácia de 94% e um erro geral de 0.22, comprovado com o consumo de CPU dos nós de Borda.*

1. Introdução

O número crescente de dispositivos conectados da Internet das Coisas (do inglês, *Internet of Things, IoT*) impacta diretamente na quantidade de dados gerados por esses dispositivos. Estima-se que os dados gerados por dispositivos IoT sejam em torno de 79,4 zettabytes até 2025 [Statista 2020]. Para atender aos requisitos de mobilidade, localidade e baixa latência das aplicações IoT, infraestruturas de computação em borda tem sido largamente utilizadas. Nesse sentido, a Computação em Borda (*Edge Computing*) é um paradigma que complementa e fornece os serviços de Nuvem na borda da rede mais próximos dos dispositivos finais em uma escala amplamente distribuída [Coutinho et al. 2016].

Os nós de borda são geralmente mais limitados em recursos quando comparados aos nós de nuvem. De acordo com [Gupta et al. 2016], a escolha de recursos para atender às diversas requisições é um dos principais desafios da Computação em Borda, devido à limitação desses recursos e ao compartilhamento entre dispositivos IoT. Portanto, é crucial gerenciar os recursos eficientemente para atender às requisições das aplicações e garantir um desempenho satisfatório da execução dos serviços [Jr. and Araújo 2021].

Para lidar com o desafio do gerenciamento eficiente de recursos na Computação em Borda, surgiram algoritmos que visam otimizar e maximizar os recursos disponíveis para atender às requisições [Li et al. 2019]. Técnicas de aprendizado de máquina (do inglês, *Machine Learning*, ML) são utilizadas para automatizar soluções nesse contexto, mostrando-se um caminho promissor para o gerenciamento de recursos na Borda da rede [Zhao et al. 2018]). Alguns exemplos são o balanceamento dos dados, escalonamento de tarefas, qualidade de serviço (QoS) e o consumo efetivo dos recursos.

O gerenciamento de recursos na borda envolve diversas atividades, como a alocação de recursos para diferentes tipos de solicitações de serviço, monitoramento desses recursos e ajuste do dimensionamento para lidar com possível escassez. Priorizar os serviços que serão executados impacta no desempenho da rede. Essa técnica possibilita uma hierarquização das requisições fazendo elas terem uma ordem lógica de execução.

Neste trabalho, o foco é a priorização de serviços, que se mostra uma técnica interessante para balancear a utilização dos recursos da rede. Para isso, são selecionadas as requisições a serem executadas primeiro em detrimento de outras que podem consumir recurso em demasiado e causar indisponibilidade da rede. O objetivo é classificar tarefas em diferentes níveis de prioridade de modo a efetuar uma alocação de recursos mais assertiva. Dessa forma, a alocação é feita com base nas necessidades específicas do serviço requisitado e na capacidade do nó da borda de fornecê-lo, visando ofertar serviços de alta qualidade para usuários e aplicações.

Nesse contexto, apresentamos um mecanismo de classificação de prioridade de requisições para dar suporte ao gerenciamento de recursos dos nós da Borda. Através do monitoramento de recursos dos nós, seleciona-se o melhor nó para atender às requisições solicitadas e os recursos sobre os quais os serviços serão posicionados. Implementamos três modelos de classificação, *Support Vector Machine (SVM), K-nearest neighbors (KNN), Decision Tree*, e comparamos seus resultados a fim de obter um melhor classificador para controle das requisições executadas.

As principais contribuições deste trabalho são: (i) classificação de tipos de serviços com base em regras e pesos de prioridade; (ii) implementação e comparação de diferentes classificadores com intuito de definir prioridades aos tipos de serviços para que os recursos possam ser gerenciados em ambientes de Computação em Borda; (iii) alocação de recursos para as requisições com base nas prioridades definidas pelo classificador selecionado; (iv) validação do modelo preditivo de classificação frente ao consumo de CPU dos nós de Borda.

2. Trabalhos Relacionados

Dentre as abordagens propostas para gerenciamento de recursos em computação em borda, destacam-se as que tratam sobre escalonamento de tarefas e recursos baseados em prioridade. Em [Bhushan and Mat 2021] é proposta uma arquitetura de computação em borda baseada em fila de prioridades que aloca dinamicamente nós de borda com base na carga do sistema. Além disso, a alocação de tarefas é realizada com base na prioridade, a qual é baseada no provisionamento de serviços e define duas classes: Alta prioridade (sensível ao atraso) e Baixa prioridade (insensível ao atraso). Também é assumido um parâmetro que denota a proporção média do número de tarefas de alta prioridade para o número total de tarefas para calcular o tempo médio de atraso de cada classe.

O trabalho [Madej et al. 2020] desafia a suposição de que os recursos necessários estão disponíveis sempre que uma solicitação de trabalho é realizada. Entretanto, nem todas as solicitações de um servidor podem ser escalonadas em um nó de borda. Assim, garantir a equidade entre os clientes (servidores de nuvem descarregando tarefas) enquanto contabiliza as prioridades das tarefas torna-se crítico. Este artigo apresenta técnicas de escalonamento: FCFS (*First Come First Serve*); orientada aos clientes, considerando que todos possuem a mesma prioridade; orientada a prioridades, que considera as tarefas e não os clientes; e uma híbrida que considera ambos.

O artigo [Sharif et al. 2023] apresenta um mecanismo adaptativo de alocação de recursos para utilização efetiva de recursos no paradigma de Computação em Borda. Para obter a utilização ideal, os recursos disponíveis são alocados dinamicamente, considerando a natureza das solicitações recebidas. Após identificar a solicitação recebida, a prioridade é definida segundo a sensibilidade ao atraso da tarefa, verificado no processo de identificação. Os recursos disponíveis são, portanto, alocados conforme as prioridades das solicitações recebidas para satisfazer as restrições de acordo. O autor recorre a técnicas de alocação de recursos dinâmicas, considerando a natureza dessa solicitação, podendo não ser a melhor forma para alocação, tendo em vista que alocar sem conhecer os recursos disponíveis nos diversos nós de Borda, pode ser um problema.

Em [Bui et al. 2021] é apresentada uma melhoria de um algoritmo *Score Based Match-Making* para resolver o desafio de balanceamento de dados e qualidade de serviço. É proposto um algoritmo *Match-Making* baseado em pontuação de seis fatores (distância entre os nós, condição da rede, latência, memória, capacidade da CPU e o histórico de execução de tarefas do nó) para lidar com desafios relacionados à prioridade na alocação de recursos com um fator de preempção para lidar com problemas de emergência. Esse fator de preempção interrompe temporariamente uma tarefa sendo executada com a intenção de retomar à tarefa posteriormente, caso entre na fila uma tarefa de maior prioridade.

Dentre os trabalhos apresentados (resumidos na Tabela 1), o de [Bui et al. 2021] é o que mais se assemelha ao mecanismo proposto neste trabalho, pois também considera o recurso a ser alocado como critério de priorização. A real contribuição da proposta, que diferencia aos demais trabalhos, se dá em que todas as outras abordagens consideram apenas as tarefas e requisições, sempre supondo que os recursos estarão disponíveis conforme a necessidade do serviço solicitante. Porém, os recursos de Borda são limitados, e portanto, nem sempre estarão disponíveis para execução de uma requisição. O presente trabalho visa gerenciar os recursos nos nós de Borda, para permitir que haja um uso balanceado dos recursos da borda para atender as necessidades dos serviços dos solicitantes.

3. Classificação de Prioridade de Serviços

A Classificação é uma técnica de aprendizado de máquina que consiste em atribuir uma ou mais classes a um conjunto de dados, com base em suas características ou variáveis. Essa classificação pode ser binária (duas classes, 1 ou 0) ou multiclasse (três ou mais classes). Essa técnica é utilizada para identificar padrões e tendências nos dados, permitindo a tomada de decisão automatizada em diversos contextos.

Definimos um nó classificador com o intuito de classificar as requisições de aplicações dos dispositivos finais *(dados)* em diferentes níveis de prioridade *(classes)* para possibilitar uma alocação de recursos baseada nas prioridades definidas. Como ilus-

Tabela 1. Trabalhos Relacionados

Trabalho	Metodologia / Algoritmo	Critérios para Priorização
[Bhushan and Mat 2021]	Priorização Baseado em Provisionamento de Serviço	Sensibilidade ao Atraso e Tempo de Espera
[Madej et al. 2020]	Combinação e Priorização Baseado em Pontuação	Distância entre os nós, memória e capacidade de processamento
[Sharif et al. 2023]	Adaptativo e Baseado em Prioridade	Restrição de Tempo da Requisição
[Bui et al. 2021]	Pontuação Baseada em Seis Fatores	Distância entre os nós, Latência, Condição Estimada da rede, Memória, Processador e Histórico de Conexões dos Nós
Mecanismo Proposto	**Classificação Baseado em Prioridade de Serviços e Recursos**	**Tipo de Serviço e Recursos (CPU, Memória e Disco)**

trado na Figura 1, as aplicações nos dispositivos finais (IoT) enviam as requisições de serviço para o nó classificador. Este, por sua vez, utiliza essas requisições como entrada e as processa, resultando em uma fila de prioridades. Um componente no sistema monitora os recursos disponíveis nos nós de borda em tempo real. Esse componente retira as requisições da fila de prioridades e as encaminha para um nó de borda específico. Caso ocorra uma falha em um nó de borda, todas as requisições presentes nele serão realocadas para outros nós disponíveis. Se um nó de borda ou seus nós vizinhos não forem capazes de atender a uma determinada requisição, essa requisição será enviada para a nuvem.

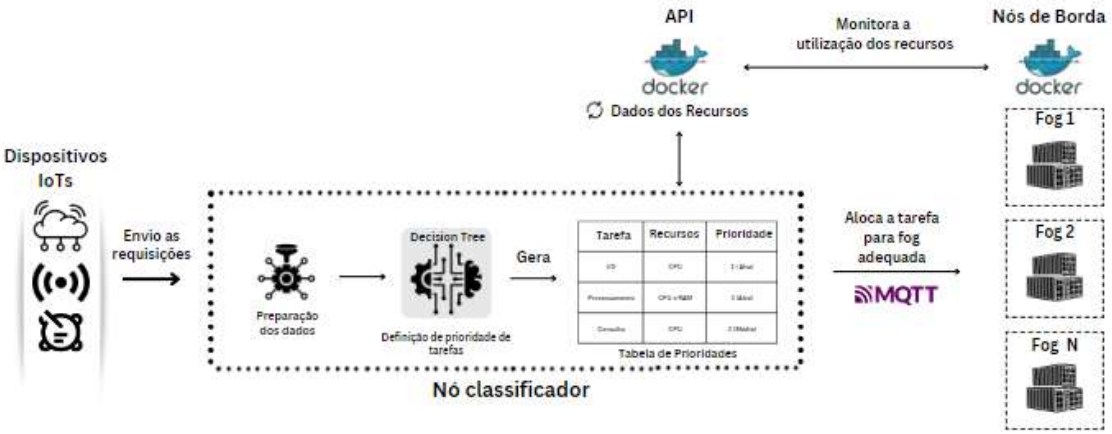


Figura 1. Arquitetura da Proposta Apresentada
Fonte: (Elaborado pelo Autor)

Seguimos a premissa de que toda requisição exige um tipo de serviço. Uma requisição é definida como uma tupla R_i = (TR_i, L_i, RN_i, H_i, D_i), onde TR_i é o tipo de

requisição i, L_i a latência da requisição i, RN_i o recurso necessário da requisição i, H_i a hora disparada a requisição i e o D_i refere-se ao tamanho do dado da requisição i.

Os dados de entrada do classificador (requisições dos dispositivos IoT) são, portanto, tuplas R_i. As requisições são geradas simulando uma aplicação de software sintético, na qual descreve todas as suas *features*, com o tipo de serviço dessa requisição, a hora que ele foi solicitada, o recurso necessário para atender a essa demanda, a latência em (ms) que descreve a distância do dispositivo final para o nó de borda mais próximo, e o tamanho do dado, de forma que possa haver um estresse do uso desses recursos da borda. O nó classificador atribui diferentes pesos e prioridades para as requisições recebidas, considerando principalmente o tipo de serviço e recursos necessários.

Classificamos as prioridades em três níveis: Alto (2), Médio (1) e Baixo (0). Para a definição das regras de prioridade dos serviços, os tipos de serviços são separados em duas classes: (i) serviços críticos, I/O (Input/Output) e Tempo real; e (ii) serviços não críticos, Consultas e Armazenamento. Os serviços críticos recebem prioridade alta (2), já os serviços da classe não crítica recebem prioridade média (1) ou baixa (0).

Definimos I/O (Input/Output) como serviço crítico com base no trabalho de [Wang et al. 2019], que trata a necessidade de interação com o usuário final como um alto nível de criticidade, tendo peso mais significativo na experiência geral do usuário. Desse modo, a necessidade de I/O pode sugerir prioridade sobre outros recursos. O critério para o serviço de tempo real se baseia também no mesmo principio de I/O, um serviço que influencia na experiência do usuário e que requer respostas rápidas, mas também pelo fato de ter necessidade de resposta rápida. Diferentemente de serviços de consulta, onde o serviço disponibiliza apenas informações para o usuário e armazenamento na qual o dado será armazenado na Borda e/ou posteriormente enviado para a Nuvem. As solicitações que requerem utilização da Nuvem, geralmente são serviços que tem baixa necessidade de resposta urgente ou necessitam de algum processamento mais complexo, tendo em vista que a Nuvem se localiza distante dos dispositivos finais [Hong and Varghese 2019].

Para atender as requisições (R_i) vindas dos dispositivos finais, pesos foram atribuídos com o intuito de equilibrar as prioridades dos serviços e os recursos necessários (RN). Os serviços que demandam mais recursos (RN), como RAM e CPU, terão prioridade alterada em relação a um serviço que necessita somente de CPU. Isso considera também os recursos de borda (RB) disponíveis em tempo real. É importante considerar que um serviço prioritário que requer o uso de mais de um recurso dos nós pode causar uma fila de espera em relação a outros serviços prioritários que utilizam apenas um recurso de borda e que podem ser atendidos rapidamente. É a mesma situação que ocorre nos sistemas operacionais com tarefas bloqueantes, que são aquelas que bloqueiam o uso dos recursos até que sua solicitação seja totalmente atendida.

Uma fila extensa de solicitações pode ocasionar a indisponibilidade do sistema na totalidade, razão pela qual as requisições recebem pesos adicionais com relação aos recursos necessários (RN). A má utilização dos recursos de borda (RB) existentes, pode levar a indisponibilidade do nó. Portanto, a regra de atribuição de pesos às requisições baseadas em recursos é a seguinte:

$$Requisição_{peso} = (0.3 * Q) + N \begin{cases} Q = 0 & \text{se Recursos Necessários (RN)} \geq 2 \\ Q = 1 & \text{senão} \end{cases} \quad (1)$$

O valor de Q refere-se a quantidade de recursos necessários (RN) por parte da requisição que assume valores diferentes dependendo da quantidade de recursos. É uma forma de evitar um consumo demasiado por parte de uma única requisição. Se ela necessita de mais recursos da borda, ela pode ter sua prioridade reduzida. Requisições que demandam menos recursos, recebem 30% a mais de prioridade em relação a uma que necessita de mais recursos. Desse modo, evita-se a indisponibilidade do nó de borda, pois um excesso de recursos para uma única requisição poderia atender múltiplas requisições. Na equação, N é uma variável de adição na qual assume valores diferentes para Serviço Crítico (0.1 e 0.3) e Serviço Não Crítico (0.5 e 0.7). Esses valores variam também do tempo de fila decorrido desde a solicitação. Como resultado da Equação 1, quanto mais alto é o peso da requisição, a requisição é mais prioritária.

Testamos empiricamente os valores de N, os quais se mostraram satisfatórios de modo a definir pesos para as requisições fazendo um balanceamento da equação dependendo do tempo decorrido e da quantidade de recursos necessária (RN). É comum utilizar abordagens empíricas para definir pesos e parâmetros em modelos ML, principalmente quando não há um conhecimento inicial da equação que maximiza seu resultado.

Com a adição das regras definidas, as prioridades podem ser estabelecidas para o classificador aprender, testar e executar novas classificações para dados desconhecidos.

4. Metodologia

Esta seção detalha a metodologia adotada neste trabalho, implementação dos classificadores e experimentos.

4.1. Implementação

Com base nos trabalhos correlatos [Rusman et al. 2019, Ribeiro et al. 2020], foram escolhidos os classificadores KNN, SVM e *Decision Tree* para fins de comparação. Implementamos em *Python* os três algoritmos de classificação. Estes algoritmos não possuem um custo computacional elevado, sendo benéfico para nossa abordagem, que necessita de um tempo curto para definição de prioridades, além de conseguirem lidar com problemas com múltiplos rótulos.

Para o experimento, implementamos a simulação com o modelo preditivo *Decision Tree*, o qual teve melhor desempenho em termos de acurácia e erro, para possibilitar a criação da fila de prioridades das requisições. Os resultados deste classificador tanto do modo estatístico, avaliando sua acurácia e o erro geral, como também no modo analítico de modo a ver como esse possibilitou uma alocação e gerencia mais eficiente os recursos com base no consumo de CPU dos nós de Borda.

4.2. Experimentos

Os experimentos foram conduzidos em uma máquina física, com Linux, processador i7-7560U, 8 GB de memória RAM e com contêineres conectados a uma rede local via Wi-Fi.

As requisições dos dispositivos IoT foram geradas por um *script Python*, na qual simula as requisições por parte do cliente e as envia para o contêiner do nó classificador.

Para iniciar a simulação, todas as configurações especificadas no arquivo *Docker Compose* foram usadas, incluindo a definição de contêineres, o número de nós de borda (no caso, seis nós), latência dos nós, capacidade e tipo de comunicação entre eles. Os experimentos visam avaliar o desempenho do modelo de classificação dos algoritmos apresentados, referente a sua acurácia e erro. Foram simulados também quatro tipos de serviços (Armazenamento, Consulta, I/O e Processamento em tempo real), além de recursos disponíveis para os tipos de serviço (CPU, RAM e Disk).

O modelo com melhor desempenho (*Decision Tree*) foi alocado em um contêiner especifico da simulação para classificar as prioridades das requisições recebidas. Os dispositivos IoT enviam diferentes tipos de solicitações para executar tarefas, como acionamento, armazenamento e processamento. Toda a comunicação entre os contêineres é feita através *MQTT* (*Message Queuing Telemetry Transport*), protocolo que permite o transporte de mensagens de publicação/assinatura.

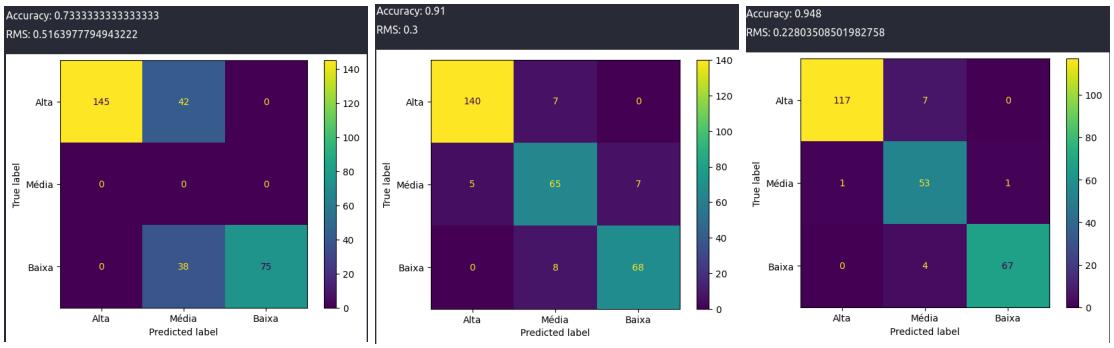
Por fim, comparamos a alocação de recursos de nós de borda utilizando o mecanismo de classificação de prioridades proposto neste trabalho com um ambiente sem a proposta, na qual as requisições são alocadas nos nós de borda segundo a menor latência. Vale salientar que todos os nós de borda possuem as mesmas quantidades de recursos disponíveis e que os modelos apresentados utilizaram a mesma base de dados para não haver interferência no resultado.

5. Resultados

Na execução do experimento, as requisições R_i de dispositivos simuladas pelo *script Python*, contendo as *features* anteriormente mencionadas, são as entradas do classificador. O conjunto de dados foi separado em *train/test/validation*, na qual o classificador foi treinado com 70% desses dados, e os outros 30% foram separados para teste. A validação é feita a partir de 1000 requisições simulando uma aplicação de software sintético. As regras estabelecidas anteriormente foram aplicadas no classificador para aprendizado da forma correta de classificar a prioridade de cada tipo de serviço.

O resultado de acurácia foi de aproximadamente 73% para o modelo *SVM*, 91% para o modelo *KNN* e 94% para o modelo *Decision Tree*, respectivamente, que indica o desempenho geral do modelo. Esses resultados indicam o quanto o classificador acertou em sua classificação, levando em consideração a base de dados fornecida para o aprendizado, na qual mapeamos requisições e suas prioridades com bases nas *features* apresentadas anteriormente. O *RMSE (Root Mean Squared Error)* obtido foi de 0.51, 0.30 e 0.22 para os modelos *SVM*,*KNN* e para o *Decision Tree*, respectivamente. O *RMSE* é uma medida que calcula ”a raiz quadrática média”dos erros entre valores observados (reais) e predições (hipóteses) do modelo. Através dos resultados obtidos dos modelos analisados, pode-se gerar a matriz de confusão (Fig. 2) como uma métrica de validação dos resultados.

Uma matriz de confusão é uma tabela que indica os erros e acertos do modelo, comparando com o resultado esperado (ou etiquetas/*labels*) dos dados. Esta matriz mostra quantos exemplos existem em cada grupo: falso positivo (FP), falso negativo (FN), verdadeiro positivo (TP) e verdadeiro negativo (TN). É interessante visualizar a quantidade desses grupos tanto em números absolutos quanto em porcentagem da classe real,



a) SVM *b) KNN* *c) Decision Tree*

Figura 2. Métricas de Avaliação dos algoritmos de classificação

pois o número de exemplos em cada classe pode variar. Analisando a Figura 2, é possível observar que os modelos *Decision Tree e KNN* tiveram melhor desempenho que o *SVM*.

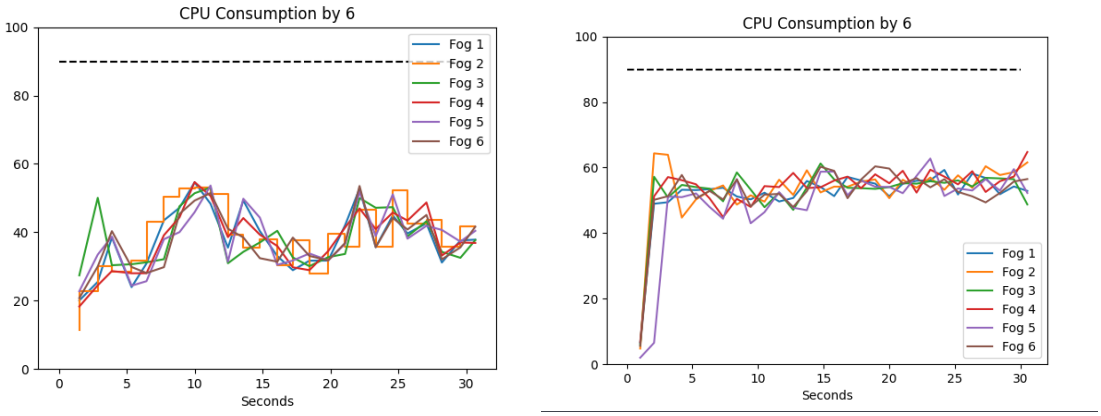
O *Decision Tree* obteve os melhores resultados. Um fator interessante que pode alterar o desempenho do modelo são seus hiperparametros. Pode-se 'tunar' os modelos com base nos hiperparametros a fim de melhorar os resultados, como, por exemplo, escolher um K diferente no modelo *KNN* ou um *min-samples-split* no *Decision Tree*, que se refere ao número mínimo de amostras que um nó deve possuir antes da divisão.

A utilização de recursos dos nós de borda é monitorada usando a API '*docker stats*'. Em seguida, comparamos a utilização desses recursos com base no mecanismo de classificação, a fim de validar a técnica de priorização. Para isso, executamos contêineres distintos, sendo um específico para execução da classificação de prioridades. Após as prioridades serem estabelecidas, uma *Thread* aloca essas requisições nos nós de borda. Durante a simulação, as requisições foram alocadas com base na tabela de prioridades fornecida pelo classificador e avaliamos a métrica de uso de CPU (Figura 3), visando alcançar um gerenciamento eficiente do uso dos recursos dos nós de borda (os quais são referenciados por nós *fog* nos gráficos dos resultados).

A alocação das requisições no ambiente sem a proposta leva em consideração apenas o requisito de latência, visando a menor latência possível, mas sem um controle do uso dos recursos da borda.

Na Figura 3, pode-se observar que o uso da proposta de classificação mostra-se eficiente, variando em média entre 30% e 50% da utilização de CPU. Enquanto no ambiente sem o uso de nossa proposta, a utilização de CPU chega em média a 60% da sua capacidade, com picos que ultrapassam essa faixa. Elevar o consumo do recurso pode ser crítico ao ponto de causar indisponibilidade por parte do nó, o que pode comprometer o funcionamento da rede.

O algoritmo utilizado no experimento irá interferir no tempo total de execução da simulação, devido à necessidade de baixa latência no ambiente de Borda. Isso é um fator crucial a ser observado para escolher o modelo preditivo além da acurácia, tendo em vista as diferentes complexidades dos algoritmos de classificação.



a) Uso do Classificador Decision Tree *b) Ambiente sem a proposta*

Figura 3. Porcentagem de uso CPU - 6 fogs e 1000 requisições

6. Conclusão

Neste trabalho propomos uma classificação de prioridades de requisições de dispositivos IoT usando um modelo de aprendizado de máquina, a fim de permitir uma alocação eficiente de recursos em nós de borda, dando prioridade às tarefas mais importantes ou que podem ser concluídas rapidamente.

O modelo de classificação de prioridades, juntamente com o monitoramento dos recursos disponíveis, possibilita um menor consumo dos recursos presentes na Borda da rede, o que é benéfico tendo em vista que recursos na borda são extremamente limitados. Como contribuição do trabalho, também definimos as regras de adição de pesos de prioridades para os diferentes tipos de serviços solicitados pelos dispositivos finais.

Os resultados mostram que o modelo preditivo *Decision Tree* obteve o melhor desempenho entre os modelos comparados (KNN e SVM), com uma acurácia de 94% de precisão e um RMSE = 0.22 na classificação de prioridade das requisições. O *Decision Tree* foi, portanto, selecionado para fornecer a fila de prioridades das requisições que foram alocadas nos nós mais adequados. O uso da CPU tornou-se mais eficiente utilizando nossa proposta de classificação de prioridades frente a uma alocação que leva em consideração apenas a menor latência, sem o controle de recursos de borda.

Como trabalhos futuros, pretende-se avaliar a proposta em termos de escalabilidade, com mais requisições e com dados de aplicações reais. Além de modificar os hiperparametros dos classificadores, pois isso permitirá o modelo prevenir de aprender apenas com os dados mostrados (*overfitting* e *underfitting*), podendo generalizar para outras situações reais. Também pretende-se maximizar a eficiência da adição de pesos nas prioridades para diferentes quantidades de recursos e adicionais tipos de serviços.

Agradecimentos

Este trabalho foi realizado com o apoio da Fundação Cearense de Apoio ao Desenvolvimento científico e Tecnológico (FUNCAP) através do processo MLC-0191-00164.01.00/22.

Referências

Bhushan, S. and Mat, M. (2021). Priority-queue based dynamic scaling for efficient resource allocation in fog computing. In *2021 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pages 1–6.

Bui, T. B., Sakr, A., Castrillón, J., and Schuster, R. (2021). Six-factors score-based matchmaking based on priority and preemption for resource allocation in edge computing. In *2021 IEEE International Conference on Edge Computing (EDGE)*, pages 44–50.

Coutinho, A. A., Carneiro, E., and Greve, F. (2016). *Computação em Névoa: Conceitos, Aplicações e Desafios*, pages 266–315.

Gupta, H., Dastjerdi, A., Ghosh, S., and Buyya, R. (2016). ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47.

Hong, C.-H. and Varghese, B. (2019). Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. *ACM Comput. Surv.*, 52(5).

Jr., J. B. and Araújo, A. (2021). Uma análise sobre gerenciamento de recursos na computação em névoa. In *Anais da IV Escola Regional de Alto Desempenho do Centro-Oeste*, pages 7–11, Porto Alegre, RS, Brasil. SBC.

Li, X., Liu, Y., Ji, H., Zhang, H., and Leung, V. C. M. (2019). Optimizing resources allocation for fog computing-based internet of things networks. *IEEE Access*, 7:64907–64922.

Madej, A., Wang, N., Athanasopoulos, N., Ranjan, R., and Varghese, B. (2020). Priority-based fair scheduling in edge computing. In *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, pages 39–48.

Ribeiro, F. M., Prati, R., Bianchi, R., and Kamienski, C. (2020). A nearest neighbors based data filter for fog computing in iot smart agriculture. In *2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, pages 63–67.

Rusman, J., Tahir, Z., and Salam, A. E. U. (2019). Fog computing concept implementation in work error detection system of the industrial machine using support vector machine (svm). In *2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, pages 160–164.

Sharif, Z., Jung, L. T., Razzak, I., and Alazab, M. (2023). Adaptive and priority-based resource allocation for efficient resources utilization in mobile-edge computing. *IEEE Internet of Things Journal*, 10(4):3079–3093.

Statista (2020). Data volume of iot connected devices worldwide 2019 and 2025. https://www.statista.com/statistics/1017863/worldwide-iot-connected-devices-data-size/. Acessado em 10/02/2023.

Wang, K., Tan, Y., Shao, Z., Ci, S., and Yang, Y. (2019). Learning-based task offloading for delay-sensitive applications in dynamic fog networks. *IEEE Transactions on Vehicular Technology*, 68(11):11399–11403.

Zhao, Z., Barijough, K. M., and Gerstlauer, A. (2018). Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2348–2359.