SIMPEX + 4º MOBICIT
CONEXÕES QUE MOVEM o FUTURO
Energia e Inteligência para Cidades Vivas
Realização: UniSENAI, Instituto SENAI de Inovação, Instituto SENAI de Tecnologia, fapesc
Apoio: Even3

POSICIONAMENTO DE APLICAÇÕES CONTEINERIZADAS EM ECC UTILIZANDO A HEURÍSTICA DE *SCORING*

Mateus Unulino dos Passos[1*]
[1]*Laboratório de Processamento Paralelo e Distribuído (LabP2D) - UDESC/CCT*

1. Introdução

A ubiquidade de aplicações conteinerizadas trouxe à tona a necessidade de estratégias de provisionamento inteligentes, automatizadas e com sensibilidade ao contexto para ecossistemas computacionais modernos. Dentre estes ecossistemas encontra-se o *Edge-Cloud Continuum*, que agrega dispositivos na *edge* - *hardware* restrito próximo às fontes de dados - nós de computação na *fog* e *data-centers* na *cloud* [1]. Cada camada do *ECC* possui um *trade-off* distinto: dispositivos da *edge* minimizam latência e mantêm localidade de dados, mas oferecem capacidade de processamento limitada, já nós de computação na *cloud* invertem este cenário.

A orquestração de *workloads* conteinerizadas é ostensivamente documentada na literatura [1][3] e soluções como *Kubernetes Scheduler Framework* [2] elegem um nó dentro de um cluster, contemplando primariamente recursos computacionais (utilização de CPU e Memória disponíveis, ocasionalmente armazenamento). Entretanto, estas soluções não reconciliam as especificações de usuário da *workload* com as características inerentes de cada camada do *continuum.*

Este trabalho descreve uma abordagem para o posicionamento de aplicações conteinerizadas em uma plataforma *ECC* utilizando a heurística de *Scoring* para selecionar a camada (*edge, fog* ou *cloud*) mais adequada. Partindo da especificação de usuário - um documento declarativo na forma de JSON - as camadas do *continuum* recebem sucessivamente pontuações que refletem sua adequação frente ao critério contemplado. Uma camada é escolhida e então um manifesto YAML é gerado para que a escolha intra-*cluster* seja feita pelo *kube-scheduler* [2], o escalonador padrão do Kubernetes.

2. Heurística e Procedimento Experimental

Frameworks de orquestração multi-objetivo como *IslandRun* [4] abordaram a tensão entre desempenho, localidade de dados e custo para inferência distribuída de modelos de IA. Em outros trabalhos [5], localidade de dados e latência têm sido identificados como o principal contraponto na escolha entre *edge* e *cloud.* Neste trabalho, quatro requisitos foram elencados: latência (*sensitive* | *none*), privacidade de dados (*on-premise* | *none*), poder computacional (*high* | *medium* | *low*) e conectividade com a *cloud* (*intermittent* | *stable*).

Para cada requisito i e cada camada L, uma função de *scoring* S(L, i) retorna uma pontuação: +3 (favorece consideravelmente L), +1 (favorece pouco L), e -2 (penaliza L). A pontuação agregada para a camada é então contabilizada sendo simplesmente a soma de todas as pontuações. Na Tabela 1 é possível conferir um exemplo do procedimento descrito para um cenário específico descrito pelo usuário através do JSON de entrada.

Exemplo de Pontuação

Critério (valor)	Edge	Fog	Cloud
Latência (sensitive)	3	1	-2
Data Privacy (on-premise)	3	3	-2
Compute (high)	-2	1	3
Conectivity (stable)	-2	1	3

Tab. 1. *Pontuação para o cenário de uma aplicação sensível à latência e privacidade de dados, que necessita de poder de processamento elevado e conectividade estável com a cloud.*

*mateus.passos1605@edu.udesc.br

3. Resultados

O procedimento de *Scoring* foi validado com três *workloads* correspondendo à modelos de IA. Na Figura 2, pode-se observar que, utilizando o escalonador padrão do Kubernetes, foi atribuído o mesmo nó computacional à carga heterogênea de modelos. Já a abordagem proposta atribuiu ao contêiner que realiza o treinamento de um modelo SVM o tier da *cloud;* a inferência, realizada pelo modelo *icame-detect-1000*, foi corretamente posicionada na *edge*, devido à proximidade dos dados e o modelo icame-storage foi adequadamente delegado à *fog*.

Teste de Posicionamento

Application	Layer Scoring (Proposed Approach)	Kubernetes Scheduler (Default)
svm-train	cloud	ecc-fog
icame-detect-1000	edge	ecc-fog
icame-storage	fog	ecc-fog

Fig. 2. *O escalonador padrão posicionou aplicações com requisitos distintos no mesmo nó..*

4. Referências

[1] Belcastro, L., Marozzo, F., Orsino, A., Talia, D., and Trunfio, P. (2025). Navigating the Edge-Cloud Continuum: A State-of-Practice Survey. arXiv:2506.02003.
[2] Kubernetes Authors. (2024). Kubernetes Scheduler. Official Documentation. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/
[3] Scheduling of Distributed Applications on the Computing Continuum: A Survey. (2023). arXiv:2405.00005. ACM/IEEE
[4] Malepati, B. S. S. A. (2025). IslandRun: Privacy-Aware Multi-Objective Orchestration for Distributed AI Inference. arXiv:2512.00595.
[5] Wang, Z., Goudarzi, M., Gong, M., et al. (2024). Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments. Future Generation Computer Systems, 152, pp. 55–69.

*mateus.passos1605@edu.udesc.br