Escalonamento Horizontal em Kubernetes com Redes Neurais Artificiais para Predição de Carga

Lucileide M. D. da Silva*†, Sérgio N. Silva* e Marcelo A. C. Fernandes*‡
*InovaAi Lab, nPITI/IMD, Universidade Federal do Rio Grande do Norte (UFRN), Natal, RN, Brasil
†Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, Santa Cruz, RN, Brasil
‡Departamento de Engenharia da Computação e Automação, UFRN, Natal, RN, Brasil
lucileide.dantas@ifrn.edu.br, s.natansilva@gmail.com e mfernandes@dca.ufrn.br

***Resumo*—Este artigo apresenta uma abordagem inovadora para o escalonamento horizontal automático em cluster Kubernetes (K8s), utilizando Redes Neurais Artificiais. A proposta é chamada de ANN-HS e em comparação com o Escalonador Horizontal padrão do K8s (HPA), o ANN-HS demonstra eficiência superior em termos de consumo de recursos, alocação otimizada de réplicas, adaptação flexível à demanda e aderência a níveis de latência. Com modelos de regressão pré-treinados, o ANN-HS oferece ajuste personalizado de recursos, promovendo uma alternativa promissora para aprimorar o escalonamento horizontal de aplicações em ambientes de micro-serviços.**

***Index Terms*—Escalonamento horizontal, Kubernetes, Aprendizagem de Máquina, Redes Neurais Artificiais, HPA**

I. Introdução

O escalonamento horizontal em clusters Kubernetes (K8s) é um desafio crucial para garantir a eficiência e o desempenho de aplicações em ambientes de nuvem [1]–[3]. Com o aumento da complexidade e da demanda do tráfego HTTP, é essencial adotar abordagens inovadoras para ajustar dinamicamente o número de réplicas de uma aplicação em execução no cluster. Neste contexto, a utilização de técnicas de aprendizagem de máquina (*Machine Learning* - ML) tem se destacado como uma solução promissora para prever a carga associada ao tráfego HTTP, permitindo o escalonamento horizontal com base em informações em tempo real.

Vários trabalhos na literatura tem feito contribuições significativas em escalonamento horizontal associado ao K8s. Em [3] é proposta uma abordagem para o dimensionamento automático de aplicações serverless usando Aprendizado por Reforço (RL), garantindo a Qualidade de Serviço (QoS) e o uso eficiente de recursos. Em [4] é apresentado o IScaler, uma abordagem para o dimensionamento de recursos em ambientes de nuvem, utilizando O algoritimo Q-learning de Aprendizagem por Reforço. O trabalho apresentado em [5] descreve um sistema extensível para o dimensionamento automático de microsserviços em K8s, integrando o escalonador DScaler baseado em Aprendizado por Reforço Profundo (DRL). O estudo apresentado em [6] investiga o uso de DRL para o dimensionamento de instâncias UPF em redes 5G/6G core. No trabalho apresentado em [7], é proposto um algoritmo genérico de dimensionamento automático para identificar a demanda de recursos de microsserviços em aplicações em nuvem, resultando em melhorias significativas no tempo de resposta e fornecendo uma solução personalizada de dimensionamento automático com mínimo envolvimento do usuário.

Este trabalho propõe uma solução baseada em Redes Neurais Artificiais (ANN) para a predição de carga em aplicações HTTP em um ambiente K8s, chamada aqui de *Artificial Neural Network Horizontal Scaling* (ANN-HS). A ANN é treinada com um dataset contendo informações de carga de CPU por réplica, taxa de pacotes recebidos por minuto (rpm) por réplica e número de réplicas em diferentes cenários de tráfego. A partir desses dados, a ANN é capaz de fazer previsões de carga e em seguida, esse valor é convertido em um novo número de réplicas, permitindo o ajuste dinâmico e automático do dimensionamento horizontal do cluster.

Para avaliar a eficácia da abordagem proposta, foram realizados experimentos em um cluster Kubernetes implementado em duas máquinas virtuais distintas. Utilizando o jMeter para gerar demandas de tráfego HTTP e o Prometheus para coletar as métricas do cluster, o sistema de predição de carga baseado na ANN foi comparado ao Horizontal Pod Autoscaler (HPA) do Kubernetes. Os resultados desses testes forneceram insights valiosos sobre o desempenho e a eficiência da abordagem de aprendizagem de máquina em relação às estratégias tradicionais de escalonamento horizontal, demonstrando o potencial dessa técnica para otimizar o gerenciamento de recursos em ambientes Kubernetes.

II. Descrição da Proposta

A Figura 1 descreve a arquitetura do ANN-HS, no qual uma ANN é utilizada para predizer a carga em uma aplicação web, possibilitando o controle do escalonamento horizontal em um cluster Kubernetes (K8s). Em cada instante de tempo n-ésimo, o ANN-HS recebe como entrada as variáveis $x^{\mathrm{cpu}}(n)$, $x^{\mathrm{rp}}(n)$ e $m^{\mathrm{p}}(n)$, que representam, respectivamente, a carga de CPU em mile cores, a taxa de pacotes recebidos por minutos (*receive packets per minutes* - rpm), ambas normalizadas pelo número de réplicas da aplicação, e o número de réplicas de uma dada aplicação. As variáveis $x^{\mathrm{cpu}}(n)$ e $x^{\mathrm{rp}}(n)$ são geradas por um módulo chamado aqui de *Normalization Module* (NM), que captura as métricas da aplicação através de um serviço de monitoramento (*Monitoring Service* - MS) associado ao cluster K8s. O MS é responsável por capturar as métricas relacionadas à carga de CPU de todas as réplicas da aplicação, $m^{\mathrm{cpu}}(n)$, a taxa de pacotes recebidos por todas as

réplicas da aplicação em rpm, $m^{\mathrm{rp}}(n)$, e o número de réplicas da aplicação, $m^{\mathrm{p}}(n)$. A cada instante n-ésimo, o ANN-HS produz a saída na variável $y(n)$, que representa um valor de predição de carga na aplicação web. O valor predito de carga no instante n-ésimo, $y(n)$, é utilizado para calcular o próximo número de réplicas a ser atingido pelo K8s, $m^{\mathrm{p}}(n+1)$. O cálculo do próximo número de réplicas é realizado por um módulo chamado *Load-to-Replica Converter Module* (LRCM), que pode utilizar desde uma simples conversão linear até uma métrica mais complexa para indicar o próximo número de réplicas com base na carga predita pela ANN. O ajuste do número de réplicas é feito diretamente através da API do K8s.

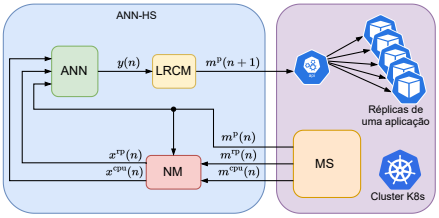

Figura 1. Arquitetura ANN Horizontal Scaling (ANN-HS).

As variáveis $x^{\mathrm{cpu}}(n)$ e $x^{\mathrm{rp}}(n)$ podem ser expressas como

$$x^{\mathrm{cpu}}(n) = \frac{m^{\mathrm{cpu}}(n)}{m^{\mathrm{p}}(n)} \tag{1}$$

e

$$x^{\mathrm{rp}}(n) = \frac{m^{\mathrm{rp}}(n)}{m^{\mathrm{p}}(n)}. \tag{2}$$

Já a variável $m^{\mathrm{p}}(n+1)$ pode ser expressa como

$$m^{\mathrm{p}}(n+1) = f(y(n)) \tag{3}$$

onde $f(\cdot)$ é uma função que mapeia o carga predita em número de réplicas a requisitado ao K8s. A variável $m^{\mathrm{p}}(n+1)$ possui um valor inteiro entre 1 e $m^{\mathrm{p}}_{\mathrm{max}}$, onde $m^{\mathrm{p}}_{\mathrm{max}}$ representa o número máximo de replicas. Tomando como base a Figura 1, as Equações 1 e 2 são implementadas pelo NM e a Equação 3 é implementada pelo LRCM.

A ANN irá desempenhar um papel crucial na predição de carga no K8s. Ao receber as métricas de carga de CPU e a taxa de pacotes recebidos por minuto (rpm), normalizadas pelo número de réplicas, a ANN é treinada para estimar a carga atual da aplicação. Com base na carga predita pela ANN, o sistema pode tomar decisões inteligentes sobre o ajuste do número de réplicas, promovendo o escalonamento horizontal sob demanda. A ANN possibilita que o sistema seja mais adaptativo e responsivo às flutuações de tráfego, otimizando o consumo de recursos e melhorando o desempenho da aplicação no ambiente do K8s.

III. Metodologia

Neste trabalho, foram realizadas duas metodologias distintas para explorar e validar a proposta de escalonamento horizontal baseada em ANN em ambientes K8s. A primeira metodologia abordou o treinamento da ANN para predição de carga em uma aplicação web em execução em um cluster K8s. Já a segunda foi focada na avaliação e teste da proposta em execução no K8s. Para ambas as metodologias o ambiente de experimentação consistiu em um K8s do tipo Micro K8s [9] implementado em uma máquina virtual (*Virtual Machines* - VMs) chamada VM Cluster (VMC), com 4 CPUs, 12GB de memória e Ubuntu 20.04 server, e outra VM chamada VM *Load Generator* (VMLG) com 3 CPUs, 12 GB de memória e e Ubuntu 22.04 server, que é usada para gerar demandas de tráfego HTTP na aplicação em execução no K8s. O jMeter [10] foi empregado para gerar demandas de tráfego HTTP, enquanto o Prometheus [13] foi utilizado para capturar métricas relevantes do cluster. A aplicação web é implementada em Java usando o Dropwizard e executa uma sequência Fibonacci de ordem 15 em resposta a cada solicitação feita pelo jMeter na aplicação web do K8s. O host é uma máquina com 128 GB de memória RAM 16 CPUs com Windows 10.

A. Treinamento da ANN

1) Criação do dataset: A Figura 2 detalha o esquema utilizado para geração do dataset de treinamento da ANN. Foram realizados $K = 50$ experimentos variando a carga (número de usuários) em uma faixa de 1 a 99, com intervalos de 2 em 2, ou seja, cada k-ésimo experimento de carga possui N_k usuários onde $N_k \in \{1, 3, \ldots, 97, 99\}$ para $k = 1, \ldots, K$. Cada k-ésimo experimento de carga com N_k usuários, foi executado $V = 20$ vezes variando o número de réplicas, R_i, da aplicação web no K8s, de 1 a 20, com passo 1, ou seja, $R_i \in \{1, 2, \ldots, 19, 20\}$ para $i = 1, \ldots, V$. Ao final foram realizados $K \times V = 1000$ experimentos, no qual, cada j-ésimo experimento, $\mathbf{e}_j$ pode ser expresso como

$$\mathbf{e}_j = \mathbf{e}_{k,i} = \{N_k, R_i\} \text{ para } j = 1, \ldots, K \times V, \tag{4}$$

onde

$$k = \left\lfloor \frac{j-1}{V} \right\rfloor + 1 \tag{5}$$

e

$$i = j - V \times (k-1). \tag{6}$$

Ambas variáveis $\mathbf{e}_j$ e $\mathbf{e}_{k,i}$ representam o experimento, para geração do dataset, com o k-ésima valor de número de usuários e o i-ésimo valor de número de réplicas. O controle de cada experimento foi realizado por um módulo de controle chamado aqui *Experiment Controller Module* (ECM). Como ilustrado na Figura 2, $u_{k,i}$ representa o usuário virtual criado pelo jMeter associado ao experimento $\mathbf{e}_{k,i}$.

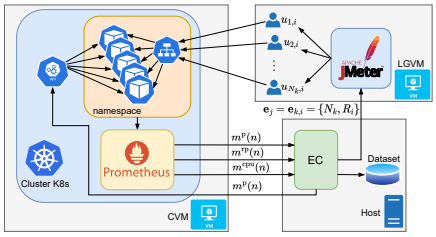

Figura 2. Esquema utilizado para criação do dataset para predição de carga.

Cada j-ésimo experimento, $\mathbf{e}_j$, (ver Equações 4, 5 e 6) utilizou o perfil de carga ilustrado na Figura 3. Este perfil foi construído usando o *Open Model Thread Group* que define a quantidade de usuários virtuais criados ao longo do tempo pelo jMeter para gerar a demanda de trafego na aplicação web sendo executada no K8s [10].

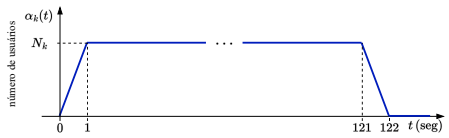

Figura 3. Perfil de criação de usuários virtuais pelo jMeter usando *Open Model Thread Group* para geração do dataset. Cada j-ésimo experimento, $\mathbf{e}_j$, (ver Equação 4) usou este perfil.

Com uma duração de 122 segundos, o perfil de carga para criação de usuários, é caracterizado por um subida em rampa no primeiro 1 segundo até atingir uma quantidade N_k usuários. Entre 1 e 121 segundos as requisições ficam constantes em N_k e após 121 segundos a quantidade de usuários decaem em rampa (seguindo a mesma taxa de subida) até os 122 segundos. A quantidade de usuários virtuais em um dado tempo, $\alpha_k(t)$, pode ser expressa como

$$\alpha_k(t) = \begin{cases} N_k t & \text{se } 0 < t \leq 1, \\ N_k & \text{se } 1 < t \leq 121, \\ -N_k t + 122 N_k & \text{se } 120 < t \leq 122, \\ 0 & \text{se } t > 122, \end{cases} \tag{7}$$

onde N_k é numero máximo de usuários virtuais que o perfil de carga pode atingir para o k-ésimo valor de carga associado ao j-ésimo experimento, $\mathbf{e}_j$ (ver Equações 4, 5 e 6).

A partir da realização dos experimentos foram criados 1000 conjuntos de dados capturados ao longo de 122 segundos (a cada 2 segundos). Assim, cada j-ésimo experimento $\mathbf{e}_j$ gerou um conjunto, $\mathbf{C}_j$ que pode ser expresso como

$$\mathbf{C}_j = \mathbf{C}_{k,i} = [\mathbf{c}_{k,i,1}, \ldots, \mathbf{c}_{k,i,v}, \ldots, \mathbf{c}_{k,i,M_{k,i}}] \tag{8}$$

onde $M_{k,i}$ é a quantidade de amostras capturadas em cada j-ésimo experimento (ver Equações 4, 5 e 6) e $\mathbf{c}_{k,i,v}$ é a v-ésima amostra capturada j-ésimo experimento que é expressa como

$$\mathbf{c}_{k,i,v} = [x^{\text{cpu}}_{k,i,v}, x^{\text{rp}}_{k,i,v}, m^{\text{p}}_{k,i,v}] \tag{9}$$

onde $x^{\text{cpu}}_{k,i,v}$, $x^{\text{rp}}_{k,i,v}$ e $m^{\text{p}}_{k,i,v}$ representam a carga de cpu por réplica (ver Equação 1) usada pela aplicação em mile cores, taxa de pacotes recebidos por réplica (ver Equação 2) da aplicação em rpm e o número de réplicas, respectivamente. Ao final dos experimentos obteve-se um dataset de aproximadamente 61 mil linhas.

2) Pré-processamento: A etapa de pré-processamento desempenha um papel fundamental no preparo dos dados antes do treinamento da ANN. Nessa etapa, foram realizadas duas operações essenciais: a remoção de outliers e o cálculo da média de consumo de CPU e pacotes recebidos por experimento.

Após a criação do dataset com cerca de 61 mil linhas, foram removidos *outliers* com valores acima do percentil 95%. Essa abordagem eliminou valores extremamente atípicos e pouco representativos, garantindo a integridade dos dados utilizados no treinamento da ANN.

Após a limpeza dos outliers, o próximo passo foi calcular a média de consumo de CPU por réplica em mile core e taxa de pacotes recebidos por réplica em rpm para cada j-ésimo experimento. Ao calcular a média de consumo desses recursos em cada j-ésimo experimento, obtém-se um valor central que suaviza flutuações temporárias e elimina possíveis ruídos nos dados. Essa abordagem proporciona uma visão mais estável e confiável dos padrões de carga média da aplicação web em diferentes cenários de tráfego. Como resultado, a ANN é treinada com informações mais consistentes e relevantes, permitindo que aprenda com maior precisão a relação entre as entradas e a carga predita.

Assim, foi gerado um novo conjunto para o treinamento da ANN que resultou em um dataset de $K \times V = 1000$ linhas, ou seja, cada j-experimento contribuio para uma linha no novo dataset. Cada j-ésima linha do novo conjunto de dados pode ser expressa como

$$\mathbf{g}_j = \mathbf{g}_{k,i} = [\bar{x}^{\text{cpu}}_{k,i}, \bar{x}^{\text{rp}}_{k,i}, m^{\text{p}}_{k,i}] \tag{10}$$

onde

$$\bar{x}^{\text{cpu}}_{k,i} = \frac{1}{M_{i,k}} \sum_{v=1}^{M_{i,k}} x^{\text{cpu}}_{k,i,v} \tag{11}$$

e

$$\bar{x}^{\text{rp}}_{k,i} = \frac{1}{M_{i,k}} \sum_{v=1}^{M_{i,k}} x^{\text{rp}}_{k,i,v}. \tag{12}$$

3) Treinamento: O ANN este estudo foi uma etapa crucial para a predição de carga e o escalonamento horizontal em ambientes K8s. Foi utilizada uma ANN do tipo MLP configurada como um modelo de regressão, visando prever um valor contínuo de carga com base nas três entradas fornecidas: carga de CPU por réplica em mile core, taxa de pacotes recebidos por réplica em rpm e o número de réplicas da aplicação em execução no cluster K8s. A arquitetura da ANN foi projetada com três camadas ocultas, cada uma contendo 200, 200 e 100 neurônios, respectivamente, e uma camada de saída. Essas camadas ocultas foram fundamentais para permitir que a ANN aprendesse os padrões complexos e não lineares associados à carga da aplicação, possibilitando uma modelagem mais precisa da relação entre as entradas e a carga predita. Durante o treinamento, a ANN foi ajustada iterativamente para minimizar o erro entre as previsões de carga e os valores reais. O algoritmo de regressão utilizado ajustou os pesos das conexões entre os neurônios, permitindo que a ANN aprendesse a relação entre as três entradas e a carga predita com maior precisão.

B. Metodologia associada aos testes da ANN

A Figura 4 detalha o esquema utilizado para o testes da proposta do ANN-HS. Foram realizados $H = 20$ experimentos variando a carga (número de usuários) em uma faixa de 5 a 100, com intervalos de 5 em 5, ou seja,

cada k-ésimo experimento de carga possui L_k usuários onde $L_k \in \{5, 10, \ldots, 95, 100\}$ para $k = 1, \ldots, H$. Cada k-ésimo experimento de teste pode ser representado pela variável $\mathbf{t}_k$.

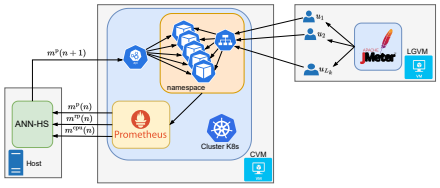

Figura 4. Esquema utilizado para o k-ésimo experimento de teste do ANN-HS.

Cada k-ésimo experimento de teste, $\mathbf{t}_k$, utilizou o perfil de carga ilustrado na Figura 5. Este perfil foi construído usando o *Open Model Thread Group* que define a quantidade de usuários virtuais criados ao longo do tempo pelo jMeter para gerar a demanda de trafego na aplicação web sendo executada no K8s [10].

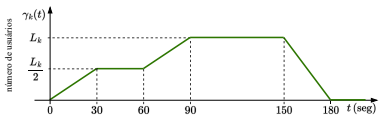

Figura 5. Perfil de criação de usuários virtuais pelo jMeter usando *Open Model Thread Group* para os testes do ANN-HS. Cada k-ésimo experimento de teste, $\mathbf{t}_k$, usou este perfil.

Com uma duração de 180 segundos, o perfil de carga para k-ésimo experimento de teste, $\mathbf{t}_k$, é caracterizado por um subida em rampa no primeiro 30 segundo até atingir uma quantidade $\frac{L_k}{2}$ usuários. Nos próximos 30 segundos o carga fica constante em $\frac{L_k}{2}$ usuários e ao atingir os 60 segundos a carga inicia uma segunda subida em rampa, de 30 segundos, até atingir L_k usuários. Após os 90 segundos a carga fica constante L_k usuários nos próximos 60 segundos. Chegando em 150 segundos a carga inicia um decaimento em rampa até zero em mais 30 segundos. A quantidade de usuários virtuais em um dado tempo, $\gamma_k(t)$, pode ser expressa como

$$\gamma_k(t) = \begin{cases} \frac{L_k t}{60} & \text{se } 0 < t \leq 30, \\ \frac{L_k}{2} & \text{se } 30 < t \leq 60, \\ \frac{L_k t}{60} - \frac{L_k}{2} & \text{se } 60 < t \leq 90, \\ L_k & \text{se } 90 < t \leq 150, \\ \frac{-N_k t}{30} + 6L_k & \text{se } 150 < t \leq 180, \\ 0 & \text{se } t > 180, \end{cases} \tag{13}$$

onde L_k é numero máximo de usuários virtuais que o perfil de carga pode atingir para o k-ésimo valor de carga associado ao k-ésimo experimento.

O ANN-HS, foi implementado e executado no próprio host do experimento, além das máquinas virtuais mencionadas anteriormente. Utilizando o ambiente de desenvolvimento Matlab (versão 596681), o ANN-HS fez uso da biblioteca Java Fabric8 Kubernetes-Client [12] para interagir com a API do K8s e realizar o escalonamento dinâmico das réplicas da aplicação web. Essa abordagem permitiu que o ANN-HS funcionasse como um componente externo ao cluster, tomando decisões de escalonamento com base nas métricas coletadas e enviando as requisições de escalonamento para o cluster K8s. As métricas necessárias para o processo de escalonamento foram coletadas utilizando o Prometheus [13], que atuou como o serviço de monitoramento (MS) conforme ilustrado na Figura 1. Neste trabalho foram utilizados $m^{\mathrm{p}}_{\max} = 20$ e uma função linear para mapear a carga predita em número de réplicas, no qual, a Equação 3 pode ser re-escrita como

$$m^{\mathrm{p}}(n+1) = \left\lfloor y(n) \times \frac{20}{100} \right\rceil. \tag{14}$$

Na metodologia de teste, o Horizontal Pod Autoscaler (HPA) nativo do K8s foi adotado como uma abordagem de comparação com o ANN-HS proposto. O HPA é uma funcionalidade nativa do K8s que permite ajustar automaticamente o número de réplicas de um pod com base em métricas de utilização, como a utilização de CPU ou métricas personalizadas [1], [2]. Os testes seguiram uma abordagem semelhante ao ANN-HS, onde a carga foi variada pelo número de usuários, conforme ilustrado no perfil de carga apresentado na Figura 5 e seguindo a Equação 13. A métrica utilizada para o HPA foi a carga de CPU por réplica com um valor médio de utilização de 10%. Ou seja, as réplicas serão escalonados para cima ou para baixo pelo HPA para manter a utilização média de recursos de CPU em torno de 10% da capacidade total disponível. O HPA foi implantado com os parâmetros: *averageUtilization:* 10, *minReplicas:* 1, *maxReplicas:* 20, *scaleDown: stabilizationWindowSeconds:* 0 e *scaleUp: stabilizationWindowSeconds:* 0. Essa comparação entre o ANN-HS e o HPA permitiu uma análise mais abrangente e detalhada do desempenho e eficácia de cada abordagem de escalonamento horizontal. Os resultados obtidos foram fundamentais para identificar as vantagens e limitações de cada solução e fornecer informações valiosas para o aprimoramento da escalabilidade e adaptabilidade das aplicações em ambientes de nuvem.

IV. Resultados

A. Treinamento da ANN

A Figura 6 mostra a relação da valor predito de carga para todos experimentos, y, e o valor verdadeiro carga observado no dataset, N. A raiz do erro quadrático médio (*Root Mean Squared Error* - RMSE) foi calculada como 0,9431. O erro quadrático médio (*Mean Squared Error* - MSE) foi calculado como 0,8895. O coeficiente de determinação (R^2) foi calculado como 0,9989, que é uma métrica estatística que indica a proporção da variância na variável de resposta que é previsível a partir das variáveis independentes. Um valor próximo de 1 indica que o modelo explica muito bem a variabilidade dos dados. Finalmente, a Média do Erro Absoluto (*Mean Absolute Error* - MAE) foi calculada como 0,6073. Essas métricas demonstram que o modelo de regressão treinado apresentou um excelente desempenho nas previsões de carga, com um baixo erro médio e uma alta capacidade de explicar a variabilidade dos dados. Isso sugere que o ANN-HS possui uma boa

precisão na previsão de carga e pode ser uma abordagem eficaz para o escalonamento horizontal de aplicações em ambientes K8s.

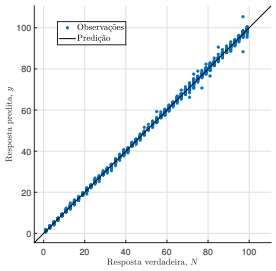

Figura 6. Relação da valor predito de carga, y, e o valor verdadeiro de carga observado no dataset, N.

B. Testes do ANN-HS

As Figuras 7, 8 e 9 mostram as curvas de consumo de CPU em mile core, $m_k^{\text{cpu}}(t)$, a taxa de pacotes $x_k^{\text{rp}}(t)$ por replica em rpm por e número de réplicas $m_k^{\text{p}}(t)$ no tempo para os experimentos $k = 46$ ($L_{46} = 50$).

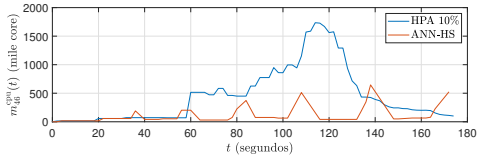

Figura 7. Curvas de consumo de CPU em mile core para $k = 46$ ($L_{46} = 50$).

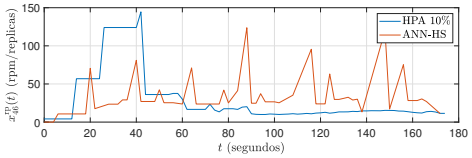

Figura 8. Curvas de taxa de pacotes recebidos em rpm/réplica para $k = 46$ ($L_{46} = 50$).

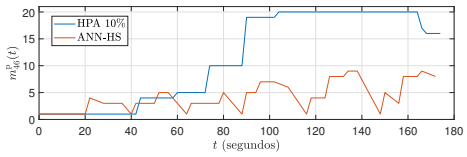

Figura 9. Curvas do número de réplicas para $k = 46$ ($L_{46} = 50$).

Já as Figuras 10, 11 e 12 ilustram a distribuição dos valores associados a média das métricas de CPU, pacotes recebidos em rpm/réplica e números de réplicas em torno dos 180 segundos associados a cada k-ésimo experimento de teste para todos os valores de L_k. Para cada k-ésimo experimento foram gerados as valores $\bar{m}_k^{\text{cpu}}$, $\bar{x}_k^{\text{rp}}$ e $\bar{m}_k^{\text{p}}$ que podem ser caracterizados como

$$\bar{m}_k^{\text{cpu}} = \frac{1}{180}\sum_{t=1}^{180} m_k^{\text{cpu}}(t), \tag{15}$$

$$\bar{x}_k^{\text{rp}} = \frac{1}{180}\sum_{t=1}^{180} x_k^{\text{rp}}(t), \tag{16}$$

e

$$\bar{m}_k^{\text{p}} = \frac{1}{180}\sum_{t=1}^{180} m_k^{\text{p}}(t). \tag{17}$$

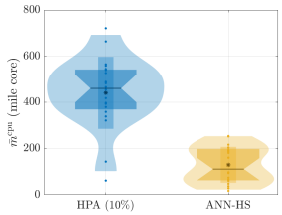

Figura 10. Distribuição dos valores associados a média de consumo de CPU durante os 180 segundos para todos os valores de L_k testados.

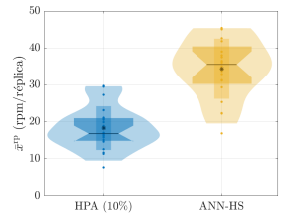

Figura 11. Distribuição dos valores associados a média de consumo de pacotes recebidos em rpm/réplicas durante os 180 segundos para todos os valores de L_k testados.

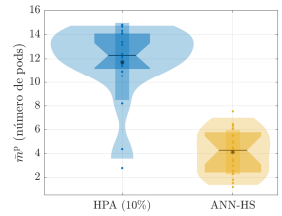

Figura 12. Distribuição dos valores associados a média de número de réplicas durante os 180 segundos para todos os valores de L_k testados.

Finalmente a Figura 13 apresenta a distribuição da taxa de violação para um SLA (*Service Level Agreement*) de latência, δ para todos os experimentos de teste. A taxa de violação para cada k-ésimo experimento de teste, $\mathbf{t}_k$, pode ser expresso como

$$\delta_k = \frac{1}{M}\sum_{j=1}^{M_k} \beta_{k,j} \tag{18}$$

onde M_k é o numero de requisições realizadas pelo jMeter no k-ésimo teste e $\beta_{k,j}$ pode ser expresso como

$$\beta_{k,j} = \begin{cases} 0 & \text{se } r_{k,j} < \text{SLA}_{\max} \\ 1 & \text{se } r_{k,j} \geq \text{SLA}_{\max} \end{cases} \tag{19}$$

onde $r_{k,j}$ é a latência da j-ésima requisição do k-ésimo teste e $\text{SLA}_{\max}$ é a taxa de violação máxima permitida [5]. A Figura 13 apresenta os resultados para um $\text{SLA}_{\max} = 250\,\text{ms}$.

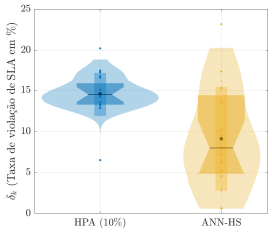

Figura 13. Distribuição da média associada a taxa de violação para um SLA de latência $\mathrm{SLA}_{\max} = 250\,\mathrm{ms}$.

V. Análise dos Resultados

Os resultados obtidos destacam as vantagens do ANN-HS em relação ao HPA, especialmente no que se refere à personalização e à redução do número de réplicas exigidas. Ao conduzir experimentos com diversos cenários de estresse (variando os valores de L_k) utilizando o jMeter, observou-se que o ANN-HS manteve um menor consumo de CPU em comparação ao HPA. Esse desempenho é evidenciado na Figura 7, que representa os resultados de um teste específico de 180 segundos ($L_k = 50$), e é corroborado pela Figura 10, a qual abrange todos os testes realizados.

O ANN-HS, ao empregar um número reduzido de réplicas em comparação ao HPA em todos os experimentos, demonstrou a capacidade de diminuir a taxa de CPU. A otimização mais eficiente da carga de trabalho entre essas réplicas resulta em menor consumo de CPU por unidade, indicando uma utilização mais proveitosa dos recursos e a redução da sobrecarga do sistema. Essa característica é especialmente vantajosa em cenários de estresse, permitindo um equilíbrio aprimorado entre a demanda de recursos e a capacidade do sistema. Este efeito é claramente visualizado na Figura 9 para um teste específico de 180 segundos ($N_k = 50$) e é detalhado na Figura 12 para todos os testes realizados.

Como apresentado nas Figuras 8 e 11 o comportamento do ANN-HS resulta em uma taxa de pacotes em rpm por réplica maior em comparação ao HPA. Essa diferença indica que o HPA tende a subutilizar as réplicas, uma vez que aloca um número maior delas em relação ao FHS para atender à mesma demanda de estresse (diferentes valores de L_k).

Observa-se diferenças marcantes entre o HPA e o ANN-HS no que tange à taxa de violação do SLA de latência de $250\,\mathrm{ms}$, conforme ilustrado na Figura 13. Enquanto o HPA registra concentração em torno de 10% a 20% de violação, o ANN-HS exibe uma distribuição mais ampla, variando de 0% a 20% de violação. O média do HPA é em torno de 15% e a do ANN-HS é em menor que 10%. Essa diferença sugere que o ANN-HS lida de maneira mais eficaz com as variações de carga e demanda, adaptando o número de réplicas de modo mais adequado para manter os níveis de latência em conformidade com o limite estabelecido pelo SLA.

Uma das principais vantagens do ANN-HS é a sua capacidade altamente precisa e eficiente de ajustar o número de réplicas. Ao analisar critérios como a carga de trabalho, a taxa de pacotes recebidos e o número de pods ou réplicas, o ANN-HS emprega um modelo de regressão previamente treinado para determinar a quantidade ideal de réplicas necessárias, evitando alocações excessivas de recursos. Isso se traduz em um uso mais otimizado dos recursos disponíveis no ambiente K8s. Essa flexibilidade personalizada e ajustes refinados do ANN-HS têm o potencial de aprimorar a experiência do usuário e assegurar a qualidade do serviço fornecido.

VI. Conclusões

Este trabalho apresentou uma abordagem baseada em redes neurais artificiais para o escalonamento automático de aplicações em ambientes com K8s, chamado de ANN-HS. O uso do ANN-HS demonstrou vantagens significativas em relação ao escalonador horizontal padrão do K8s (HPA), ao permitir uma distribuição mais eficiente de recursos, uma alocação precisa de réplicas, adaptação dinâmica à demanda e controle aprimorado dos níveis de latência. A aplicação de modelos de regressão pré-treinados confere ao ANN-HS a capacidade de personalizar o escalonamento de acordo com as necessidades específicas de cada aplicação, proporcionando melhor utilização dos recursos disponíveis e aprimorando a qualidade do serviço oferecido.

Referências

[1] M.-N. Tran, D.-D. Vu, and Y. Kim, "A survey of autoscaling in kubernetes," in *2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2022, pp. 263–265.

[2] Q. Huo, S. Li, Y. Xie, and Z. Li, "Horizontal pod autoscaling based on kubernetes with fast response and slow shrinkage," in *2022 International Conference on Artificial Intelligence, Information Processing and Cloud Computing (AIIPCC)*. IEEE, 2022, pp. 203–206.

[3] A. Zafeiropoulos, E. Fotopoulou, N. Filinis, and S. Papavassiliou, "Reinforcement learning-assisted autoscaling mechanisms for serverless computing platforms," *Simulation Modelling Practice and Theory*, vol. 116, p. 102461, 2022.

[4] H. Sami, H. Otrok, J. Bentahar, and A. Mourad, "Ai-based resource provisioning of ioe services in 6g: A deep reinforcement learning approach," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3527–3540, 2021.

[5] Z. Xiao and S. Hu, "Dscaler: A horizontal autoscaler of microservice based on deep reinforcement learning," in *2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2022, pp. 1–6.

[6] H. T. Nguyen, T. Van Do, and C. Rotter, "Scaling upf instances in 5g/6g core with deep reinforcement learning," *IEEE Access*, vol. 9, pp. 165 892–165 906, 2021.

[7] A. A. Khaleq and I. Ra, "Intelligent autoscaling of microservices in the cloud for real-time applications," *IEEE Access*, vol. 9, pp. 35 464–35 476, 2021.

[8] V. Ojha, A. Abraham, and V. Snášel, "Heuristic design of fuzzy inference systems: A review of three decades of research," *Engineering Applications of Artificial Intelligence*, vol. 85, pp. 845–864, 2019.

[9] "Microk8s: Lightweight kubernetes," https://microk8s.io/, acesso em: 18-07-2023.

[10] The Apache Software Foundation, "Apache jmeter," https://jmeter.apache.org/, 2023, acesso em: 18-07-2023.

[11] Dropwizard Development Team, "Dropwizard," https://www.dropwizard.io/, 2023, acesso em: 18-07-2023.

[12] Red Hat, "Fabric8 kubernetes-client," https://github.com/fabric8io/kubernetes-client, 2023, acesso em: 18-07-2023.

[13] The Prometheus Authors, "Prometheus," https://prometheus.io/, 2023, acesso em: 18-07-2023.