Lógica Fuzzy Aplicada a Escalonamento Horizontal

Sérgio N. Silva*, Lucileide M. D. da 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 trabalho apresenta uma abordagem baseada em lógica Fuzzy para o escalonamento de réplicas em um ambiente Kubernetes. O sistema proposto, denominado FHS (Fuzzy-based Horizontal Scaling), foi comparado ao mecanismo padrão de escalonamento do Kubernetes, o HPA (Horizontal Pod Autoscaler). A comparação levou em consideração o consumo de recursos, o número de réplicas utilizadas e aderência aos Acordos de Nível de Serviço (SLAs - Service-Level Agreements) de latência. Os experimentos foram realizados em um ambiente com uma máquina virtual (VM) executando o cluster Kubernetes, outra VM com o JMeter para gerar demandas de tráfego e o controlador FHS no host. Os resultados demonstraram que o FHS obteve uma redução no consumo de CPU, utilizou menos réplicas para uma mesma condição de estresse e apresentou taxas de violação dos SLAs de latência mais distribuídas em comparação ao HPA, que teve valores mais concentrados. Esses resultados indicam que o FHS oferece uma solução mais eficiente e customizável para o escalonamento de réplicas em ambientes Kubernetes.**

***Index Terms*—Escalonamento horizontal, Kubernetes, Lógica Fuzzy, HPA**

I. Introdução

Com o aumento da demanda por serviços e aplicativos distribuídos, a eficiência no gerenciamento de recursos em ambientes de nuvem se tornou uma questão crítica. Os clusters Kubernetes (K8s) surgiram como uma solução popular para a implantação e gerenciamento escalável de aplicações em nuvem [1]–[3]. No entanto, o escalonamento horizontal adequado dos recursos é essencial para otimizar o desempenho e garantir a capacidade de resposta. Uma abordagem comum para o escalonamento horizontal é baseada na carga de CPU, que é um indicador importante, mas não é o único fator relevante para determinar o número de réplicas de um serviço. A taxa de pacotes recebidos e o número atual de pods, por exemplo, também são indicadores que podem afetar significativamente o desempenho do sistema. Portanto, é necessário um mecanismo de controle mais sofisticado para tomar decisões de escalonamento [1]–[3].

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 (*User Plane Function*) 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.

Neste artigo, é proposta a utilização de lógica fuzzy como uma abordagem alternativa para o escalonamento horizontal em clusters K8s. A lógica fuzzy permite modelar a incerteza e a imprecisão presentes nos sistemas reais, tornando-a adequada para lidar com informações vagas e tomar decisões em condições incertas [8]. Assim, este trabalho propõem um sistema chamado de *Fuzzy-based Horizontal Scaling* (FHS) que utiliza um sistema fuzzy do tipo Mandani com três entradas: carga de CPU, taxa de pacotes recebidos e número atual de pods e uma saída que controla o incremento ou decremento do número de réplicas (pods) de uma dada aplicação no cluster K8s.

Comparando os resultados obtidos do FHS com o Horizontal Pod Autoscaler (HPA) do K8s, o FHS é capaz de fornecer um melhor desempenho em termos de tempo de resposta das requisições e conusmo de recursos do cluster, considerando múltiplos fatores na tomada de decisão de escalonamento.

II. Descrição da Proposta

A Figura 1 descreve a arquitetura do FHS no qual um sistema fuzzy, também conhecido na literatura como *Fuzzy Control System* (FCS), é utilizado para o controle de escalonamento horizontal de uma dada aplicação em um cluster Kubernetes (K8s). Em cada n-ésimo instante de tempo, o FCS tem como entrada as variáveis $x^{\mathrm{cpu}}(n)$, $x^{\mathrm{rp}}(n)$ e $m^{\mathrm{p}}(n)$ que representam a carga de CPU em milicore (onde cada milicore é uma unidade que divide um núcleo de CPU em 1000 partes iguais), a taxa de pacotes que a aplicação está recebendo por minuto (*receive packets per minutes* - rpm), ambas normalizadas pelo número de réplicas da aplicação e o número de réplicas de uma dada aplicação, respectivamente. As variáveis $x^{\mathrm{cpu}}(n)$ e $x^{\mathrm{rp}}(n)$ são geradas por um módulo chamado aqui de *Normalization and Saturation Module* (NSM) 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 relativas a carga de CPU de todas as réplicas da aplicação, $m^{\text{cpu}}(n)$, a taxa de pacotes recebidos por todas as réplicas da aplicação em rpm, $m^{\text{rp}}(n)$, e o número de réplicas da aplicação, $m^{\text{p}}(n)$. A cada n-ésimo instante, o FCS tem como saída a variável $d(n)$ que representa um possível incremento para o número de réplicas da aplicação. O valor do incremento $d(n)$ é somado ao número de réplicas atual, $m^{\text{p}}(n)$, para gerar o próximo número de réplicas a ser atingido pelo K8s. O valor associado ao próximo número de réplicas é previamente arredondado para um inteiro e depois saturado por um módulo chamado de *round and saturation mudule* (RSM) gerando o valor, $m^{\text{p}}(n+1)$, que é enviando ao cluster K8s. O ajuste do número de réplicas é feito diretamente pela API do K8s.

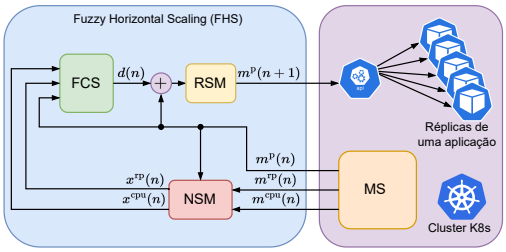

Figura 1. Arquitetura da proposta de controle fuzzy para escalonamento horizontal de aplicações em cluster Kubernetes.

A variável $x^{\text{cpu}}(n)$ pode ser expressa como

$$x^{\text{cpu}}(n) = \begin{cases} \frac{m^{\text{cpu}}(n)}{m^{\text{p}}(n) \times m^{\text{cpu}}_{\max}} & \text{se } \frac{m^{\text{cpu}}(n)}{m^{\text{p}}(n)} < m^{\text{cpu}}_{\max}, \\ 1 & \text{se } \frac{m^{\text{cpu}}(n)}{m^{\text{p}}(n)} \geq m^{\text{cpu}}_{\max}. \end{cases} \quad (1)$$

onde m^{cpu} é a carga por réplica, m^{p} é o número de réplicas associadas aplicação e $m^{\text{cpu}}_{\max}$ é a carga máxima esperada por réplica da aplicação. Já a variável $x^{\text{rp}}(n)$ pode ser expressa como

$$x^{\text{rp}}(n) = \begin{cases} \frac{m^{\text{rp}}(n)}{m^{\text{p}}(n) \times m^{\text{rp}}_{\max}} & \text{se } \frac{m^{\text{rp}}(n)}{m^{\text{p}}(n)} < m^{\text{rp}}_{\max}, \\ 1 & \text{se } \frac{m^{\text{rp}}(n)}{m^{\text{p}}(n)} \geq m^{\text{rp}}_{\max}. \end{cases} \quad (2)$$

onde m^{rp} representa o número de pacotes recebidos por réplica da aplicação por minuto (rpm) e $m^{\text{rp}}_{\max}$ é o número máximo de pacotes recebidos por réplica da aplicação, também por minuto (rpm). Finalmente, a variável

$$m^{\text{p}}(n+1) = \begin{cases} \lfloor m^{\text{p}}(n) + d(n) \rceil & \text{se } m^{\text{p}}(n) < m^{\text{p}}_{\max}, \\ m^{\text{p}}_{\max} & \text{se } m^{\text{p}}(n) \geq m^{\text{rp}}_{\max}. \end{cases} \quad (3)$$

onde $m^{\text{p}}_{\max}$ é o número máximo de réplicas associado a aplicação. A variável $m^{\text{p}}(n+1)$ possui um valor inteiro entre 1 e $m^{\text{p}}_{\max}$. Tomando como base a Figura 1, as Equações 1 e 2 são implementadas pelo NSM e a Equação 3 é implementada pelo SM. Como observado pelas Equações 1 e 2 os valores de $x^{\text{cpu}}(n)$ e $x^{\text{rp}}(n)$ são normalizados entre 0 e 1. Já os valores $d(n)$ são limitados entre $-K$ e $+K$ onde K é o número máximo de incrementos que podem ser realizados a cada n-ésimo instante.

O sistema fuzzy de inferência adotado neste trabalho é do tipo Mandani, que é amplamente utilizado em problemas de controle fuzzy [8], [9]. Para a representação das entradas do sistema fuzzy, são definidas funções de pertinência que capturam a relação entre os valores de entrada e sua pertinência em cada conjunto fuzzy [8]. As entradas $x^{\text{cpu}}(n)$ e $x^{\text{rp}}(n)$ possuem ambas cinco funções de pertinência na forma triangular, distribuídas uniformemente entre 0 e 1. Já a entrada $m^{\text{p}}(n)$ possui três funções de pertinência, com trapezoidais nas extremidades e uma função triangular no centro, distribuídas uniformemente entre 1 e $m^{\text{p}}_{\max}$. As representações gráficas dessas funções de pertinência podem ser observadas nas Figuras 2, 3 e 4. Para as funções de pertinência associadas a variável $m^{\text{p}}(n)$, a Figura 4 mostra um exemplo para $m^{\text{p}}_{\max} = 20$.

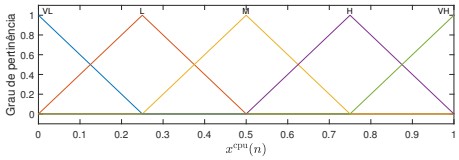

Figura 2. Funções de pertinência associada a carga de cpu, $x^{\text{cpu}}(n)$.

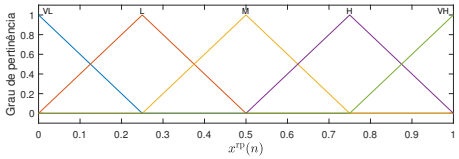

Figura 3. Funções de pertinência associada a taxa de pacotes recebidos, $x^{\text{rp}}(n)$.

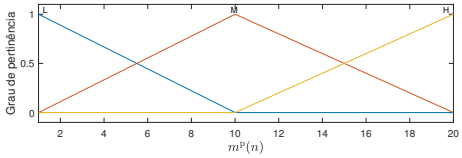

Figura 4. Funções de pertinência associada ao número de réplicas, $m^{\text{p}}(n)$. Neste exemplo $m^{\text{p}}_{\max} = 20$.

A saída $d(n)$ é representada por sete funções de pertinência, com trapezoidais nas pontas e funções triangulares no restante do intervalo. Essas funções de pertinência representam as diferentes possibilidades de incremento ou decremento do número de réplicas da aplicação. A Figura 5 ilustra as funções de pertinência usadas para $d(n)$ para o caso onde $k = 5$.

Os termos linguísticos associados as funções de pertinência das variáveis $x^{\text{cpu}}(n)$ e $x^{\text{rp}}(n)$ foram os mesmos como ilustrados nas Figuras 2 e 3. São eles: Very Low (VL), Low (L), Mean (M), High (H) e Very High (VH). Já a variável $m^{\text{p}}(n)$ fez uso dos termos Low (L), Mean (M) e High (H) para suas funções (ver Figura 4). No caso da variável de saída, $d(n)$ os termos linguísticos foram High Decrement (HD),

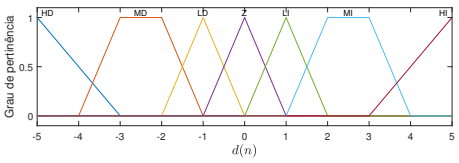

Figura 5. Funções de pertinência associada a variável de saída $d(n)$ que representa o incremento para o número de réplicas do aplicação ($k = 5$).

Mean Decrement (MD), Low Decrement (LD), Zero (Z), Low Incrment (LI), Mean Incriment (MI) e High Increment (HI). A máquina de inferência fuzzy do FHS faz uso de 75 regras de produção apresentadas em detalhes na Tabela I.

A Figura 6 ilustra através de um diagrama do tipo *mosaic* o mapeamento entre entrada e saída do FCS para vários valores distribuídos uniformemente entre 0 e 1 para as variáveis $x^{\text{cpu}}(n)$ e $x^{\text{rp}}(n)$ e entre 1 e $m^{\text{p}}_{\text{max}} = 20$ para a variável $m^{\text{p}}(n)$. Nesta figura, o valor da variável de saída, $d(n)$, foi aredondado ($\lfloor d(n) \rceil$).

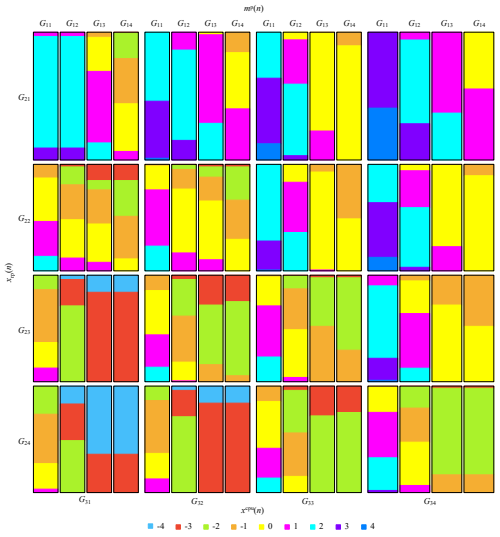

Figura 6. Diagrama do tipo *mosaic* ilustrando o mapeamento entre entrada e saída do FCS.

Como pode ser observado na Figura 6, cada variável de entrada foi agrupada em quatro grupos de valores representados como $G_{i,j}$, $j = 1, \ldots, 4$. A Tabela II detalha o valor de cada grupo onde $x^{\text{cpu}}(n) \in G_{1,j}$, $x^{\text{rp}}(n) \in G_{2,j}$ e $m^{\text{p}}(n) \in G_{3,j}$.

III. Metodologia

O experimento consistiu na utilização de duas máquinas virtuais (*Virtual Machines* - VMs) distintas para diferentes propósitos. A primeira VM foi configurada como um cluster K8s do tipo Micro K8s [10], onde uma aplicação web alvo do experimento foi implantada. Essa máquina atuou como o ambiente alvo para o teste de escalonamento das réplicas. A segunda VM foi dedicada à execução do JMeter [11]. Essa máquina foi responsável por gerar as demandas de tráfego HTTP para a aplicação web hospedada no cluster K8s. A aplicação web implantada no K8s foi desenvolvida em Java,

Tabela I
Descrição das regras utilizadas pelo FHS.

Número	Regra
1	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is L then $d(n)$ is Z
2	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is L then $d(n)$ is Z
3	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is L then $d(n)$ is LI
4	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is L then $d(n)$ is LI
5	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is L then $d(n)$ is MI
6	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is L then $d(n)$ is Z
7	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is L then $d(n)$ is LI
8	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is L then $d(n)$ is LI
9	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is L then $d(n)$ is MI
10	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is L then $d(n)$ is MI
11	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is L then $d(n)$ is LI
12	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is L then $d(n)$ is LI
13	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is L then $d(n)$ is MI
14	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is L then $d(n)$ is MI
15	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is L then $d(n)$ is HI
16	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is L then $d(n)$ is LI
17	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is L then $d(n)$ is MI
18	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is L then $d(n)$ is MI
19	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is L then $d(n)$ is HI
20	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is L then $d(n)$ is HI
21	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is L then $d(n)$ is MI
22	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is L then $d(n)$ is MI
23	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is L then $d(n)$ is HI
24	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is L then $d(n)$ is HI
25	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is L then $d(n)$ is HI
26	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is M then $d(n)$ is Z
27	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is M then $d(n)$ is Z
28	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is M then $d(n)$ is LI
29	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is M then $d(n)$ is LI
30	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is M then $d(n)$ is MI
31	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is M then $d(n)$ is Z
32	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is M then $d(n)$ is LI
33	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is M then $d(n)$ is LI
34	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is M then $d(n)$ is MI
35	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is M then $d(n)$ is MI
36	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is M then $d(n)$ is LI
37	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is M then $d(n)$ is LI
38	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is M then $d(n)$ is MI
39	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is M then $d(n)$ is HD
40	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is M then $d(n)$ is HD
41	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is M then $d(n)$ is MD
42	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is M then $d(n)$ is MD
43	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is M then $d(n)$ is LD
44	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is M then $d(n)$ is HD
45	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is M then $d(n)$ is MD
46	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is M then $d(n)$ is MD
47	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is M then $d(n)$ is LD
48	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is M then $d(n)$ is LD
49	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is M then $d(n)$ is MD
50	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is M then $d(n)$ is MD
51	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is H then $d(n)$ is HD
52	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is H then $d(n)$ is HD
53	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is H then $d(n)$ is MD
54	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is H then $d(n)$ is MD
55	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is VL and $m^{\text{p}}(n)$ is H then $d(n)$ is LD
56	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is H then $d(n)$ is HD
57	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is H then $d(n)$ is MD
58	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is H then $d(n)$ is MD
59	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is H then $d(n)$ is LD
60	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is L and $m^{\text{p}}(n)$ is H then $d(n)$ is LD
61	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is H then $d(n)$ is MD
62	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is H then $d(n)$ is MD
63	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is H then $d(n)$ is LD
64	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is H then $d(n)$ is LD
65	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is M and $m^{\text{p}}(n)$ is H then $d(n)$ is Z
66	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is H then $d(n)$ is MD
67	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is H then $d(n)$ is LD
68	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is H then $d(n)$ is LD
69	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is H then $d(n)$ is Z
70	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is H and $m^{\text{p}}(n)$ is H then $d(n)$ is Z
71	if $x^{\text{cpu}}(n)$ is VL and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is H then $d(n)$ is LD
72	if $x^{\text{cpu}}(n)$ is L and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is H then $d(n)$ is LD
73	if $x^{\text{cpu}}(n)$ is M and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is H then $d(n)$ is Z
74	if $x^{\text{cpu}}(n)$ is H and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is H then $d(n)$ is Z
75	if $x^{\text{cpu}}(n)$ is VH and $x^{\text{rp}}(n)$ is VH and $m^{\text{p}}(n)$ is H then $d(n)$ is Z

Tabela II
Descrição dos valores associados aos grupos apresentados na Figura 6.

Variável	i	$G_{i,1}$	$G_{i,2}$	$G_{i,3}$	$G_{i,4}$
$m^{\text{p}}(n)$	1	< 5.5	$5.5 - 10.5$	$10.5 - 15.5$	≥ 15.5
$x^{\text{rp}}(n)$	2	< 0.225	$0.225 - 0.475$	$0.475 - 0.725$	≥ 0.725
$x^{\text{cpu}}(n)$	3	< 0.225	$0.225 - 0.475$	$0.475 - 0.725$	≥ 0.725

utilizando o Dropwizard [12], realiza cálculos de sequência Fibonacci de ordem 15 e retorna esta sequência como resposta às solicitações realizadas pelo JMeter. A aplicação implantada no K8s foi limitada em 150 milicores e 256 MB. O FHS foi configurado para trabalhar com $m^{\text{cpu}}_{\text{max}} = 100$ milicores por réplica, $m^{\text{rp}}_{\text{max}} = 50$ rpm por réplica e $m^{\text{p}}_{\text{max}} = 20$ réplicas.

Além dessas máquinas virtuais, o controlador Fuzzy pro-

posto para o escalonamento das réplicas, o FHS, foi implementado e executado no próprio host do experimento. Utilizando o Matlab (596681) como ambiente de desenvolvimento, o FHS utilizou a biblioteca Java Fabric8 Kubernetes-Client [13] para interagir com a API do K8s e realizar o escalonamento dinâmico das réplicas da aplicação web. Dessa forma, o FHS atuou como um componente externo ao cluster, tomando decisões de escalonamento com base nas métricas coletadas e enviando as requisições de escalonamento para o cluster Kubernetes Micro K8s. As métricas foram coletadas utilizando o Prometheus [14] que funciona como o MS (ver Figura 1). A VM com o cluster K8s possui 12GB de memória RAM, 4 CPUs e Ubuntu 20.04 server, já a outra VM possui 12 GB de memória RAM, 3 CPUs e Ubuntu 22.04 server. O host é uma máquina com 128 GB de memória RAM 16 CPUs com Windows 10.

Os resultados foram gerados a partir do perfil de carga ilustrado na Figura 7. 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 [11].

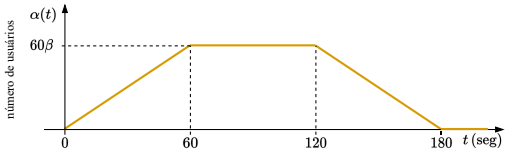

Figura 7. Perfil de criação de usuários virtuais pelo JMeter usando *Open Model Thread Group* para geração de acessos na aplicação web.

Com uma duração de 180 segundos, o perfil de carga para criação de usuários, é caracterizado por uma subida em rampa nos primeiros 60 segundos até atingir uma quantidade $\beta \times 60$ usuários. Entre 60 e 120 segundos as requisições ficam constantes em $\beta \times 60$ e após 120 segundos a quantidade de usuários diminui em rampa (seguindo a mesma taxa de subida) até os 180 segundos. A quantidade de usuários virtuais em um dado tempo, $\alpha(t)$, pode ser expressa como

$$\alpha(t) = \begin{cases} \beta t & \text{se } 0 < t \leq 60, \\ 60\beta & \text{se } 60 < t \leq 120, \\ -\beta t + 180\beta & \text{se } 120 < t \leq 180, \\ 0 & \text{se } t > 180, \end{cases} \tag{4}$$

onde β é numero máximo de usuários virtuais que o perfil de carga pode atingir. Foram realizados testes com 30 valores diferentes para β, no qual cada k-ésimo valor pode ser expresso como $\beta_k \in 1, 3, 5, \ldots, 57, 59$, para $k = 1, 2, \ldots, 30$. Para cada k-ésimo valor de β_k, os testes foram repetidos 3 vezes totalizando 90 experimentos.

Para fins de comparação, também foram conduzidos experimentos adicionais utilizando o Horizontal Pod Autoscaler (HPA) nativo do K8s [1], [2]. Esses testes seguiram a mesma abordagem com o FHS, variando a carga pelo número de usuários através do perfil ilustrado na Figura 7 e pela Equação 4. O HPA foi implantado com os parâmetros: *averageUtilization:* 2, *minReplicas:* 1, *maxReplicas:* 20, *scaleDown: stabilizationWindowSeconds:* 0 e *scaleUp: stabilizationWindowSeconds:* 0.

Ao final dos testes com o FHS e HPA foram criados 90 conjuntos de dados com carga de cpu usada pela aplicação em milicores, $m_{k,i}^{\text{cpu}}(t)$, taxa de pacotes recebidos pela aplicação em rpm, $m_{k,i}^{\text{rp}}(t)$, número de réplicas, $m_{k,i}^{\text{p}}(t)$ e a latência das $M_{k,i}$ requisições realizadas pelo JMeter, $r_{k,i}$, associados a i-ésima repetição do k-ésimo teste de 180 segundos. É importante destacar que as variáveis $m_{k,i}^{\text{cpu}}(t)$ e $m_{k,i}^{\text{rp}}(t)$ levam em consideração todas as réplicas em execução no dado instante de tempo, t. As variáveis $x_{k,i}^{\text{cpu}}(t)$, $x_{k,i}^{\text{rp}}(t)$, e $m_{k,i}^{\text{p}}(t)$ foram obtidas pelo Prometheus e a variável $r_{k,i}(t)$ foi obtida pelo JMeter.

IV. Resultados

As Figuras 8, 9, 10, 11, 12 e 13 mostram as curvas médias de consumo de CPU em milicore, $m_k^{\text{cpu}}(t)$, a média da taxa de pacotes $\bar{v}_{k,i}^{\text{rp}}(t)$ por réplica em rpm e a média do número de réplicas $\bar{m}_{k,i}^{\text{p}}(t)$ no tempo para os experimentos $k = 5$ ($\beta_5 = 9$) e $k = 25$ ($\beta_{25} = 49$). Neste caso a média foi realizada entre as três repetições para cada k-ésimo teste no qual o cálculo de $m_k^{\text{cpu}}(t)$, $\bar{v}_k^{\text{rp}}(t)$ e $m_{k,i}^{\text{p}}(t)$, podem ser expressos como

$$\bar{m}_k^{\text{cpu}}(t) = \frac{1}{3}\sum_{i=1}^{3} m_{k,i}^{\text{cpu}}(t), \tag{5}$$

$$\bar{m}_k^{\text{p}}(t) = \frac{1}{3}\sum_{i=1}^{3} m_{k,i}^{\text{p}}(t) \tag{6}$$

e

$$\bar{v}_k^{\text{rp}}(t) = \frac{1}{3}\sum_{i=1}^{3} v_{k,i}^{\text{rp}}(t) \tag{7}$$

onde

$$v_{k,i}^{\text{rp}}(t) = \frac{m_{k,i}^{\text{rp}}(t)}{m_{k,i}^{\text{p}}(t)} \tag{8}$$

no qual $v_{k,i}^{\text{rp}}(t)$ é taxa de pacotes recebidos em rpm por réplica.

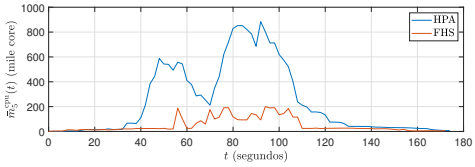

Figura 8. Curvas média de consumo de CPU em milicore para $k = 5$ ($\beta_5 = 9$).

As Figuras 14, 15 e 16 mostram a distribuição dos valores associados a média das métricas de CPU, pacotes recebidos e número de réplicas em torno dos 180 segundos associados a cada k-ésimo teste para todos os valores de β. Para cada k-ésimo teste foram gerados os valores $\bar{y}_k^{\text{cpu}}$, $\bar{y}_k^{\text{rp}}$ e $\bar{y}_k^{\text{p}}$ que podem ser expressos como

$$\bar{y}_k^{\text{cpu}} = \frac{1}{180}\sum_{t=1}^{180} \bar{m}_k^{\text{cpu}}(t), \tag{9}$$

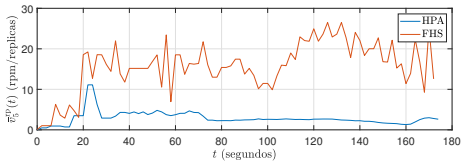

Figura 9. Curvas média de taxa de pacotes recebidos em rpm para $k = 5$ ($\beta_5 = 9$).

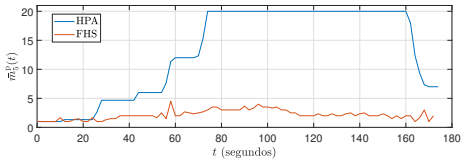

Figura 10. Curvas média do número de réplicas para $k = 5$ ($\beta_5 = 9$).

$$\bar{y}_k^{\mathrm{rp}} = \frac{1}{180} \sum_{t=1}^{180} \bar{v}_k^{\mathrm{rp}}(t), \tag{10}$$

e

$$\bar{y}_k^{\mathrm{p}} = \frac{1}{180} \sum_{t=1}^{180} \bar{m}_k^{\mathrm{p}}(t). \tag{11}$$

Finalmente a Figura 17 apresenta os dados de distribuição da média associada a taxa de violação para um SLA (*Service Level Agreement*) de latência, $\bar{\gamma}_k$, previamente caracterizado como $\mathrm{SLA}_{\max}$ [5]. A taxa de violação do SLA de latência, $\gamma_{k,i}$, para cada i-ésima repetição associado ao k-ésimo teste, foi calculada com base nas medidas de latência das requisições HTTP realizadas pelo JMeter, ou seja,

$$\gamma_{k,i} = \frac{1}{M} \sum_{j=1}^{M_{k,i}} \delta_{k,i,j} \tag{12}$$

onde $M_{k,i}$ é o numero de requisições realizadas pelo JMeter na i-ésima repetição do k-ésimo teste e $\delta_{k,i,j}$ pode ser expresso como

$$\delta_{k,i,j} = \begin{cases} 0 & \text{se } r_{k,i,j} < \mathrm{SLA}_{\max} \\ 1 & \text{se } r_{k,i,j} \geq \mathrm{SLA}_{\max} \end{cases} \tag{13}$$

onde $r_{k,i,j}$ é a latência da j-ésima requisição da i-ésima repetição associada ao k-ésimo teste. Com base nos valores de $\gamma_{k,i}$ foi calculada a média associada a taxa de violação SLA para cada k-ésimo teste, ou seja,

$$\bar{\gamma}_k = \frac{1}{3} \sum_{i=1}^{3} \gamma_{k,i}. \tag{14}$$

A Figura 17 apresenta os resultados para um $\mathrm{SLA}_{\max} = 250\,\mathrm{ms}$.

V. Análise dos Resultados

Os resultados obtidos demonstraram que o FHS apresentou vantagens em relação HPA do Kubernetes, especialmente no que diz respeito à customização e redução do número de réplicas necessárias. Durante os 90 experimentos realizados com diferentes cenários de estresse (diferentes valores de β)

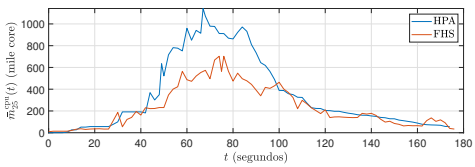

Figura 11. Curvas média de consumo de CPU em milicore para $k = 25$ ($\beta_{25} = 49$).

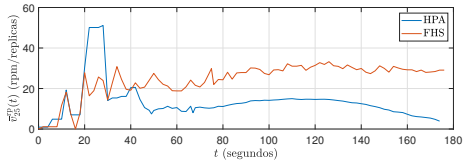

Figura 12. Curvas média de taxa de pacotes recebidos em rpm para $k = 25$ ($\beta_{25} = 49$).

gerados pelo JMeter, observou-se que o FHS conseguiu manter um consumo de CPU mais baixo em comparação ao HPA. As Figuras 8 e 11 ilustram estes resultados durante os 180 segundos de dois testes específicos ($\beta = 9$ e $\beta = 49$) e a Figura 14 reforça este resultado para todos os testes realizados.

O fato de o FHS utilizar um número menor de réplicas em comparação ao HPA em todos os experimentos contribui para a redução na taxa de CPU. Com menos réplicas em operação, a carga de trabalho é distribuída de forma mais eficiente entre elas, resultando em um menor consumo de CPU por réplica. Isso indica uma melhor utilização dos recursos disponíveis e uma redução na sobrecarga do sistema. Essa característica do FHS pode ser vantajosa em cenários de estresse, pois permite uma otimização mais precisa e um melhor equilíbrio entre a demanda de recursos e a capacidade do sistema. As Figuras 10 e 13 ilustram estes resultados durante os 180 segundos de dois testes específicos ($\beta = 9$ e $\beta = 49$). Já a Figura 16 detalha este resultado para todos os testes realizados.

Como apresentado nas Figuras 9, 12 e 15 o comportamento do FHS 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 β).

É interessante observar que, em relação à taxa de violação do SLA de latência de 250 ms, os resultados (ver Figura 17) mostraram diferenças significativas entre o HPA e o FHS. Enquanto o HPA apresentou valores concentrados em torno de 5% e 7% de violação, o FHS exibiu uma distribuição mais ampla, variando de 0% a 8% de violação. Isso indica que o FHS foi capaz de lidar de forma mais eficiente com as variações de carga e demanda, adaptando o número de réplicas de maneira mais adequada para manter os níveis de latência dentro do limite estabelecido pelo SLA.

A capacidade de ajustar o número de réplicas de forma mais precisa e eficiente é um dos principais benefícios do FHS. Ao analisar a carga de trabalho, a taxa de pacotes

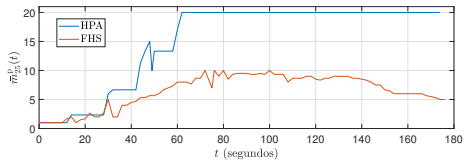

Figura 13. Curvas média do número de réplicas para $k = 25$ ($\beta_{25} = 49$).

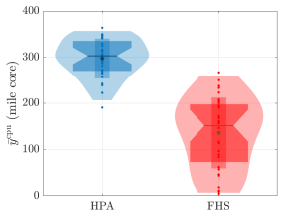

Figura 14. Distribuição dos valores associados a média de consumo de CPU durante os 180 segundos para todos os valores de β testados.

recebidos e o número de pods ou réplicas, o FHS aplicou regras fuzzy personalizadas para determinar a quantidade ideal de réplicas necessárias para atender à demanda, evitando assim a alocação excessiva de recursos. Isso resultou em um melhor aproveitamento dos recursos disponíveis no K8s. Essa capacidade de customização e ajuste mais fino do FHS pode contribuir para uma melhor experiência do usuário e garantir a qualidade do serviço oferecido.

VI. Conclusões

Este trabalho apresentou a abordagem FHS (Fuzzy-based Horizontal Scaling) para o escalonamento de réplicas em clusters K8s. Os resultados obtidos demonstraram que o FHS possui vantagens significativas em relação ao HPA (Horizontal Pod Autoscaler), proporcionando uma redução no consumo de CPU, uma taxa de pacotes recebidos por réplica mais elevada e uma distribuição mais ampla da taxa de violação do SLA de latência. Além disso, o FHS mostrou-se capaz de adaptar o número de réplicas de forma mais customizada e eficiente, ajustando-se melhor às demandas de carga de trabalho. Assim, essa abordagem oferece uma alternativa promissora para otimizar o desempenho e a utilização de recursos em ambientes K8s, contribuindo para a eficiência operacional e a qualidade de serviço das aplicações hospedadas.

Referências

[1] M.-N. Tran, D.-D. Vu, and Y. Kim, "A survey of autoscaling in kubernetes," in *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 *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.

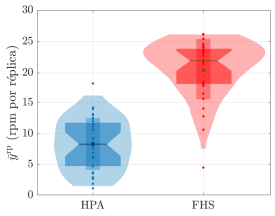

Figura 15. Distribuição dos valores associados a média de consumo de pacotes recebidos por réplicas em rpm durante os 180 segundos para todos os valores de β testados.

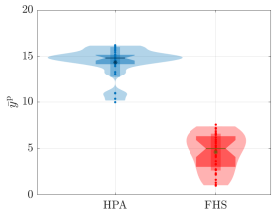

Figura 16. Distribuição dos valores associados a média do número de réplicas durante os 180 segundos para todos os valores de β testados.

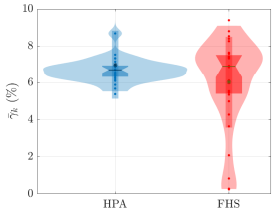

Figura 17. Distribuição da média associada a taxa de violação para um SLA de latência $\mathrm{SLA}_{\max} = 250\,\mathrm{ms}$.

[5] Z. Xiao and S. Hu, "Dscaler: A horizontal autoscaler of microservice based on deep reinforcement learning," in *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] A. M. Radwan and I. M. Ellabib, "Fuzzy inference systems for load balancing of wireless networks," in *2023 IEEE 3rd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*, 2023, pp. 154–158.

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

[11] "Apache jmeter," https://jmeter.apache.org/, 2023, acesso em: 18-07-2023.

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

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

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