# Fuzzy Admission Control Using Non-Intrusive QoS Inference for Resource-Constrained Environments

Abdullah Muslim, Ali Beiti Aydenlou, and Stephan Recker
*Department of Computer Science*
*University of Applied Sciences and Arts*
Dortmund, Germany
{abdullah.muslim,ali.beitiaydenlou,stephan.recker}@fh-dortmund.de

***Abstract*—Edge and fog nodes operate under limited resources and often host multiple workloads simultaneously, making it difficult to maintain application quality of service (QoS) under resource stress. Admission control is therefore essential, especially for black-box workloads where application-level metrics are unavailable. However, many existing approaches rely on such metrics or fixed resource thresholds.**

**This paper presents a fuzzy logic-based admission control framework that uses non-intrusive QoS signals derived from kernel-level delays. Scheduling, network, and I/O delays are collected using eBPF and combined into a contention indicator. This indicator, together with CPU utilization and Pressure Stall Information (PSI), is used by a fuzzy controller to determine whether new workloads can be admitted without degrading QoS.**

**We evaluate the approach on a single-node Kubernetes cluster using microservice benchmarks from the DeathStarBench suite and an AI inference workload. Results show that the fuzzy controller reduces resource contention and improves tail latency by up to 35–40% compared with CPU-based Horizontal Pod Autoscaling (HPA) under mixed and dynamic workloads. These results demonstrate that kernel-level delay signals combined with fuzzy inference provide an effective admission control mechanism for resource-constrained edge environments.**

***Index Terms*—Edge computing, admission control, fuzzy logic, QoS, eBPF, Kubernetes, mahalanobis distance**

## I. Introduction

Edge and fog computing environments are usually deployed on platforms with limited space and fixed hardware capacity [1], [2]. Unlike cloud data centers, which can rely on horizontal scaling as demand grows, edge and fog nodes operate within strict hardware limits. As a result, multiple workloads are often placed on the same server, where a server refers to an edge node with fixed compute resources such as CPU, memory, and local storage, while network conditions are assumed to be sufficient. When load increases, this shared use of resources leads to contention and makes it harder to maintain acceptable application quality of service (QoS).

Given these constraints, efficient workload admission at the server level is necessary to limit resource contention. New workloads should be admitted only when they are unlikely to degrade the performance of applications already running on the node. In practice, admission decisions are commonly made using fixed CPU or memory thresholds, which are conservative and fail to handle short-term load increases [4].

Admission decisions require some understanding of application-level QoS. However, server operators, who manage the server, often have little visibility into hosted applications, which are therefore treated as black boxes [5]. Application internals are often inaccessible because of confidentiality constraints or independent ownership, making intrusive instrumentation difficult. At the same time, standard system metrics such as CPU and memory utilization are weak and often misleading indicators of application-level performance as experienced by users [3].

In this paper, we use a non-intrusive, extended Berkeley Packet Filter (eBPF) based QoS inference signal derived from kernel-level delay behavior [6]. The signal provides a relative indication of performance degradation and requires no application-specific instrumentation. Because the signal is noisy and not an absolute QoS metric, it cannot be used directly for admission decisions. Therefore, admission decisions are made using a fuzzy controller that considers the inferred QoS signal together with server-side resource indicators, such as CPU utilization and Pressure Stall Information (PSI) [7]. Fuzzy logic is a good fit in this case because it handles noisy and uncertain signals.

### A. Contributions

The main contributions of this work are:

- We propose a fuzzy control-based admission control mechanism that uses non-intrusive QoS indicators derived from kernel-level delay behavior for resource-constrained environments.
- We evaluate the proposed approach using real-world microservice application benchmarks from the DeathStarBench suite [10].

## II. Methodology and Paper Structure

Kernel-level delay signals are observed using eBPF and combined into a contention indicator using a Mahalanobis distance-based metric [9]. This indicator, together with server resource metrics, is used as input to a fuzzy admission controller.

Section III reviews related work. Section IV describes the QoS inference method. Section V presents the fuzzy admission control design. Section VI describes the system architecture. Section VII presents the evaluation, and Section VIII concludes the paper.

## III. Related Work

This section reviews prior work on admission control and QoS management in resource-constrained systems, including approaches based on fuzzy logic.

### A. Admission Control and QoS Management in Edge/Fog Settings

Recent work on QoS preservation in edge environments focuses on managing when and how workloads are executed under limited resources. A common approach is to limit local execution when a node is overloaded, either by rejecting tasks, delaying them, or offloading requests to the cloud. For example, Chen et al. and Ouyang et al. propose admission and provisioning methods based on application-level models, such as queueing delay or utility metrics, assuming that workload types and service behavior are known [13], [14]. Both methods are evaluated in simulation using predefined workloads with known characteristics.

Other systems aim to maintain QoS in edge-cloud setups by learning when to move workloads away from overloaded nodes. For example, Han et al. learn to predict QoS drops and send new requests to the cloud to maintain high performance [15]. These systems require centralized control and detailed telemetry. Some approaches also reserve resources for latency-critical tasks to prevent overload during contention [16].

Unlike these methods, we focus on resource-constrained nodes running black-box workloads. We do not rely on application metrics or orchestration; instead, we use kernel signals to detect QoS issues and decide when to admit new workload.

### B. Fuzzy Controllers for Resource and Admission Control

Fuzzy logic has been applied in computing systems to improve control decisions when workloads behave unpredictably. It allows systems to combine different signals and respond more flexibly to changing resource conditions.

In cloud and edge settings, fuzzy logic is used to support admission and planning decisions. For instance, Tomás and Tordsson apply it in cloud data centers to assess overbooking risk and decide whether to accept new workloads based on current resource usage [11]. In edge environments, Sahoo *et al.* apply fuzzy inference to connect CPU, memory, and I/O usage with latency-aware resource planning [12].

Overall, these studies show that fuzzy logic can help manage uncertainty in edge and fog environments. However, they often assume knowledge of workload characteristics or rely on mechanisms such as offloading, scaling, or task migration, which may not be feasible on constrained nodes.

## IV. Non-Intrusive QoS Inference

We consider three kernel-level delays. *CPU scheduling delay* captures how long runnable tasks wait for CPU time under contention. *Network SoftIRQ processing time* reflects overhead introduced by interrupt-driven network processing. *Uninterruptible I/O sleep time* (*D-state*) indicates periods where tasks are blocked while waiting for disk or network I/O. These signals can be observed directly in the kernel using eBPF.

We represent the state of a server node with a feature vector derived from kernel delays. Each feature vector corresponds to a point in a multi-dimensional space. We then use a distance-based metric learning approach to measure deviations in the node's behavior [8]. At each time step $t$, the node state is captured by this feature vector.

$$\mathbf{x}_t = \begin{bmatrix} x_{sched}(t) & x_{softirq}(t) & x_{dstate}(t) \end{bmatrix}^\top \tag{1}$$

To assess how much the current behavior deviates from normal operation, we compute the Mahalanobis distance to a baseline distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$:

$$F_t = \sqrt{(\mathbf{x}_t - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x}_t - \boldsymbol{\mu})}. \tag{2}$$

The resulting scalar $F_t$, referred to as *friction*, serves as a relative indicator of kernel-level contention. Because friction comes from noisy kernel signals, it does not directly represent QoS. We therefore use it as a helpful indicator when making admission control decisions. To account for short-term fluctuations, we also compute an *energy* signal that reflects how rapidly friction changes over time.

$$E_t = \frac{1}{W_t - 1} \sum_{i=t-W_t+2}^{t} \Delta F_i. \tag{3}$$

where $W_t$ is a dynamic sliding window capturing recent friction changes, and $\Delta F_i = F_i - F_{i-1}$ is the change between consecutive samples.

This approach requires a calibration phase during which the mean and covariance matrix are learned to represent the baseline system behavior. During the production phase, the friction value $F_t$ is computed relative to this learned baseline.

## V. Proposed Fuzzy Admission Control Approach

This section presents the design of the proposed fuzzy admission control mechanism.

### A. Fuzzy Input Variables and Membership Functions

The fuzzy admission controller uses four input variables to describe the current state of a node. Each variable is mapped to fuzzy sets using predefined membership functions.

CPU utilization is classified into two fuzzy sets: *Normal* and *High*. This classification indicates whether the node operates within a stable utilization range or approaches resource saturation.

PSI reflects the percentage of time tasks spend waiting due to resource contention. In this paper, we use a 1 s measurement interval. Based on this value, PSI is categorized into three fuzzy sets: *Low*, *Medium*, and *High*.

Friction represents kernel-level contention and blocking caused by competition for shared CPU, network, and I/O resources and reflects deviation from normal operating behavior. Friction is categorized as *UnderUtilized*, *Low*, *Medium*, or *High*. Energy represents short-term instability in the inferred

TABLE I
FUZZY RULE BASE (OUTPUT FOR EACH FRICTION-ENERGY COMBINATION UNDER CPU AND PSI CONDITIONS)

| Friction | CPU = Normal | | | | | | | | | CPU = High | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSI = Low | | | PSI = Medium | | | PSI = High | | | PSI = Low | | | PSI = Medium | | | PSI = High | | |
| | Energy | | | | | | | | | | | | | | | | | |
| | Short | Moderate | Long | Short | Moderate | Long | Short | Moderate | Long | Short | Moderate | Long | Short | Moderate | Long | Short | Moderate | Long |
| **Under** | UnderUtilized | Low | Low | Low | Low | Medium | Low | Medium | Medium | Low | Low | Low | Low | Medium | Medium | High | High | High |
| **Low** | Low | Low | Medium | Low | Medium | Medium | Medium | Medium | High | Low | Medium | Medium | Medium | Medium | High | High | High | High |
| **Medium** | Low | Medium | Medium | Medium | Medium | High | Medium | High | High | Medium | High | High | High | High | High | High | High | High |
| **High** | Medium | High | High | High | High | High | High | High | High | High | High | High | High | High | High | High | High | High |

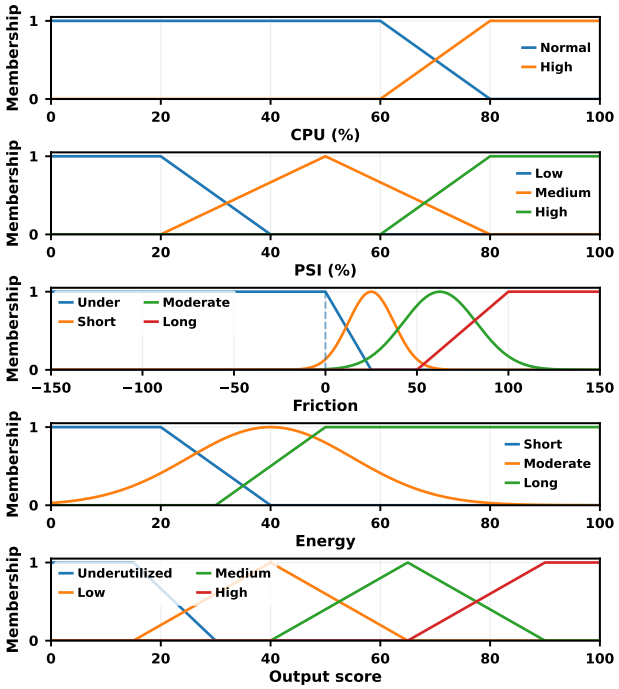



Fig. 1. Membership functions used by the fuzzy admission controller: CPU utilization, PSI, friction, energy, and admission output score.

QoS behavior and captures how rapidly friction changes over time. Energy is classified as *Short*, *Moderate*, or *Long*.

All inputs to the fuzzy controller are defined over bounded ranges to ensure stable operation. CPU utilization and PSI are expressed as percentages in the range 0–100. Friction measures how far the current system state is from a normal baseline and therefore has no physical unit. Energy is computed from changes in friction and is also unitless. Both values are scaled to fixed ranges before being used by the fuzzy controller. The membership functions are defined using a combination of triangular, trapezoidal, and Gaussian shapes, as shown in Fig. 1. Triangular and trapezoidal functions capture clear low and high operating regions, while Gaussian functions are used around moderate conditions to provide smoother transitions and avoid abrupt changes between states. For friction, negative values are also considered. When the system state is below the baseline, the friction value becomes negative to indicate underutilization in comparison to the baseline.

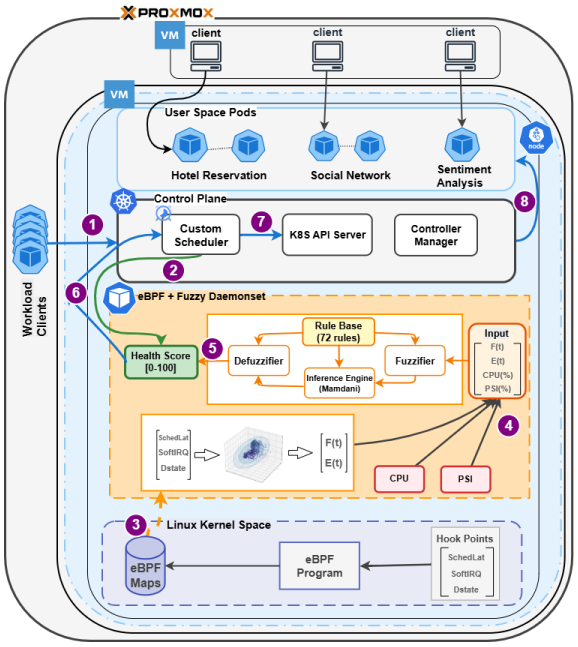



Fig. 2. System architecture of the proposed eBPF-based fuzzy admission control framework.

### B. Fuzzy Rule Base

Admission decisions are derived from a fuzzy rule base that combines information about CPU utilization, PSI, friction, and energy as shown in Table I.

The controller evaluates all combinations of the four input variables, resulting in 72 fuzzy rules. During operation, the rules are processed using a standard Mamdani inference approach. The activation strength of each rule is determined by the minimum of its input memberships, and rules producing the same output are combined using the maximum operator. In this way, only the rules that match the current system state affect the final decision, while the rule base itself remains fixed.

The combined fuzzy output is converted into a single admission score using centroid defuzzification.

## VI. SYSTEM ARCHITECTURE

Fig 2 shows the architecture of the proposed admission control framework. Applications run as user-space pods on a single-node Kubernetes (K8) cluster, while their clients run on a separate virtual machine (VM) outside the cluster.

TABLE II
SYSTEM SPECIFICATIONS

| Component | Specification |
|---|---|
| Environment | KVM-based virtual machine |
| Operating System | Ubuntu 22.04.5 LTS |
| Kernel | Linux 6.8.0-90-generic |
| CPU | AMD EPYC 7643 (virtualized) |
| CPU Cores (vCPU) | 16 |
| CPU Frequency | 2.0 GHz |
| Cache (L1/L2/L3) | 1 MB / 8 MB / 256 MB |
| Memory | 64 GB ECC RAM |
| Storage | 200 GB virtual disk |

Kernel-level behavior is monitored at 1 s intervals using eBPF programs attached to selected hook points. The signals are collected in the Linux kernel and exported to user space with low overhead.

In user space, the computed friction and energy metrics, together with CPU utilization and PSI, are provided as inputs to a fuzzy admission controller. The controller applies a fixed Mamdani rule base and produces a scalar health score that reflects the current load and stability of the application QoS.

The health score is sent to the K8 control plane and used by a custom scheduler to decide whether new workloads can be admitted without degrading QoS. The decision is applied through the K8 API. In our setup, the default admission webhook is disabled so that workload admission is handled by the custom scheduler.

If friction or energy cannot be obtained, the controller rejects the request. Otherwise, the fuzzy inference calculates the score. Low scores indicate stable operation, while high scores indicate an increased risk of QoS degradation.

The defuzzified score is then mapped to an admission decision. If the score is high, the request is rejected right away. If the score is low, a workload is admitted only after the system has been stable for a while. When the score is in the medium range, this indicates uncertain conditions. In this case, the controller keeps its previous decision and applies short counters to avoid frequent switching between allow and deny decisions. See algorithm 1 for further details.

This design allows the admission controller to respond quickly to overload while remaining robust to short-term fluctuations in system metrics.

**Algorithm 1** Score-Based Admission Decision

**State:** lastDecision $\in \{\text{ALLOW}, \text{DENY}\}$; goodCnt; badCnt
**Params:** goodThresh=2; badThresh=3; denyCooldown=3; allow_on_missing=false
**Bands:** BELOW-LOW ($< 45$); MED-LOW ($[45, 55)$); HOLD ($[55, 60)$); MED-HIGH ($[60, 70)$); HIGH ($\geq 70$)
score $\leftarrow$ EVALUATE($F_t, E_t, u_t, p_t$).score
**if** $F_t$ missing **or** $E_t$ missing **or** score $\geq 70$ **then**
  decision $\leftarrow$ DENY
  badCnt $\leftarrow$ denyCooldown
  goodCnt $\leftarrow 0$
**else**
  **if** score $< 55$ **then**
    goodCnt $\leftarrow$ goodCnt $+1$
    badCnt $\leftarrow 0$
    **if** goodCnt $\geq$ goodThresh **then**
      decision $\leftarrow$ ALLOW
    **else**
      decision $\leftarrow$ lastDecision
    **end if**
  **else**
    **if** score $< 60$ **then**
      decision $\leftarrow$ lastDecision
    **else**
      badCnt $\leftarrow$ badCnt $+1$
      goodCnt $\leftarrow 0$
      **if** badCnt $\geq$ badThresh **then**
        decision $\leftarrow$ DENY
        badCnt $\leftarrow$ denyCooldown
      **else**
        decision $\leftarrow$ lastDecision
      **end if**
    **end if**
  **end if**
**end if**
lastDecision $\leftarrow$ decision
**return** decision

## VII. EXPERIMENTAL SETUP AND RESULTS

We evaluate the proposed fuzzy admission control framework on a single-node K8 cluster deployed inside a KVM-based VM. The node runs a Linux kernel with eBPF support and hosts all workloads locally. Table II summarizes the hardware and software configuration used in our experiments.

### A. Benchmark Applications

We run two DeathStarBench microservice applications and one AI inference workload on a single node.

- **Social Network**: DeathStarBench social networking service.[1]
- **Hotel Reservation**: DeathStarBench hotel search and reservation service.
- **Feedback Analysis**: Python-based sentiment-analysis inference application.[2]

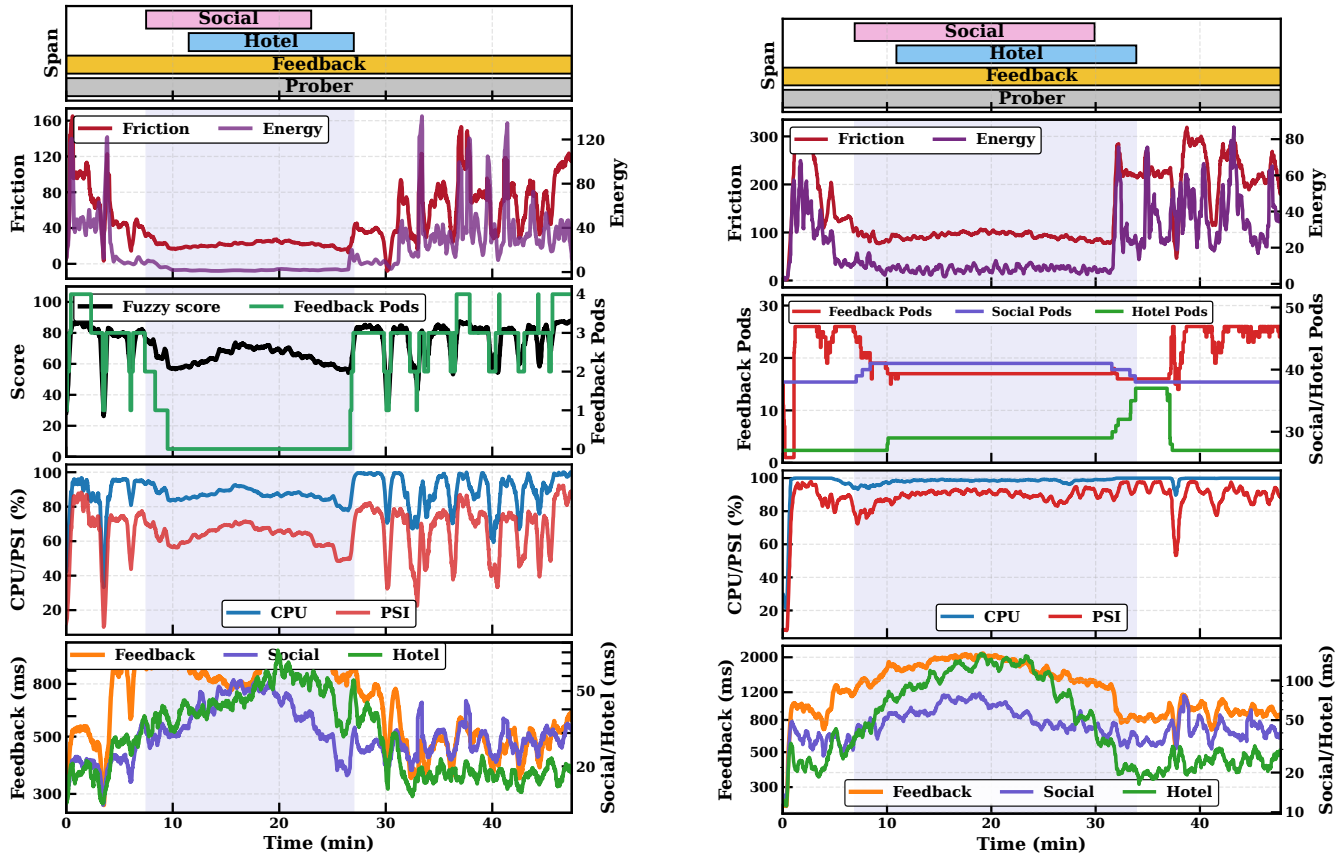
### B. Experimental Methodology

Each experiment is conducted for a duration of 1 h. In each run, the first 10 min are used to calibrate the friction metrics under normal system load. During this calibration phase, a controlled workload is applied such that no noticeable latency spikes occur. This phase allows the friction signal to establish a baseline and learn the normal behavior and covariance of latency-correlated kernel metrics, as defined in (2).

After the calibration phase, the experiment starts with a lightweight prober. The prober continuously sends a low, fixed-rate stream of requests (10 RPS) to each application and

[1]https://github.com/delimitrou/DeathStarBench

[2]https://github.com/BhanuPDas/CustomerFeedback




(a) (b)

Fig. 3. Comparison of fuzzy-based admission control and Horizontal Pod Autoscaler (HPA), illustrating their impact on QoS under varying workloads.

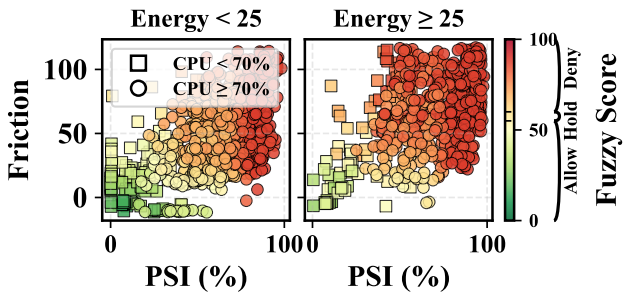



Fig. 4. Fuzzy controller behavior over PSI and friction. Color shows the fuzzy score, marker shape shows CPU utilization, and the two plots show low and high energy cases.

records the P99 latency at 1 s intervals. The prober operates as an observer and does not introduce significant additional load, providing a stable ground-truth latency measurement. Also, the feedback analysis application pods are always available for deployment on the node, with each pod running for a random duration of 2–4 min.

Following this, workload is introduced to the social network and hotel reservation applications by gradually increasing the request rate and later decreasing it. This varying load pattern emulates realistic client behavior. This workload generates resource contention on the node. During this phase, new workload instances are requested, and the fuzzy controller decides whether to admit or reject them.

We compare a fuzzy-controller approach with the Horizontal Pod Autoscaler (HPA). In both experiments, pods run without resource limits and can use all available resources of the node.

### C. Results and Comparison

This section shows only the production phase of the experiment and excludes the calibration phase. Shaded areas in Fig. 3 indicate when the Social and Hotel workloads are running. The data are smoothed in Fig. 3 to better show system behavior.

*1) Controller Behavior Under Contention:* Fig. 3a shows the fuzzy controller behavior. Before contention, Feedback pods are admitted gradually. During the shaded interval, when the Social and Hotel workloads are active, the controller stops admitting additional Feedback pods to protect QoS. Outside this interval, short spikes in friction, PSI, and energy are caused by pod admissions and removals. Fig. 4 shows that higher energy shifts controller scores toward higher-risk regions.

Fig. 3b shows the behavior of the HPA-based approach. HPA scales application pods based on CPU utilization. When

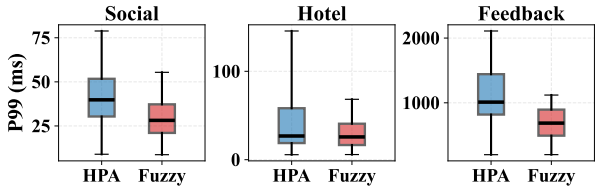



Fig. 5. P99 latency comparison between HPA and the fuzzy controller.

CPU usage exceeds 70 %, new instances are created. During the shaded region, the Social and Hotel workloads increase, causing their pod counts to scale up, while Feedback pods are not added due to limited available CPU. Once the Social and Hotel workloads stop, their pod counts decrease, which frees up resources. At this point, HPA begins scaling the Feedback application, resulting in increased latency outside the shaded region.

*2) Tail-Latency Comparison:* Fig. 5 compares the P99 latency of the fuzzy controller and HPA. In the Social application, the fuzzy controller achieves a lower median P99 latency of about 25 ms, while HPA is closer to 40 ms. For the Hotel application, both methods show a similar median P99 latency of around 35 ms. The largest difference is observed in the Feedback application, where HPA reaches nearly 1000 ms, whereas the fuzzy controller keeps the median P99 latency at about 600 ms. Overall, HPA shows a wider latency spread across all three applications, indicating that the fuzzy controller provides more stable tail latency.

The source code used in this work is publicly available at: https://github.com/abmuslim/fuzz-admission-controller.

## VIII. Conclusion and Outlook

This paper presents a fuzzy logic-based admission control framework for resource-constrained edge and fog nodes. It does not require application-level instrumentation. The framework uses non-intrusive kernel signals collected via eBPF to detect QoS degradation. These signals are combined into a single indicator called friction, which is then used to generate an energy signal. The friction and energy signals, along with CPU utilization and PSI, are used by a fuzzy controller to make admission decisions.

We compare the fuzzy controller approach with CPU-based Horizontal Pod Autoscaling (HPA). Under high load, the fuzzy controller reduces P99 latency by about 35–40 % and avoids large latency spikes, leading to more stable QoS. This is because the proposed friction signal detects system jitter and allows the controller to better control tail latency.

Future work will explore two main directions. First, we plan to test the proposed controller on a multi-node Kubernetes cluster to understand how it behaves when workloads are distributed across several nodes. Second, this work focuses on admission control, but it can be extended to handle workloads that are already running. The system could detect workloads that use too many resources and take action to prevent QoS degradation. Managing both new and running workloads could further improve QoS stability.

## Acknowledgment

Supported by the EMULATE project, funded by the German Federal Ministry for Economic Affairs and Climate Action (13IPC012).

## References

[1] J. Bachiega Jr., B. Costa, L. R. Carvalho, M. J. F. Rosa, and A. Araujo, "Computational resource allocation in fog computing: A comprehensive survey," *ACM Computing Surveys*, vol. 55, no. 14s, Art. no. 336, pp. 1–31, July 2023, doi: 10.1145/3586181.

[2] X. Zhang and S. Debroy, "Resource management in mobile edge computing: A comprehensive survey," *ACM Computing Surveys*, vol. 55, no. 13s, Art. no. 291, pp. 1–37, Dec. 2023, doi: 10.1145/3589639.

[3] S. Volpert, S. Winkelhofer, S. Wesner, D. Seybold, and J. Domaschka, "Exemplary determination of cgroups-based QoS isolation for a database workload," in *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion)*, London, United Kingdom, May 2024, pp. 235–241, doi: 10.1145/3629527.3652267.

[4] J. Huang, B. Lv, Y. Wu, Y. Chen, and X. Shen, "Dynamic admission control and resource allocation for mobile edge computing enabled small cell network," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 2, pp. 1964–1973, Feb. 2022, doi: 10.1109/TVT.2021.3133696.

[5] S. A. Javadi, A. Suresh, M. Wajahat, and A. Gandhi, "Scavenger: A black-box batch workload resource manager for improving utilization in cloud environments," in *Proc. ACM Symposium on Cloud Computing (SoCC '19)*, Santa Cruz, CA, USA, Nov. 2019, pp. 272–285, doi: 10.1145/3357223.3362734.

[6] eBPF Foundation, "What is eBPF?," *eBPF.io*. [Online]. Available: https://ebpf.io/what-is-ebpf/.

[7] Facebook, "Pressure Stall Information," 2019. [Online]. Available: https://facebookmicrosites.github.io/psi/docs/overview.

[8] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance metric learning, with application to clustering with side-information," in *Proc. 16th Int. Conf. Neural Information Processing Systems (NIPS)*, Cambridge, MA, USA, pp. 521–528, 2002.

[9] P. C. Mahalanobis, "On the generalised distance in statistics," *Sankhyā A*, vol. 80, suppl. 1, pp. 1–7, 2018, reprint of the 1936 original.

[10] Y. Gan *et al.*, "An open-source benchmark suite for microservices and their hardware–software implications for cloud and edge systems," in *Proc. 24th Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Providence, RI, USA, 2019, pp. 3–18, doi: 10.1145/3297858.3304013.

[11] L. Tomás and J. Tordsson, "An autonomic approach to risk-aware data center overbooking," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 292–305, Jul.–Sep. 2014, doi: 10.1109/TCC.2014.2326166.

[12] S. K. Sahoo, A. Dash, D. R. Vemula, C. K. Swain, and S. K. Mishra, "Latency-aware resource planning in edge using fuzzy logic," in *Proc. 2nd Int. Conf. Ambient Intelligence in Health Care (ICAIHC)*, Bhubaneswar, India, 2023, pp. 1–6, doi: 10.1109/ICAIHC59020.2023.10431439.

[13] S. Chen, L. Wang, and F. Liu, "Optimal Admission Control Mechanism Design for Time-Sensitive Services in Edge Computing," in *Proc. IEEE INFOCOM*, London, UK, 2022, pp. 1169–1178. doi: 10.1109/INFOCOM48880.2022.9796847.

[14] T. Ouyang, K. Zhao, G. Hong, X. Zhang, Z. Zhou, and X. Chen, "Dynamic Edge-Centric Resource Provisioning for Online and Offline Services Co-Location via Reactive and Predictive Approaches," *IEEE/ACM Transactions on Networking*, vol. 33, no. 3, pp. 1388–1403, June 2025. doi: 10.1109/TON.2025.3535499.

[15] Y. Han, S. Shen, X. Wang, S. Wang, and V. C. M. Leung, "Tailored Learning-Based Scheduling for Kubernetes-Oriented Edge-Cloud System," in *Proc. IEEE INFOCOM*, Vancouver, BC, Canada, 2021, pp. 1–10. doi: 10.1109/INFOCOM42981.2021.9488701.

[16] X. Foukas and B. Radunović, "Concordia: Teaching the 5G vRAN to Share Compute," in *Proc. ACM SIGCOMM*, New York, NY, USA, 2021, pp. 580–596. doi: 10.1145/3452296.3472894.