PCDA: Prediction-opinion based container Caching Decision Algorithm in Serverless Edge Environment

Tianhao Wu
FEECS, Ningbo University
Ningbo, China

Haiming Chen*
FEECS, Ningbo University
Ningbo, China
chenhaiming@nbu.edu.cn

Peihao Wu
FEECS, Ningbo University
Ningbo, China

2025 International Conference on Sensor-Cloud and Edge Computing System (SCECS) | 979-8-3315-2068-7/25/$31.00 ©2025 IEEE | DOI: 10.1109/SCECS65243.2025.11065016

***Abstract*—As the serverless computing model is integrated into edge environments, it enhances the flexibility and scalability of IoT systems, significantly improving the Quality of Service (QoS) for Function as a Service (FaaS) users. However, serverless edge computing faces several challenges, with one primary issue being the latency caused by container cold starts. While container caching mitigates, determining optimal caching duration in resource-constrained edge environments requires balancing conflicting resource and latency costs. We formulate this fundamental trade-off through a ski-rental problem lens and propose the Prediction-Opinion based Caching Decision Algorithm (PCDA). Our solution introduces a *Dual-phase caching* with deterministic keep-alive windows and prediction-driven pre-warm windows, and a confidence metric λ that dynamically calibrates decision randomness against prediction errors. Extensive simulation experiments demonstrated that, compared to existing algorithms, PCDA performed best under various caching conditions, with an overall system cost reduction of 12.04% to 73.97%.**

***Index Terms*—edge computing, serverless, container caching**

I. Introduction

Cloud computing has accelerated development of numerous industries with its robust computing capabilities and flexible resource management [1]. However, as the popularity and number of IoT devices increase, there is a growing demand for real-time data processing and low-latency responses [2]. Traditional cloud computing has shown its obvious limitations, which has prompted the evolution of a more decentralized approach called edge computing [3]–[6]. Edge computing, as an extension of cloud computing, allocates computing resources and services closer to the data source at the edge of the network, thereby substantially reducing data transmission latency, providing latency assurance services to end users.

As an emerging cloud computing model, serverless computing further enhances the flexibility of accessing cloud services [7]–[9]. Serverless computing is primarily implemented as Function as a Service (FaaS), which decouples application development and runtime environments from the underlying infrastructure. By adhering to an event-based model, serverless computing achieves high elasticity of cloud resources while maintaining excellent programmability. In serverless computing, services are encapsulated in lightweight containers and are invoked only when an event triggers a service request. In recent years, serverless computing platforms, such as AWS Lambda [10], have expanded to edge computing. Serverless edge computing is considered a key development direction for future computing architectures.

Serverless edge computing terminates containers post-event processing, inducing cold-start delays (100ms–seconds) from resource provisioning and function loading [9]. With 52% of Azure functions executing in $<$1s [11], cold starts often exceed service durations, degrading QoS and causing failures in latency-sensitive applications.

We classify current methods for mitigating container cold start latency into three categories: container pooling, container preheating, and container reuse. Container pooling minimizes preparation time by maintaining a pool of ready-to-use containers. Container preheating involves warming up containers in anticipation of incoming requests, while container reuse focuses on caching containers to handle future requests more efficiently. In resource-constrained edge environments, container reuse proves particularly effective for reducing cold-start latency [13] [14].

In this paper, we attempt to mitigate the cold start latency by caching containers by addressing the following challenges. Firstly, caching container consumes additional resources, which conflicts with the fundamental elastic principle of serverless computing [15] to utilize resources only when necessary. Secondly, only a small subset of serverless functions are invoked frequently. Prolonged caching of infrequently called functions leads to resource wastage. Especially in edge environments with limited computing resources, the trade-offs between resource consumption and latency reduction are even more pronounced. Balancing the costs of caching containers with the benefits of reduced latency is a challenging task [16].

All this challenge highlights the need for a holistic approach to container caching.

The problem addressed in this paper encounters several challenges: (1) Edge nodes vary in initializing containers, processing function requests, and caching containers. This heterogeneity leads to different costs across nodes, necessitating diverse caching strategies for different functions and nodes. (2) Function request patterns may not be accurately predictable, especially for functions with frequently changing invocation patterns. Existing prediction methods may not adapt well to such dynamic patterns, resulting in less robust caching strategies.

In summary, this paper has made the following major contributions:

- We map the container caching problem to a classic ski rental problem and establish a trade-off model for cold start latency and cache consumption.
- We propose a lightweight hybrid forecasting model for more time edge environments, which can better predict the time series data requested by users.
- We design a Prediction Opinion-based Caching Decision Algorithm (PCDA), which is based on prediction opinions. It is worth noting that this algorithm remains robust even in cases of prediction failure.
- Numerous simulation experiments validate the feasibility and superiority of our caching strategy.

The remainder of this paper systematically addresses these innovations: Section II formalizes our background and motivation . Section III details of PCDA Section IV validates our approach through comprehensive simulations.

II. Background and Motivation

In this section, we first describe how the container caching problem maps to the ski rental problem, and then we build a consumption model that trades the container cache cost against the cold start cost.

A. Mapping to the Ski-Rental Problem

In serverless computing environments, the system must determine a caching duration for containers upon completing request execution. This critical decision specifies how long the container remains active in memory, during which it incurs time-proportional caching costs. If the container persists in cache when subsequent requests arrive, it eliminates cold-start latency; otherwise, the system must pay the penalty of cold-start initialization delays.This fundamental trade-off mirrors the classic ski-rental problem [16]–[19]. The ski-rental problem describes such a scenario where a skier is faced with a to-rent-or-to-buy struggle. Since the suitable days for skiing is unknown, the skier has to choose between renting a pair of skis or simply buying it. Assume the cost of renting the ski and the cost of buying it. The skier has to make the decision for the current skiing season such that the total cost is minimized. As illustrated in Fig. 1, the analogy establishes precise mathematical correspondence:

- **Ski Purchase Cost** mapping to the cold start delay cost, the purchase corresponds to the destruction of the container, so the container initialization overhead is paid when the next request arrives
- **Daily Rental Fee** corresponds to caching cost , accruing per-time-unit while maintaining container availability
- **Unknown Ski Days** reflects the stochastic nature of next request arrival , which remains unpredictable when making caching decisions

The problem's core dilemma persists: Premature container destruction risks repeated cold-start penalties, while excessive caching wastes resources through unnecessary reservation of memory capacity. This tension between immediate latency avoidance and long-term cost efficiency forms the optimization crux in container lifecycle management.

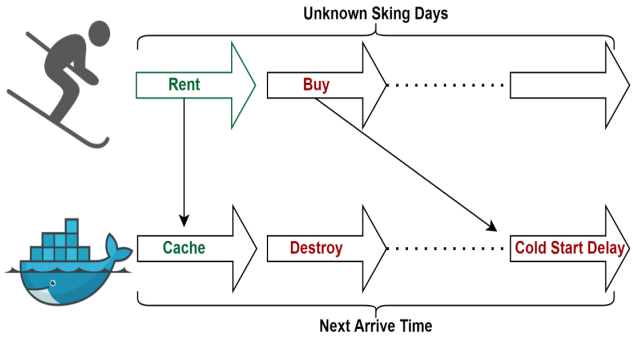

Fig. 1. Mapping of the container caching problem to the ski-rental problem.

B. Cost Model

Compared to the traditional ski-rental problem model, if we can predict the time interval until the next user request, we can introduce a pre-warm decision process in addition to the basic caching decision. Therefore, we establish two decision windows for container lifecycle management:

- **Keep-alive Window** ($T_k \geq 0$): Mandatory caching duration (in time units) immediately after request execution, providing deterministic cold-start avoidance
- **Pre-warm Window** ($T_p \geq 0$): Prediction-driven time gap (in time units) between container destruction and proactive re-caching, balancing resource conservation and latency prevention

Container-specific Parameters:

- M_i: Memory footprint of container type i
- $M_{\min} = \min\{M_i\}$: Smallest container memory in the system
- T_i: Cold-start initialization time for container type i (in time units)

Under this decision framework, the container lifecycle consists of three distinct phases, as illustrated in Fig. 2:

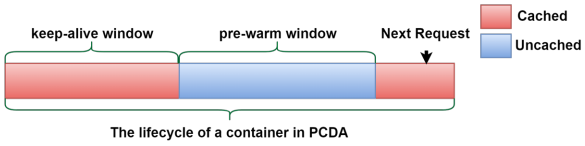

Fig. 2. Container lifecycle phases: Active caching (T_k), Risk window (T_p), and Re-caching phase.

Let $c_i = M_i/M_{\min}$ denote the *normalized caching cost rate* per time unit, $B_i = T_i$ denote the *cold-start penalty* for container type i, and X the random variable representing time until next request arrival. The total cost C is:

$$C(T_k, T_p, X) = \begin{cases} c_i X, & X \leq T_k \\ c_i T_k + B_i, & T_k < X \leq T_k + T_p \\ c_i (X - T_p), & X > T_k + T_p \end{cases} \quad (1)$$

Let β be the proportionality coefficient ensuring the normalized caching cost does not exceed $1/\beta$ of the cold-start penalty, leading to the constraint:

$$\beta c_i \leq B_i \quad \forall i \quad (2)$$

This constraint guarantees that for each container type, the normalized caching cost per time unit does not exceed a β-adjusted fraction of its cold-start penalty. If $\beta c_i > B_i$, caching for one time unit would be more expensive than paying the cold-start penalty, rendering caching economically irrational for type i.

The expected total cost integrates over the request interval distribution $f_X(x)$:

$$\begin{aligned} \mathbb{E}[C] = c_i \int_0^{T_k} x f_X(x) dx \quad \text{(Active caching)} \\ + (c_i T_k + B_i) \int_{T_k}^{T_k + T_p} f_X(x) dx \quad \text{(Risk period)} \\ + c_i \int_{T_k + T_p}^{\infty} (x - T_p) f_X(x) dx \quad \text{(Re-caching)} \end{aligned} \quad (3)$$

C. Key Challenges

After mapping the container caching issue to the ski rental problem, we identify two critical challenges in optimizing container lifecycle management:

Uncertainty in dynamic request patterns: Accurate prediction of request intervals is essential but challenging in edge environments. While LSTM networks achieve 95% prediction accuracy [20], their high computational cost makes them impractical for resource-constrained edge nodes. [21]–[23]illustrate alternative prediction paradigms with lower dimensional requirements, but they may not be suitable for a wide variety of function types. Developing lightweight yet effective prediction models remains a key challenge.

Balancing prediction and real-time responsiveness: The classic ski-rental dilemma offers two choices: continuous caching or cold starts. We introduce a third option—a prediction-driven *pre-warm window*—that can potentially avoid both extremes. However, this approach introduces new risks:

- Over-reliance on predictions may lead to unnecessary caching costs
- Prediction errors could simultaneously incur cold-start penalties and wasted caching
- Rapid decision-making conflicts with computationally intensive prediction methods

This dual challenge requires an adaptive algorithm that dynamically balances prediction accuracy with real-time responsiveness, while maintaining low computational overhead in resource-constrained edge environments.

III. Algorithm Design

In this section, we will provide a detailed explanation of how the PCDA is designed. We first develop a hybrid prediction model that effectively captures the temporal patterns in user request sequences. The predictions are then integrated as input with a classical randomized algorithm, where a confidence metric λ (0-1) is introduced to ensure the algorithm's robustness against prediction errors.

A. Hybrid Prediction Model

To address the limitations of single prediction methods and improve computational efficiency, we develop a hybrid prediction model that combines ARIMA and TES (Triple Exponential Smoothing). As illustrated in Fig. 3, this hybrid approach leverages the complementary strengths of both methods: ARIMA effectively captures linear relationships in the data while TES excels at identifying nonlinear trends. The combination enables more accurate and robust prediction of function request arrival times across diverse workload patterns.

The hybrid prediction is computed as:

$$\hat{X} = w_{\text{ARIMA}} \cdot \hat{X}^{\text{ARIMA}} + w_{\text{TES}} \cdot \hat{X}^{\text{TES}} \quad (4)$$

where $\hat{X}^{\text{ARIMA}}$ and $\hat{X}^{\text{TES}}$ denote the predictions from ARIMA and TES models respectively.

The weights are dynamically adapted based on historical prediction errors:

$$w^i = \frac{\exp(-\eta \sum \epsilon^i)}{\sum_{j \in \{\text{ARIMA,TES}\}} \exp(-\eta \sum \epsilon^j)} \quad (5)$$

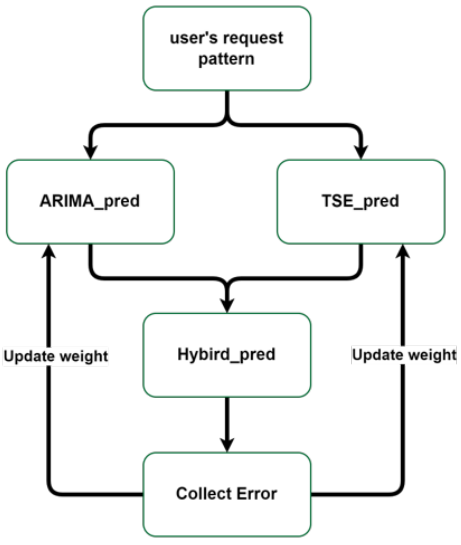

Fig. 3. Hybrid prediction workflow with dynamic weight adaptation

This hybrid design achieves two key benefits: adaptability to changing workload patterns, and reduced computational overhead compared to deep learning approaches. where ϵ represent the mean absolute error of historical predictions,calculated as $\epsilon = \frac{1}{N} \sum_{i=1}^{N} |X_i - \hat{X}_i|$.

B. Prediction-Opinion Based Caching Decision Algorithm

As shown in Algorithm 1, the inputs include the predicted time for the next request ($\hat{X}$), the costs associated with caching (c) and cold starts (B), mean absolute error of historical predictions ϵ, average real value ($\mathbb{E}[X]$), and the current system parameter β. The algorithm outputs a keep-alive window (T_k) and a pre-warm window (T_p).

Due to high variability in invocation patterns for some functions, predictions may sometimes be inaccurate. To address this, we introduce a confidence metric $\lambda \in [0, 1]$, calculated as $\lambda = \min(\epsilon/\mathbb{E}[X], 1)$. The parameter $T_{\text{SR}} = B/c$ denotes the caching decision threshold in the ski rental model.

The algorithm begins by comparing $\hat{X}$ with T_{SR} to guide caching or deletion of the container. λ plays a critical role in adjusting the algorithm's behavior, where it influences the range of random selection. Probabilities are assigned to each integer, allowing dynamic selection of T_k based on computed confidence.

When $\lambda = 0$, the algorithm fully trusts predictions, leading to immediate container deletion or indefinite caching. Conversely, $\lambda = 1$ reflects complete distrust in predictions, introducing maximal randomization.

The PCDA leverages accurate predictions by pre-warming containers when $\lambda < 0.2$, calculating $T_p = \max(\hat{X} - T_k + \epsilon, 0)$. To mitigate prediction errors, pre-warming duration is bounded by T_{SR}, aligning with the next expected request timing.

Finally, the algorithm's robustness is strengthened through λ, and performance evaluations confirm that, even under complete prediction failures, PCDA achieves better outcomes than random caching strategies.

Complexity Analysis: The time complexity of PCDA is dominated by the probability distribution generation loops. Let $k = \lfloor \lambda T_{\text{SR}} \rfloor$ and $l = \lceil T_{\text{SR}}/\lambda \rceil$ denote the maximum iterations in each branch. The worst-case complexity is $O(\max(k, l))$ where:

- $k \leq T_{\text{SR}}$ since $\lambda \in [0, 1]$
- $l \leq T_{\text{SR}}/\lambda_{\text{min}}$ for minimal confidence λ_{min}

In practice, the $T_{\text{SR}} = B/c$ ratio remains bounded as cold-start penalties (B) and caching costs (c) are positively correlated in real deployments. Empirical measurements show linear scaling with T_{SR} (typically $T_{\text{SR}} \leq 50$ slots in our experiments), making the algorithm suitable for real-time decision making.

Algorithm 1: Prediction Opinion-Based Caching Decision Algorithm (PCDA)

Input: $\hat{X}$, c, B, ϵ, $\mathbb{E}[X]$, β
Output: T_k, T_p

$\lambda \leftarrow \min\left(\frac{\epsilon}{\mathbb{E}[X]}, 1\right)$;
$T_{\text{SR}} \leftarrow B/c$;
$c_{\text{eff}} \leftarrow \beta c$;
if $\hat{X} \geq T_{SR}$ **then**
  $k \leftarrow \lfloor \lambda T_{\text{SR}} \rfloor$;
  for $j = 1$ **to** k **do**
    $p_j \leftarrow \left(\frac{T_{\text{SR}} - c_{\text{eff}}}{T_{\text{SR}}}\right)^{k-j} \cdot \frac{c_{\text{eff}}}{T_{\text{SR}}\left(1 - \left(1 - \frac{c_{\text{eff}}}{T_{\text{SR}}}\right)^{k}\right)}$;
  end
  Choose T_k from $\{1, ..., k\}$ with $\{p_j\}$;
  if $\lambda \leq 0.2$ **then**
    $T_p \leftarrow \max(\hat{X} - T_k + \epsilon, 0)$;
  else
    $T_p \leftarrow -1$;
  end
else
  $l \leftarrow \lceil T_{\text{SR}}/\lambda \rceil$;
  for $i = 1$ **to** l **do**
    $q_i \leftarrow \left(\frac{T_{\text{SR}} - c_{\text{eff}}}{T_{\text{SR}}}\right)^{l-i} \cdot \frac{c_{\text{eff}}}{T_{\text{SR}}\left(1 - \left(1 - \frac{c_{\text{eff}}}{T_{\text{SR}}}\right)^{l}\right)}$;
  end
  Choose T_k from $\{1, ..., l\}$ with $\{q_i\}$;
  $T_p \leftarrow -1$;
end
return T_k, T_p;

IV. Performance Evaluation

In this chapter, we begin by individually testing the robustness of the Prediction opinion-based Caching Decision Algorithm (PCDA), conclude with a comprehensive evaluation of the overall performance of PCDA.

A. Simulation Settings

To model hardware heterogeneity, we configure five edge nodes with CPU frequencies $f_i \in [2.4, 3.6]$ GHz sampled uniformly. The cold-start penalty B_i inversely scales with node capability:

$$B_i = B_{\text{base}} \times \frac{2.4}{f_i} \tag{6}$$

where B_{base} denotes the baseline cold-start penalty when running on 2.4GHz nodes. Container memory footprints M and cold-start durations T_c are generated across 20 function types with:

- Cold-start time: $T_c \sim \mathcal{U}(10, 50)$ time slots
- Memory requirement: $M \sim \mathcal{U}(140, 400)$ MB

Request patterns follow our original event modeling approach with:

- Function popularity: Zipf distribution ($\alpha = 0.5$–1.5)
- Request arrival: Poisson process ($\lambda = 0.05$–0.15)
- Evaluation scale: 10K requests per run (10 iterations)

This parameterization captures three key dimensions: 1) hardware heterogeneity through CPU frequency variations, 2) functional diversity via memory/cold-start requirements, and 3) realistic invocation patterns via heavy-tailed popularity distributions.

B. Robustness Verification of PCDA

To verify the robustness of the caching decision algorithm, this study selects the request pattern of a single user as the request data. Five sets of control request data were used as the input of the algorithm:

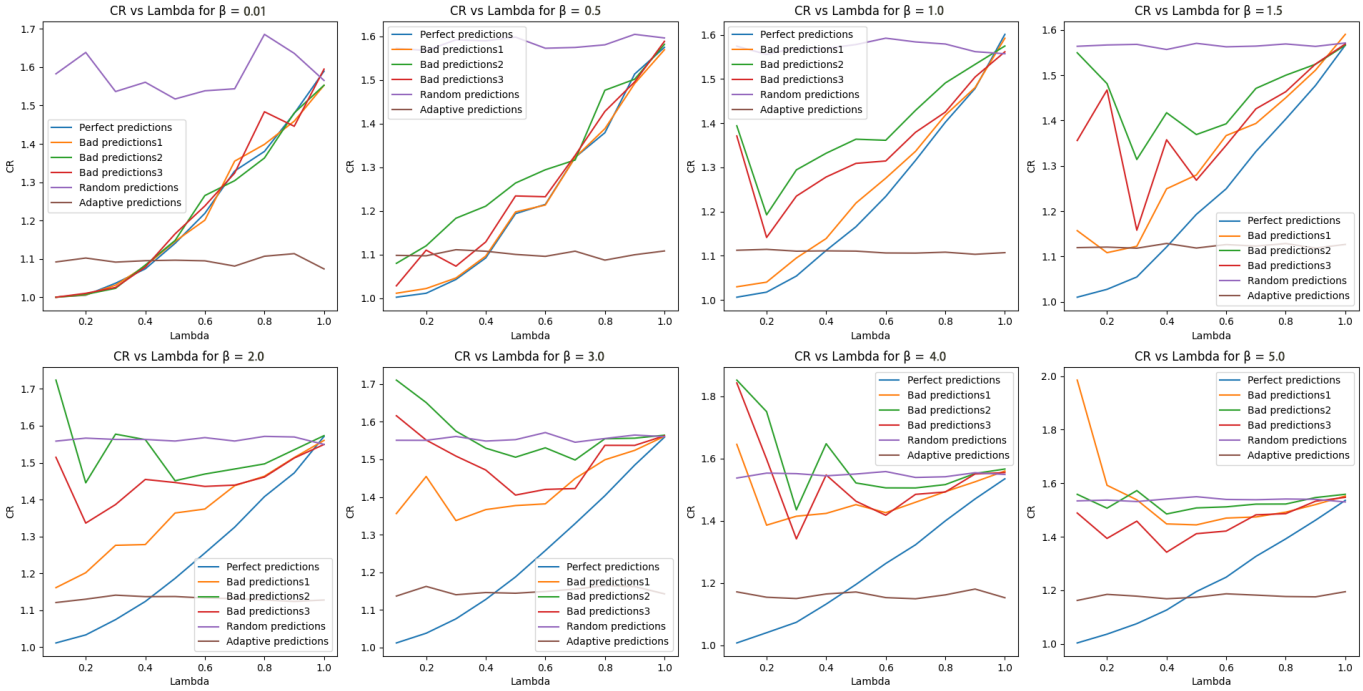

Fig. 4. Competitive ratio under different β values

- **Perfect Prediction Group**: The predicted data is the actual data.
- **Bad Prediction One**: The actual data minus the average of the actual data (adding a negative offset).
- **Bad Prediction Two**: The actual data plus the average of the actual data (adding a positive offset).
- **Bad Prediction Three**: Using the average of the data as the prediction value.
- **Random Group**: In this group, the confidence value is always set to one. This group is mainly used to contrast the performance with the other groups.
- **Adaptive Group**: In this group, we used the real predictions as the input for PCDA.The confidence value is set as in PCDA, i.e., the first line of Algorithm 1.

To evaluate performance, we employ the *competition ratio* (CR), defined as the cost ratio between our algorithm and the optimal offline strategy. The optimal strategy caches containers until the next request if it arrives before $t + \frac{b_n^k}{\beta q_n^k}$, otherwise destroys them immediately. All results are averaged over 10 iterations. As shown in Fig. 4, we observe:

Impact of Bad Predictions: When the value of β is small, the caching cost is low, and the negative impact of bad predictions on the algorithm's performance is minimal because the algorithm tends to make conservative decisions regardless of the prediction results. As β increases, the caching cost also increases, and the negative impact of bad predictions becomes smaller because the algorithm tends to make more aggressive deletion decisions.

Impact of Adaptive λ: The Adaptive Group consistently maintains a lower CR, with performance fluctuations of less than 0.1 across different β values. This indicates that the adaptive confidence value calculation method we designed is successful, and it ensures the robustness of PCDA.

Overall Analysis: Across different β values, all groups with poor prediction accuracy can still achieve performance comparable to the completely random group by selecting an appropriate confidence level. This indicates that even when the prediction module fails, PCDA performance can be maintained by tuning the confidence value.

C. Performance Verification of PCDA

To verify the performance of our caching decision algorithm, we use the following baseline methods for comparison:

- **Ski Rental (SR)**: Classic Ski Rental algorithm without pre-warming
- **Fixed Caching (FC)**: AWS Lambda-style fixed duration caching
- **Histogram (HIST)** [11]: History-based caching from Azure Functions
- **Reinforcement Learning (RL)** [12]: Q-learning approach with cost rewards
- **Pre Warm (PW)** [21]: Prediction-only pre-warming
- **Keep Warm (KW)**: Always-cache strategy

Overall Performance Summary: From Fig. 5, PCDA demonstrates: Maximum 73.97% cost reduction vs KW at $\beta = 3.0$ 52.09% improvement over FC 46.34% better than HIST in medium β range 12.04% advantage vs SR

Effect of parameter β: Low β (≤ 1.0): Caching dominates (PW closest to PCDA) Medium β (1.0-3.0): PCDA's adaptive

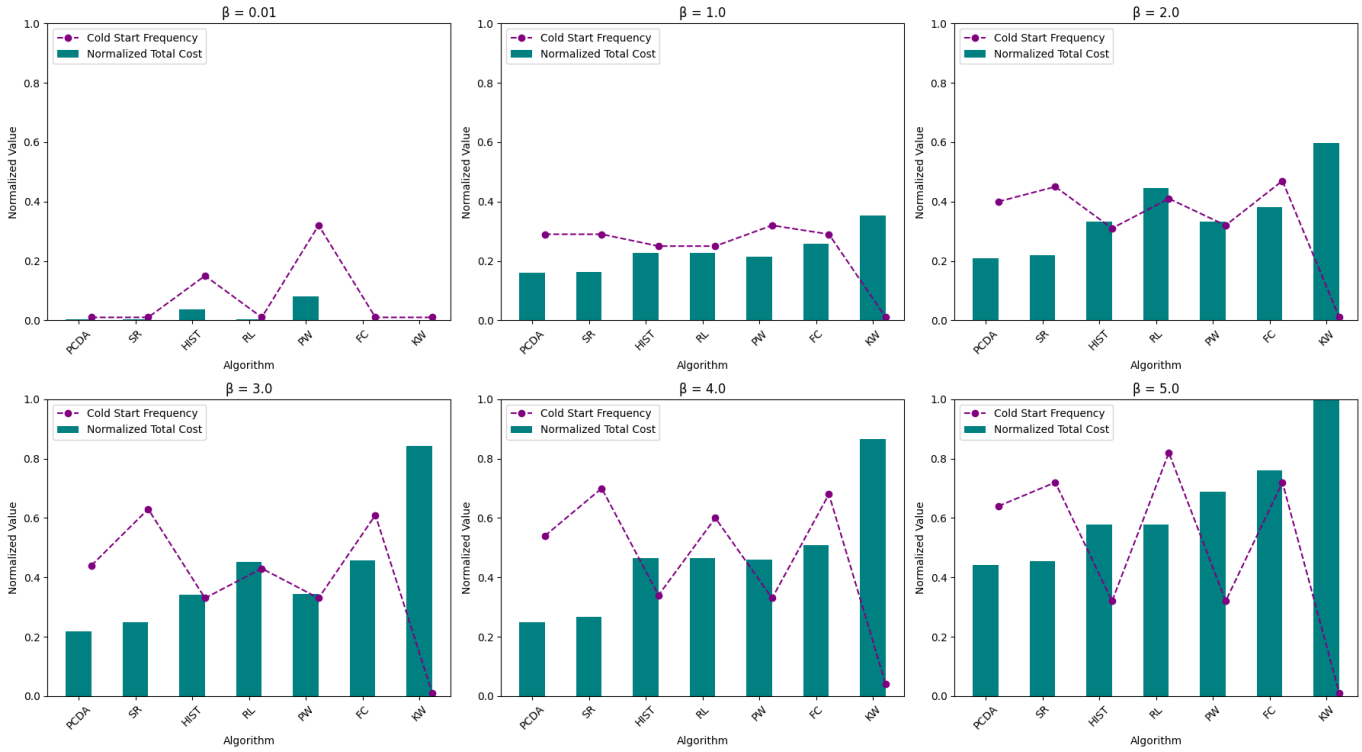

Fig. 5. Total cost under different β Values (PCDA vs Baselines)

λ shows maximum benefit High β (≥ 3.0): Strategies converge as caching cost dominates

Individual Algorithm Analysis:PCDA: Maintains 2-competitiveness through:Dynamic λ adaptation (0.2 threshold), Error-compensated pre-warming ($T_p = \max(\hat{X} - T_k + \epsilon, 0)$), Probabilistic keep-alive selection SR: Lacks pre-warming capability, 12% higher cold starts HIST: Suffers 32% error rate with shifting request patterns RL: Requires 500+ episodes for Q-value convergence PW: Prediction failures cause 41% cost spikes FC: Inflexible for bursty workloads KW: Linear cost growth with β (no adaptation)

Cold Start Analysis: PCDA achieves **58% lower cold starts** than SR ,HIST/PW have 0.3 cold start probability due to prediction errors,RL reduces cold starts by 22% after convergence

V. Conclusion

This paper proposes PCDA, an adaptive container caching strategy for serverless edge computing that addresses the cold-start versus resource trade-off by formalizing it as a ski-rental problem. Our solution introduces dual-phase caching (keep-alive and prediction-driven pre-warm windows) with a dynamic confidence metric λ to balance deterministic caching and prediction-based optimization. The lightweight hybrid model combining ARIMA and TES enables accurate request forecasting while minimizing computational overhead. Extensive simulations demonstrate PCDA reduces total system costs by 12.04%–73.97% across diverse edge configurations, outperforming existing approaches through robust cold-start mitigation and efficient memory utilization.

Acknowledgment

This work was supported by the Natural Science Foundation of Zhejiang Province (No. LY24F020001), Ningbo 2035 Key Technology Breakthrough Program (2024Z145), and Major S&T Projects of Ningbo High-tech Zone (No. 2022BCX05001).

References

[1] V V. Arutyunov. Cloud computing: Its history of development, modern state, and future considerations. Scientific and Technical Information Processing, 2012, 39: 173-178.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, et al. Internet of things: A survey on enabling technologies, protocols, and applications. IEEE communications surveys and tutorials, 2015, 17(4): 2347-2376.

[3] M. Satyanarayanan, N. Davies. Augmenting cognition through edge computing. Computer, 2019, 52(7): 37-46.

[4] L. Peterson, T. Anderson, S. Katti, et al. Democratizing the network edge. ACM SIGCOMM Computer Communication Review, 2019, 49(2): 31-36

[5] M. Li, Q. Zhang, F. Liu. Finedge: A dynamic cost-efficient edge resource management platform for NFV network. 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS). IEEE, 2020: 1-10.

[6] Q. Chen, Z. Zheng, C. Hu, et al. On-edge multi-task transfer learning: Model and practice with data-driven task allocation. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(6): 1357-1371.

[7] N. Kratzke. A brief history of cloud application architectures. Applied Sciences, 2018, 8(8): 1368.

[8] I. Baldini, P. Castro, K. Chang, et al. Serverless computing: Current trends and open problems. Research advances in cloud computing, 2017: 1-20.

[9] M. Sewak, S. Singh. Winning in the era of serverless computing and function as a service. 2018 3rd International Conference for Convergence in Technology (I2CT). IEEE, 2018: 1-5
[10] AWS Lambda. Available online: https://aws.amazon.com/lambda/, 2019.
[11] M. Shahrad, R. Fonseca, I. Goiri, et al. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. 2020 USENIX annual technical conference (USENIX ATC 20). 2020: 205-218.
[12] P. Vahidinia, B. Farahani, F. S. Aliee. Mitigating cold start problem in serverless computing: a reinforcement learning approach. IEEE Internet of Things Journal, 2022, 10(5): 3917-3927.
[13] K. Xiao, S. Yang, F. Li, et al. Making Serverless Not So Cold in Edge Clouds: A Cost-Effective Online Approach. IEEE Transactions on Mobile Computing, 2024.
[14] E. Jonas, J. Schleier-Smith, V. Sreekanti, et al. Cloud programming simplified: A Berkeley view on serverless computing. arXiv preprint arXiv:1902.03383, 2019.
[15] L. Pan, L. Wang, S. Chen, et al. Retention-aware container caching for serverless edge computing. IEEE INFOCOM 2022-IEEE Conference on Computer Communications. IEEE, 2022: 1069-1078.
[16] V. Farhadi, F. Mehmeti, T. He, et al. Service placement and request scheduling for data-intensive applications in edge clouds. IEEE/ACM Transactions on Networking, 2021, 29(2): 779-792.
[17] B. Gao, Z. Zhou, F. Liu, et al. Winning at the starting line: Joint network selection and service placement for mobile edge computing. IEEE INFOCOM 2019-IEEE conference on computer communications. IEEE, 2019: 1459-1467.
[18] S. Wang, J. Li, S. Wang. Online algorithms for multi-shop ski rental with machine learned advice. Advances in Neural Information Processing Systems, 2020, 33: 8150-8160.
[19] Y. Xie, M. Jin, Z. Zou, et al. Real-time prediction of docker container resource load based on a hybrid model of ARIMA and triple exponential smoothing. IEEE Transactions on Cloud Computing, 2020, 10(2): 1386-1401.
[20] C. Chen, L. Nagel, L. Cui, et al. S-cache: Function caching for serverless edge computing. Proceedings of the 6th International Workshop on Edge Systems, Analytics and Networking, 2023: 1-6.
[21] R. B. Roy, T. Patel, D. Tiwari. Icebreaker: Warming serverless functions better with heterogeneity. Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2022: 753-767.
[22] J. Tan, W. Liu, T. Wang, et al. A high-accurate content popularity prediction computational modeling for mobile edge computing using matrix completion technology. Transactions on Emerging Telecommunications Technologies, 2021,32(6):e3871.
[23] T. Wang, Z. Peng, J. Liang, et al. Following targets for mobile tracking in wireless sensor networks. ACM Transactions on Sensor Networks, 2016,12(4):1-24.