Check #1: Mitigating Serverless Cold Starts in Edge Environments via Lightweight Change-Detection Heuristics

Fengcheng Yu*
fyu54107@usc.edu

Tian Xie*
xietian@usc.edu

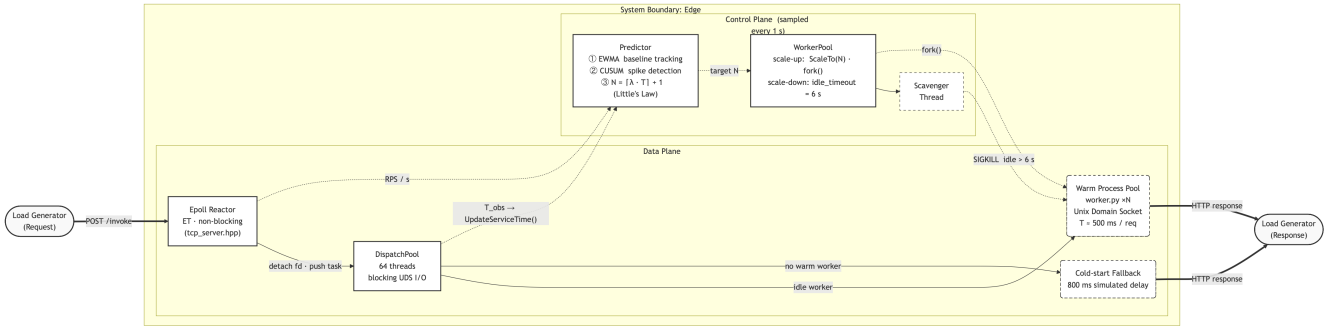

Figure 1: Architecture of the lightweight edge Serverless node. The system decouples the asynchronous Epoll Data Plane from a mathematically rigorous Control Plane. The Predictor leverages EWMA/CUSUM and Little's Law to dynamically pre-provision OS-level processes, ensuring sub-millisecond hot starts while eliminating the overhead of heavy orchestration frameworks.

1 The Specific Problem (Updated)

Serverless computing (FaaS) at the Network Edge faces a critical trade-off between tail latency (caused by cold starts) and memory waste (caused by over-provisioning) [11, 12]. While heavy orchestration frameworks and complex time-series forecasting models (e.g., LSTMs [6], ARIMA [2]) can predict cyclic workloads to mitigate cold starts, they introduce non-trivial inference overhead (CPU/Memory) [10]. Therefore, the specific problem we address is evaluating the overhead of the prediction control plane itself. Rather than reinventing data plane isolation (where MicroVMs like AWS Firecracker [1] are the industry standard), our project isolates and evaluates a lightweight, mathematically rigorous control plane algorithm to achieve near-optimal pre-warming with near-zero system overhead. The GitHub repository for this project is available at: https://github.com/ffengc/edge-faas-cpp. It contains comprehensive experimental setups, code configurations, and detailed development logs documenting our thought process across multiple testing iterations.

2 Solution Approach & Evolution

Over the past month, our solution approach has evolved significantly based on theoretical refinement and initial testing. **What didn't work:** We initially proposed a vague "histogram-based" prediction and lacked a rigorous mathematical foundation for scaling. We quickly realized this was too primitive for accurate sudden-change detection and lacked a theoretical basis for deciding how many workers to pre-provision.

Our refined approach: To fix this, we completely dropped the histogram concept and upgraded our core engine to use EWMA (Exponentially Weighted Moving Average) [9] combined with CUSUM (Cumulative Sum Control Chart) [8]. This combination tracks the Request Arrival Rate (RPS) in $O(1)$ time and accurately triggers alarms for sudden spikes. Furthermore, we integrated Little's Law ($N = \lambda \times T$) [7] into our control plane to dynamically calculate the exact number of instances to pre-provision based on the predicted RPS (λ) and dynamically tracked average service time (T), plus a safety margin.

As illustrated in Figure 1, to strictly measure the overhead of this control plane without hypervisor noise, we successfully built a custom, event-driven HTTP gateway from scratch using C++ Epoll (ET mode). We bypassed heavy VMs and leveraged OS-level process primitives to manage a multi-tenant pre-warmed pool [3]. Currently, our C++ gateway seamlessly routes HTTP requests to pre-forked Python workers via Unix Domain Sockets. Initial local load testing confirms that our non-blocking architecture can successfully absorb traffic spikes (e.g., 60 RPS) and dynamically scale the worker pool, drastically reducing the tail latency of simulated heavy AI/CV payloads.

3 Addressing Proposal Feedback

To ensure our methodology is sound, we have integrated the professor's valuable feedback into our design:

Why not Firecracker [1]/Fast Containers [4]? We agree MicroVMs (e.g., Firecracker [1]) are the industry standard for Data Plane isolation. However, our use of bare-metal OS processes is strictly a methodology choice to create a "clean-slate proxy sandbox". This allows us to decouple components and precisely measure our lightweight prediction algorithm's CPU/Memory footprint without the noise of KVM hypervisor overhead.

*Both authors contributed equally to this research.

Histogram & Scaling Logic. As detailed in Section 2, we addressed the "Histogram of what" (RPS metric), "how to detect sudden changes" (CUSUM algorithm), and "how much to pre-provision" (Little's Law) through our algorithmic pivot.

Time-Series Baseline. We gladly accepted the challenge to compare our heuristic against a heavyweight forecasting model. We are adding an ARIMA [2] baseline (in Python) to our evaluation. We hypothesize that ARIMA's high inference latency on edge nodes will validate our Pareto trade-off. (Note: Missing Figure 1 citation in the proposal has also been corrected).

4 Evaluation Workload & Methodology

To ensure our evaluation extends beyond synthetic log outputs, we designed a realistic Edge AI workload: **Image Processing**. The "cold start" penalty is not artificially mocked; it is physically manifested through heavy dependency initialization (e.g., importing OpenCV/Pillow), which typically takes ~800ms.

Our final deliverable will feature a dual-evaluation setup to provide both tangible and quantitative proof of our predictive pre-warming mechanism:

(1) **Interactive Web UI:** A frontend application where users can upload images to directly experience the stark contrast between a slow "cold start" (loading libraries on demand) and a sub-millisecond "hot start" via our pre-warmed pool.

(2) **Quantitative Benchmarks:** To rigorously evaluate our EWMA [9]/CUSUM [8] heuristic, we will deploy the gateway on bare-metal CloudLab nodes, utilizing `wrk` [5] to generate cyclic, bursty traffic (Poisson arrivals). We will compare our approach against three specific baselines:

- **Reactive Baseline:** A pure scale-on-demand policy with no prediction. This establishes the worst-case P99 latency.
- **Static Baseline:** A fixed-size, keep-alive worker pool with no dynamic scaling. This establishes the worst-case memory footprint.
- **Heavyweight Baseline (ARIMA [2]):** A traditional time-series forecasting model. By measuring its inference delay and CPU usage, we aim to prove that heavy algorithms negate their predictive benefits in resource-constrained edge environments.

By extracting P99 latency and memory footprints across these baselines, we will plot a Pareto frontier to demonstrate our system's efficiency. *For comprehensive implementation specifics, automated load-testing scripts, and up-to-date experimental progress, please refer to the README file in our GitHub repository.*

5 Checklist & Timeline

Completed Milestones:

☒ Architect & implement C++ Epoll HTTP Gateway.
☒ Develop EWMA/CUSUM predictive engine.
☒ Build OS-level Worker Pool & Python FaaS worker.
☒ Verify non-blocking end-to-end hot/cold routing.
☒ Finished simple two phase test. *(Please refer to README file for further detail)*

Remaining Tasks & Target Dates:

☐ Implement ARIMA baseline in Python. *(Late Mar)*
☐ Build Reactive/Static test baselines. *(Mid Mar)*
☐ Develop Frontend Web UI & API integration. *(Late Mar)*
☐ CloudLab migration & `wrk` [5] integration. *(Late Mar)*
☐ Profile metrics (P99 Latency, CPU, Mem). *(Apr)*
☐ Pareto Frontier analysis & final report. *(Apr)*

References

[1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX symposium on networked systems design and implementation (NSDI 20)*. 419–434.

[2] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

[3] Xiaohu Chai, Tianyu Zhou, Keyang Hu, Jianfeng Tan, Tiwei Bie, Anqi Shen, Dawei Shen, Qi Xing, Shun Song, Tongkai Yang, et al. 2025. Fork in the road: Reflections and optimizations for cold start latency in production serverless systems. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*. 199–218.

[4] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 2020. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 467–481.

[5] Will Glozer. 2012. wrk: a HTTP benchmarking tool. https://github.com/wg/wrk.

[6] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[7] John DC Little. 1961. A proof for the queuing formula: L= λ W. *Operations research* 9, 3 (1961), 383–387.

[8] Ewan S Page. 1954. Continuous inspection schemes. *Biometrika* 41, 1/2 (1954), 100–115.

[9] S. W. Roberts. 1959. Control Chart Tests Based on Geometric Moving Averages. *Technometrics* 1, 3 (1959), 239–250. doi:10.1080/00401706.1959.10489860

[10] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2022. Icebreaker: Warming serverless functions better with heterogeneity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 753–767.

[11] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX annual technical conference (USENIX ATC 20)*. 205–218.

[12] Kongyange Zhao, Zhi Zhou, Lei Jiao, Shen Cai, Fei Xu, and Xu Chen. 2023. Taming serverless cold start of cloud model inference with edge computing. *IEEE Transactions on Mobile Computing* 23, 8 (2023), 8111–8128.