Pressure-Aware Resource Management for Resident LLM Services in Operating Systems

Alex Gonzalez
New York University
Email: adg7899@nyu.edu

***Abstract*—Operating systems increasingly host large language models (LLMs) as resident services, introducing critical resource management challenges. LLM inference exhibits highly variable, bursty execution patterns that differ fundamentally from traditional workloads, making static OS-level resource controls potentially inadequate. This paper evaluates three resource management strategies for LLM workloads operating as resident services: unrestricted baseline, static cgroup v2 isolation (50% CPU quota, 4GB memory limit), and cgroup isolation with Pressure Stall Information (PSI)-gated admission control that dynamically defers requests during measured CPU pressure.**

Through controlled experiments on Ubuntu 25.10 running GPT-2 (124M parameters) under 1–8 concurrent requests, we demonstrate that static resource limits cause severe performance degradation, with p95 latencies reaching 20–40× baseline values (82–171 seconds versus 2.4–4.6 seconds). However, PSI-aware admission control prevents tail-latency explosions at high concurrency, achieving 6.8× better p95 latency than static controls alone at 8 concurrent requests (25s versus 171s). Our findings reveal that static isolation mechanisms designed for traditional workloads are fundamentally mismatched to LLM inference patterns, but pressure-aware admission shows promise for adaptive resource management in operating systems hosting resident AI services.

***Index Terms*—Operating systems, resource management, large language models, cgroups, pressure stall information, tail latency**

I. Introduction

Operating systems are evolving to host AI services directly as resident components. Modern platforms increasingly embed large language models (LLMs) for on-device features such as intelligent assistants, real-time translation, and context-aware search. This shift promises lower latency, enhanced privacy, and offline functionality, but introduces a fundamental resource management challenge: balancing system protection with acceptable LLM service performance. Unlike traditional services with predictable resource profiles, LLM inference exhibits highly variable execution patterns—token generation depends on prompt length, model size, and concurrent load—making static resource allocation potentially inadequate.

Current solutions fall into two categories: accelerator-level optimizations (quantization, KV-cache paging, GPU scheduling) that improve ML framework efficiency without addressing OS-level contention, and static OS controls (fixed cgroup quotas, priority assignments) that provide isolation but lack dynamic responsiveness. No prior work has empirically characterized how OS-native resource controls impact LLM service performance under concurrent load.

We evaluate three configurations across 1–8 concurrent requests: unrestricted baseline, static cgroup v2 isolation (50% CPU, 4GB memory), and cgroup isolation with PSI-gated admission control. Our results demonstrate that static limits cause severe degradation (20–40× latency increases), but PSI-aware admission prevents tail-latency explosions, achieving 6.8× better p95 latency at high concurrency.

II. Related Work

Current LLM optimization research focuses on memory management and model compression but does not address OS-level resource contention when LLMs operate as resident services.

Sheng et al. [1] introduced FlexGen for GPU/CPU/disk memory offloading, and Kwon et al. [2] proposed PagedAttention with OS-style paging, improving throughput by 2-4×. Lin et al. [3] developed AWQ quantization, reducing memory by 4× with 3.2-3.3× speedup. These approaches optimize memory efficiency but target cloud infrastructure and do not manage resource isolation when LLMs compete with foreground applications.

Crankshaw et al. [4] introduced Clipper for low-latency prediction serving with adaptive batching, and Bhardwaj et al. [5] proposed InferLine for resource-aware ML pipeline scheduling. While these demonstrate workload-aware management, they focus on distributed serving rather than OS-level resource controls.

Unlike these approaches, we focus on OS-native mechanisms—cgroups v2 and Pressure Stall Information—to characterize resource isolation impacts. We demonstrate that static controls cause severe degradation (20-40× latency increases), while pressure-aware admission prevents tail-latency explosions under high concurrency.

III. Methodology

A. Experimental Environment

We conducted experiments on an Ubuntu 25.10 virtual machine with 4 CPU cores and 8GB RAM, hosted on macOS using UTM virtualization. The test environment ran GPT-2 (124M parameters) as a resident LLM service. (1) **Baseline:** unrestricted resource access; (2) **cgroups-only:** static limits via cgroups v2 (50% CPU quota, 4GB memory ceiling); (3) **cgroups+PSI:** identical limits plus PSI-gated admission that deferred requests when CPU pressure (avg10

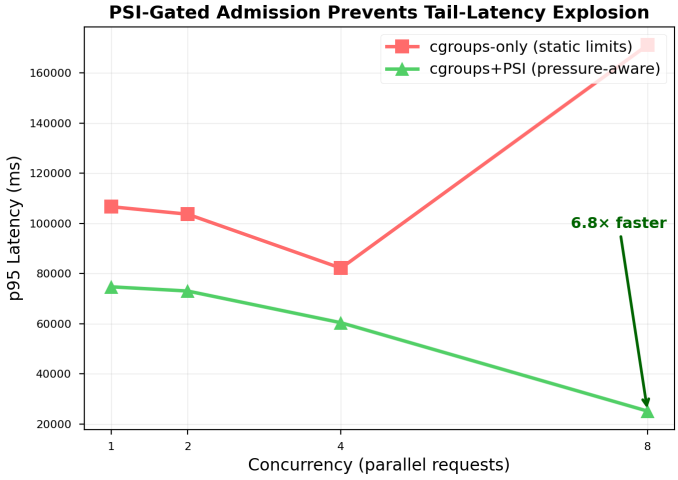

Fig. 1. p95 latency: cgroups-only vs. cgroups+PSI. PSI achieves 6.8× improvement at c=8.

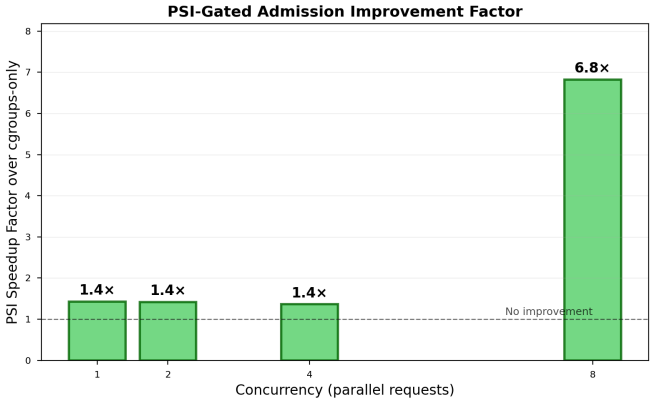

Fig. 2. PSI speedup factor over cgroups-only. Improvement increases from 1.4× to 6.8× with concurrency.

from /proc/pressure/cpu) exceeded 95.0, checking at 500ms intervals with 10-attempt maximum.

For each configuration and concurrency level (1, 2, 4, 8), we submitted 20 inference requests using prompts of varied prompts (27-41 characters). Concurrency was implemented using Python's ThreadPoolExecutor with 100ms request spacing. End-to-end latency was measured from submission to completion, yielding distributions for p50, p95, and p99 analysis. Each configuration was run once, generating sufficient data to characterize tail-latency behavior under resource constraints.

IV. Results

Static cgroup limits caused severe degradation, with p95 latencies of 82–171 seconds versus the baseline's 2.4–4.6 seconds (Figure 1). PSI-gated admission achieved 1.4× improvement at low concurrency (c=1,2,4) and 6.8× at high concurrency (25s vs. 171s at c=8, Figure 2). Table I shows complete distributions.

Critically, cgroups-only performance degraded with increasing load while cgroups+PSI counterintuitively improved, revealing that static limits become increasingly problematic as contention rises while dynamic admission becomes more effective precisely when needed most.

The p99 results mirror p95 trends (cgroups-only: 190s vs. cgroups+PSI: 28s at c=8), confirming PSI benefits extend to worst-case scenarios.

TABLE I
p95 Latency (ms) by Condition and Concurrency

Concurrency	Baseline	cgroups-only	cgroups+PSI
1	4605.4	106653.4	74698.0
2	3570.3	103659.2	73011.4
4	2397.6	82132.6	60363.5
8	3199.0	171138.2	25097.7

V. Conclusion

We demonstrated that static cgroup isolation causes severe LLM service degradation (20-40× latency increases), but PSI-gated admission prevents tail-latency explosions, achieving 6.8× better p95 latency at high concurrency. PSI's advantage increased from 1.4× at low loads to 6.8× at high loads, showing that dynamic feedback mechanisms can mitigate performance collapse under resource constraints.

Future work should explore optimal resource allocations, adaptive thresholds, and production-scale evaluation. Our work provides empirical evidence that resident AI services require adaptive, pressure-aware resource management rather than static isolation mechanisms.

References

[1] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, D. Y. Fu, Z. Xie, B. Chen, C. Barrett, J. E. Gonzalez, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: High-throughput generative inference of large language models with a single GPU," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 202. Honolulu, HI, USA: PMLR, Jul. 2023, pp. 31 094–31 116. [Online]. Available: https://proceedings.mlr.press/v202/sheng23a.html

[2] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23)*, Koblenz, Germany, Oct. 2023.

[3] J. Lin, J. Tang, H. Tang, S. Yang, W. Chen, W. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for on-device LLM compression and acceleration," in *Proceedings of Machine Learning and Systems 6 (MLSys 2024)*, 2024. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html

[4] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA, 2017, pp. 613–627.

[5] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. B. Gibbons, and B. Arzani, "Inferline: Latency-aware provisioning and scaling for prediction serving pipelines," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21, Virtual Event, Germany, 2021, pp. 477–491.