pMVX: Policy-Level Multi-Version Execution for Agentic OS Kernel Self-Tuning

Sujot Singh
University of Illinois Chicago
Chicago, Illinois, USA
ssing231@uic.edu

Eddie Federmeyer
University of Illinois Chicago
Chicago, Illinois, USA
efeder2@uic.edu

Kenan Alghythee
University of Illinois Chicago
Chicago, Illinois, USA
kalghy2@uic.edu

Xiaoguang Wang
University of Illinois Chicago
Chicago, Illinois, USA
xgwang9@uic.edu

ABSTRACT

Operating system performance depends critically on kernel policy choices, yet these policies are typically selected statically despite dynamic and heterogeneous workloads. While an agent managing an operating system can observe the outcome of the policy it applies, it fundamentally lacks access to counterfactual feedback - how alternative kernel policies would have behaved under the same workload without risking system stability.

We present pMVX, a framework for adaptive kernel policy selection based on policy-level multi-version execution. pMVX evaluates multiple kernel policy variants concurrently across isolated OS instances executing identical workloads, enabling empirical comparison without impacting production systems. The resulting observations are distilled into a lightweight mapping from eBPF-derived workload characteristics to effective policies, which guides safe runtime policy selection. We instantiate pMVX with a self-tuning Linux scheduler built on top of sched_ext, and show that it improves performance over static schedulers across a range of scheduling-sensitive workloads.

KEYWORDS

Self-Tuning, Scheduler, Multi-Version Execution

ACM Reference Format:
Sujot Singh, Eddie Federmeyer, Kenan Alghythee, and Xiaoguang Wang. . pMVX: Policy-Level Multi-Version Execution for Agentic OS Kernel Self-Tuning. In *Proceedings of ACM Conference (AgenticOS'26).* ACM, New York, NY, USA, 3 pages.

1 INTRODUCTION

Autonomous agents are rapidly becoming operators of real computing systems, from developer workflows to DevOps pipelines and production orchestration. Unlike traditional automation, which relies on rigid, predefined rules, these agents exhibit a higher degree of autonomy: they plan actions, observe outcomes, and interactively refine their behavior based on feedback from the system.

However, when agents are asked to reason about and control OS-level policies and mechanisms, such as scheduling, parameter selection, isolation, or resource management, they encounter a fundamental limitation. Production systems are ill-suited to the trial-and-error learning paradigm that underlies most agentic approaches. Sequential experimentation cannot reliably capture transient system dynamics, and unsafe decisions may directly violate performance or availability guarantees. As a result, agents are placed in a feedback vacuum: they are expected to learn from interaction, yet the OS provides no safe way to observe how *alternative kernel policies* would have behaved under the same workload.

This gap creates a paradox for agent-managed systems. Without access to counterfactual feedback, namely answers to the question "what would have happened had a different policy been applied?", an agent's ability to optimize OS behavior is fundamentally constrained. Existing OS abstractions expose metrics and controls, but they do not enable principled comparison among competing policies under identical execution contexts.

In this paper, we argue that counterfactual reasoning should be elevated to a first-class service in an agentic operating system. We define counterfactual reasoning at the OS level as the ability to evaluate the outcomes that would have resulted had the kernel selected a different policy, given the same workload and execution context. To realize this capability, we introduce Policy-Level Multi-Version Execution (pMVX), which repurposes the classic idea of multi-version execution beyond its traditional use in security and fault tolerance [2, 3, 6, 7, 9, 12–14]. Unlike conventional MVX, which runs diversified replicas to detect intrusions or tolerate faults, pMVX executes multiple kernel-policy variants concurrently in isolated environments under synchronized workload drivers to enable empirical comparison. By comparing their outcomes, pMVX produces high-fidelity counterfactual feedback for alternative policies without exposing the primary production system to experimental risk. These observations are summarized in a lightweight policy memory that agents can query at runtime, enabling informed, data-driven decisions without unsafe online exploration.

To demonstrate this idea, we implement a prototype of pMVX focused on autonomously managing Linux scheduling decisions. Rather than relying on fixed heuristics or sequential trial-and-error in production, pMVX learns from counterfactual outcomes generated by parallel policy evaluation. It explores a set of Linux eBPF-based schedulers (sched_ext [11]) by executing isolated OS instances with different schedulers concurrently under identical workloads, enabling direct comparison of scheduling behaviors.

These outcomes are aggregated into a lightweight policy memory that maps workload characteristics to effective scheduling policies. During live operation, pMVX collects kernel events via eBPF and

encodes them into normalized feature vectors, mitigating scale differences across machines and workloads. These vectors are used to query a policy vector database to identify the most suitable scheduling policy observed under similar system conditions. pMVX then safely actuates the selected policy through the constrained sched_ext interface. Our early results show that counterfactual, agent-driven policy selection consistently outperforms static schedulers (e.g., the default EEVDF scheduler [10]), demonstrating that pMVX is a practical substrate for agent-managed OS control.

2 SYSTEM OVERVIEW

pMVX is designed to automatically discover, evaluate, and select kernel policies based on empirical evidence gathered from controlled experiments and live workloads. The key challenge it addresses is enabling safe, data-driven policy selection in environments where *sequential experimentation is infeasible*. To this end, pMVX adopts a two-phase architecture that cleanly separates *policy evaluation* from *policy deployment*, as shown in Figure 1.

During the self-tuning phase, pMVX employs policy-level multi-version execution to evaluate alternative kernel policies under identical workloads. Although we refer to this phase as self-tuning, pMVX is not restricted to offline training or requiring synthetic workloads. Instead, it can be deployed as part of a live cluster running real or representative workloads. The empirical evidence gathered through this process is continuously distilled and made available to the production OS, which applies these insights during live operation by monitoring workload behavior and dynamically selecting the most appropriate policy. This design decouples policy exploration from policy serving without requiring a fixed, pre-trained model, enabling counterfactual feedback to be obtained safely even as *workloads evolve*.

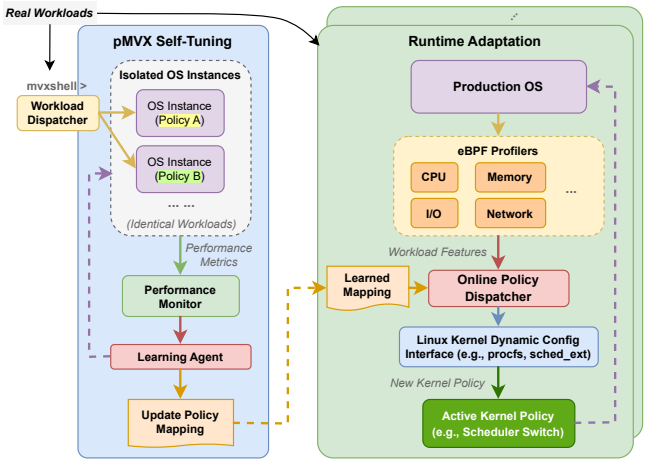

Figure 1: pMVX Architecture Overview.

Each OS instance is provisioned with identical (virtual) hardware and a distinct policy configuration. A lightweight controller synchronizes workload execution and aggregates performance results across repeated trials. Workload characteristics are profiled using eBPF events and performance counters. These signals are encoded into normalized workload vectors that preserve semantic similarity, allowing cosine similarity to reflect stable workload characteristics even when absolute scales differ. A learning agent aggregates workload vectors, execution outcomes, and policy choices to construct a compact policy mapping using a lightweight vector database. This mapping summarizes accumulated evidence and serves as the interface between policy evaluation and runtime policy selection.

During the runtime adaptation phase, pMVX continuously monitors the production OS using eBPF-based instrumentation. Kernel events are aggregated into a normalized feature vector that represents the current workload regime. This vector is used to query the learned policy mapping, identifying the policy that performed best under similar conditions during the runtime adaptation phase. When a policy change is warranted, pMVX activates the selected policy through the constrained interface (e.g., sched_ext [11], cache_ext [16]). To ensure stability, runtime adaptation incorporates basic safeguards, such as minimum dwell times. These mechanisms allow pMVX to adapt quickly to workload changes while keeping policy updates safe, reversible, and isolated from failure.

Case Study: auto_ext. We implement auto_ext as a concrete instantiation of pMVX for autonomously managing Linux scheduling decisions. auto_ext integrates policy-level multi-version execution with runtime workload sensing and policy actuation via sched_ext, forming a closed-loop, agent-managed control system.

To characterize workload behavior with low overhead, auto_ext employs eBPF-based profilers that monitor CPU, memory, I/O, and network activity. Rather than relying on coarse system statistics, these profilers attach to a small number of kernel tracepoints that expose subsystem-level pressure signals. Observed events are aggregated into normalized workload vectors, enabling consistent workload characterization across execution phases and mitigating scale differences across machines.

During the self-tuning phase, identical workloads are replicated across isolated OS instances running different sched_ext schedulers. Performance and energy outcomes from these executions are combined with the corresponding workload vectors to construct a compact policy mapping using a lightweight vector database. This mapping captures empirical relationships between workload characteristics, observed outcomes, and effective scheduling policies.

During runtime adaptation, auto_ext continuously monitors kernel events, constructs workload vectors, and queries the learned policy mapping to select an appropriate scheduler. Scheduler activation is performed through the constrained sched_ext interface, allowing scheduling logic to be switched safely and efficiently without kernel recompilation or system restarts. Our preliminary evaluation shows that auto_ext delivers 20-30% performance improvements on scheduling-sensitive workloads, such as IPC-intensive, pipeline-driven, and network-facing applications, while matching native Linux behavior on workloads where scheduling is not the dominant bottleneck.

3 DISCUSSION AND CONCLUSION

Prior systems explore dynamic or workload-aware scheduling, including user-space scheduling frameworks and energy-aware kernel schedulers [1, 4, 5, 8, 11], but largely focus on designing individual schedulers or policies. The growing ecosystem of sched_ext

schedulers demonstrates the benefits of specialization, yet policy selection remains manual and static. Recent agentic approaches such as SchedCP optimize scheduling via LLM-driven control planes [15]; in contrast, pMVX focuses on enabling counterfactual policy evaluation as a native OS capability that complements, rather than replaces, higher-level agent reasoning.

This work highlights a fundamental limitation of agent-managed OSes: while an agent can observe the outcome of the policy it applies, it lacks access to counterfactual feedback about how alternative kernel policies would have behaved under the same workload. pMVX addresses this gap by repurposing policy-level multi-version execution as an OS substrate for counterfactual reasoning. By decoupling policy evaluation from policy deployment, pMVX enables agents to select kernel policies based on empirical evidence, without unsafe online trial-and-error or reliance on pre-trained models.

Although we instantiate pMVX using Linux scheduling, the architecture is intentionally general and applies to any kernel subsystem exposing multiple policy variants and observable runtime signals, such as memory reclamation, I/O scheduling, and NUMA placement. We view pMVX as a primitive for agentic operating systems that enables OS-level *what-if* reasoning. Open challenges include scaling pMVX in cluster environments, balancing evaluation fidelity with resource cost, and defining robust safety and rollback mechanisms as agents assume greater control over system behavior.

REFERENCES

[1] Adam Beckett. 2024. 2025 will be the year of 'write your own CPU scheduler'? https://steamcommunity.com/discussions/forum/11/4625855365895983085/. Accessed: May 9, 2025.

[2] Benjamin Cox, David Evans, Adrian Filipi, Jonathan Rowanhill, Wei Hu, Jack Davidson, John Knight, Anh Nguyen-Tuong, and Jason Hiser. 2006. N-Variant Systems: A Secretless Framework for Security through Diversity.. In *USENIX Security Symposium*, Vol. 114. 114.

[3] Petr Hosek and Cristian Cadar. 2015. Varan the unbelievable: An efficient n-version execution framework. *ACM SIGARCH Computer Architecture News* 43, 1 (2015), 339–353.

[4] Jack Tigar Humphries, Neel Natu, Ashwin Chaugule, Ofir Weisse, Barret Rhoden, Josh Don, Luigi Rizzo, Oleg Rombakh, Paul Turner, and Christos Kozyrakis. 2021. ghost: Fast & flexible user-space delegation of linux scheduling. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 588–604.

[5] Julia Lawall, Himadri Chhaya-Shailesh, Jean-Pierre Lozi, Baptiste Lepers, Willy Zwaenepoel, and Gilles Muller. 2022. Os scheduling with nest: Keeping tasks close together on warm cores. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 368–383.

[6] Sebastian Österlund, Koen Koning, Pierre Olivier, Antonio Barbalace, Herbert Bos, and Cristiano Giuffrida. 2019. kMVX: Detecting kernel information leaks with multi-variant execution. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 559–572.

[7] Luís Pina, Anastasios Andronidis, Michael Hicks, and Cristian Cadar. 2019. Mvedsua: Higher availability dynamic software updates via multi-version execution. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 573–585.

[8] Feitong Qiao, Yiming Fang, and Asaf Cidon. 2024. Energy-Aware process scheduling in Linux. *ACM SIGENERGY Energy Informatics Review* 4, 5 (2024), 91–97.

[9] Babak Salamat, Todd Jackson, Andreas Gal, and Michael Franz. 2009. Orchestra: intrusion detection using parallel execution and monitoring of program variants in user-space. In *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 33–46.

[10] The kernel development community. 2024. EEVDF Scheduler. https://docs.kernel.org/scheduler/sched-eevdf.html. Accessed: Nov 30, 2025.

[11] The kernel development community. 2024. Linux Kernel Documentation: sched_ext (SCX). https://docs.kernel.org. Accessed: Nov 9, 2025.

[12] Stijn Volckaert, Bart Coppens, Alexios Voulimeneas, Andrei Homescu, Per Larsen, Bjorn De Sutter, and Michael Franz. 2016. Secure and efficient application monitoring and replication. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. 167–179.

[13] Xiaoguang Wang, SengMing Yeoh, Robert Lyerly, Pierre Olivier, Sang-Hoon Kim, and Binoy Ravindran. 2020. A framework for software diversification with {ISA} heterogeneity. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. 427–442.

[14] SengMing Yeoh, Xiaoguang Wang, Jae-Won Jang, and Binoy Ravindran. 2024. smvx: Multi-variant execution on selected code paths. In *Proceedings of the 25th International Middleware Conference*. 62–73.

[15] Yusheng Zheng, Yanpeng Hu, Wei Zhang, and Andi Quinn. 2025. Towards Agentic OS: An LLM Agent Framework for Linux Schedulers. *arXiv preprint arXiv:2509.01245* (2025).

[16] Tal Zussman, Ioannis Zarkadas, Jeremy Carin, Andrew Cheng, Hubertus Franke, Jonas Pfefferle, and Asaf Cidon. 2025. cache_ext: Customizing the Page Cache with eBPF. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*. 462–478.