SCX Ba-Bawm
Scheduler Context Extension & Burst and BPF-Aware Monitoring

Eddie Federmeyer
University of Illinois Chicago
eddie@federmeyer.com

Sujot Singh
University of Illinois Chicago
ssing231@uic.edu

Mahmoud Zayad
University of Illinois Chicago
mzayad2@uic.edu

Abstract

Recently, there has been a significant effort to add and maintain sched_ext (SCX) support in newer versions of the Linux kernel (>= 6.12). Thanks to its ease of implementation and the flexibility of the eBPF environment, sched_ext has enabled the development of many specialized and domain-specific schedulers. This is particularly powerful, as these schedulers can potentially offer significantly better performance than general-purpose ones like the Completely Fair Scheduler (CFS). However, in dynamic environments with changing workloads—such as personal computers—the burden of selecting and dispatching the appropriate scheduler often falls on the user. In this paper, we propose a method for automating workload profiling using eBPF. We also present an implementation of our solution, called *SCX Ba-Bawm*, as a portable, system-agnostic package.

1. Introduction

The CPU scheduler of a system is known to be one of the key factors dictating the performance of modern operating systems under any given workload. As a result, general purpose schedulers like linux's default Completely Fair Scheduler (CFS) are designed to provide a robust and steady baseline performance across a wide range of workloads. However, workloads vary dramatically in behavior, ranging from I/O-bound background tasks to latency-sensitive user interactions or compute-heavy batch jobs and, as a result, workload-specific schedulers often provide significant performance benefits compared to the general purpose schedulers such as the CFS when running their specialized workloads.

The recent introduction of the *sched_ext* (SCX) framework in Linux kernel versions 6.12 and above has made it easier to develop and deploy such specialized schedulers. SCX allows developers to write custom scheduling policies in eBPF, thus further enabling rapid prototyping and deployment without the need for kernel patches or restarts. This has spurred the creation of a variety of schedulers optimized for specific domains such as gaming, virtual machines, and high-performance computing. Not only does the *sched_ext/scx* project provide a variety of specialized schedulers, developers are creating more and more schedulers heavily optimized for their unique workloads every day.

Despite this boom within the field of kernel schedulers, a major limitation remains, i.e., the scheduler selection is static and user-driven. While a scheduler may be extremely performant and optimized for a specific workload, in dynamic environments, such as personal computing systems or mixed-use servers, where workloads change frequently and unpredictably, a general purpose provides much more consistent performance across a wider spectrum of workloads. Additionally, having to manually choose and switch schedulers is neither practical nor efficient, and as a result the users either never tend to change the default scheduler of a system, or stick with a single scheduler once they have changed the default despite it being less performant for general workloads. This creates a mismatch between the dynamic nature of workloads and the static nature of scheduler configuration.

To address this challenge, we propose *SCX Ba-Bawm*, a system that automates both workload classification and scheduler dispatch. Our system continuously monitors the system using low-overhead eBPF-based profilers to classify current workload characteristics, such as CPU intensity, I/O activity, memory pressure, and network activity. Our system then dynamically switches to the most appropriate scheduler within its configuration for the identified workload. This approach aims to deliver optimal scheduling performance in real time, without requiring user intervention, thus bringing dynamicism to the currently static realm of CPU schedulers.

SCX Ba-bawm is not a CPU scheduler, and is rather intended to be a dynamic CPU scheduler dispatcher which changes the system's current scheduler to the most appropriate one for a given workload. In this report, we describe the design, implementation, and evaluation of SCX Ba-Bawm.

2. Background

CPU schedulers are critical components of modern operating systems which are responsible for determining which tasks are executed on which processors and when. The default scheduler in Linux, i.e., the Completely Fair Scheduler (CFS), is designed to provide balanced performance across a plethora of workloads. As a tradeoff for a good baseline performance across various workloads, CFS and similar general-purpose schedulers inherently involve design decisions that prevent them from being optimal in all situations. For example, CFS emphasizes fairness over latency, which may not suit latency-sensitive or throughput-critical workloads [1].

To support workload-specific optimization, the Linux kernel introduced the sched_ext (SCX) framework in version 6.12 [2] that allows developers to write custom scheduling policies using the extended Berkeley Packet Filter (eBPF) which is a technology that enables safe and fast in-kernel code execution without the need to modify or recompile the kernel [3].

eBPF has become a powerful tool for dynamic system profiling, enabling real-time visibility into the system internals with minimal overhead [4]. The emergence of tools like *bcc* and *libbpf* have also allowed developers to attach eBPF probes to kernel tracepoints and measure workload patterns in fine detail. These capabilities provide a natural foundation for automated workload classification, a key component of our adaptive scheduler system.

3. Related Works

Several recent efforts have explored improving or extending Linux's scheduling behavior to support more flexible, adaptive, or energy-aware systems. These works have inspired our approach of dynamically switching between schedulers based on live workload profiling instead of focusing on building a workload-aware scheduler.

In ghOSt (Google-hosted Scheduler) [10], Humphries et al., present a framework for delegating Linux scheduling decisions to user space while retaining performance close to in-kernel solutions. This enables rapid prototyping and experimentation by decoupling scheduling policy from the kernel and supporting fast feedback loops. While ghOSt enables dynamic and domain-specific scheduling similar to our goals, it requires a dedicated user-space agent and customized infrastructure. In contrast, our system uses the in-kernel (>=linux 6.12) *sched_ext* (SCX) interface and leverages eBPF for both profiling and control, resulting in a portable and self-contained design that requires no persistent user-space daemons.

Qiao et al. introduce an energy-aware scheduler for Linux [11] that improves energy efficiency by adapting task placement and CPU frequency based on workload characteristics. Their scheduler is implemented as a modification to the existing Linux scheduling framework, focusing primarily on mobile and embedded systems. This work highlights the benefits of dynamically adjusting scheduling policies to match workload demands. While this work closely aligns with our system, their energy focus and static kernel patching differ from our more general-purpose and runtime-adaptive approach, which can be used in diverse environments without kernel recompilation.

Albeit unconventional, our work is also inspired by community discussions such as a Steam forum thread [6] that underscores the growing user interest in workload-specific schedulers. While [6] is particularly in the gaming and desktop context, the users report experimenting with *sched_ext*-based alternatives like *scx_rusty* or *scx_niceness* to improve latency, responsiveness, and performance in real world workloads on Linux. To us, not only does this signal an increasing awareness and end-user interest in CPU scheduling policies on their systems, but also a clear abundance of already available and significantly optimized scheduling solutions for specialized workloads. However, these efforts currently rely on manual trial and error, with no automated mechanism for adapting to changing workloads. Our system aims to close this gap by profiling the system in real-time and automatically dispatching the scheduler best suited for the current workload from a variety of available options.

4. Methodology

Our work builds on the foundation of [8], [9], [10], and [11] by integrating scheduler switching with real-time workload classification using modern Linux capabilities.

4.1 Scheduler Choice

The selection of schedulers for the dispatcher relied on analyzing the available *sched_ext* implementations and their use cases for various workloads. The schedulers used for the dispatcher were scx_simple, scx_nest, scx_prev. The default scheduler was *scx_simple* is used for idle, CPU, and Memory intensive loads because of its reasonable performance on varying moderate workloads like a general purpose scheduler. *Scx_nest* was selected for high network load because it optimizes workloads when

CPU utilization is low and also for its minimal latency design. For *IO* intensive *scx_prev* was chosen because of its performance on simple topology combined with high preference given to IO tasks.

Other schedulers we considered for the dispatcher, these can be loaded by simply having their binaries built within your system and replacing their names within the *dispatcher.py* file.

- *scx_rusty*
  - Considered for the default Scheduler
  - Suitable for general moderate workloads
- scx_lavd
  - Considered for Network Intensive load
  - Explicit focus on latency.
- *scx_bpfland*
  - Considered for High IO load/High Memory load
  - Prioritizes tasks with frequent voluntary context switches.
- *scx_flatcg*
  - Considered for High CPU load
  - Leverages weight-based cgroup CPU control.

4.2 Building the profiler

The goal of the system workload profilers is to collectively identify the current dominant workload type the system is experiencing in real time. Using eBPF to monitor specific kernel events, our architecture defines the following four classes of profilers: CPU-profiler, IO-profiler, Memory-profiler, and Network-profiler. Each individual profiler is used to measure the workload a selected kernel event is facing that can reliably indicate the CPU, IO, Memory, or Network activity of the system.

Below, we provide an analysis of the tracepoints selected for each profiler, and also list the alternate options we had for the same:

4.2.1 CPU Profiler

- Selected tracepoint: *sched:sched_switch*
- This tracepoint fires on every context switch, which is a reliable signal of active CPU scheduling.
- Alternates considered:
  - *sched:sched_wakeup*
  - *cpu_clock()*
  - *bpf_get_smp_processor_id()*

4.2.2 IO Profiler

- Selected tracepoint: *block:block_rq_issue*
- This event covers both read and write operations to block devices (e.g., SSDs, HDDs).
- Alternates considered:
  - *block:block_rq_complete*
  - *sys_enter_read/sys_enter_write*
  - *eBPF kprobes on vfs_read, vfs_write*

4.2.3 Memory Profiler

- Selected tracepoint: *kmem:mm_page_alloc*
- Directly measures actual memory pressure, as it tracks allocations of physical pages.
- Alternates considered:
  - *mm:mm_page_alloc_zone_locked*
  - *kmem:mm_page_alloc_extfrag*
  - *sys_brk, sys_mmap, sys_munmap*
  - Kprobes on slab allocations (kmalloc)

4.2.4 Network Profiler

- Selected tracepoint: *net:net_dev_queue*
- Captures outgoing network activity, which is often the dominant signal in network-intensive tasks.
- Alternates considered:
  - *net:netif_receive_skb*
  - *tcp_sendmsg, tcp_recvmsg*
  - kprobes on *sys_sendto, sys_recvfrom*

5. Implementation

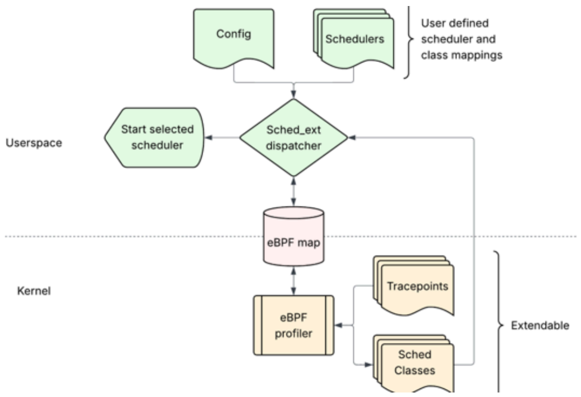

Figure 1: *Structure of SCX Ba-Bawm workflow*

We have implemented *SCX Ba-Bawm* on Ubuntu 25.04 (Plucky Puffin) running Linux Kernel version 6.14, which includes native support for sched_ext. Our implementation consists of four eBPF-based profilers and a central dispatcher, all implemented as Python scripts using the

BPF Compiler Collection (BCC). Each profiler is designed to monitor a specific aspect of system behavior: CPU usage, memory pressure, I/O intensity, and network requests. These profilers operate independently, collecting real-time metrics from kernel space and exporting with pinned eBPF maps, allowing them to be used for user-space decision-making.

To simplify deployment, we provide a startup script that launches all four profilers as background processes. Once the profilers are active, the dispatcher can be executed with administrative privileges using *sudo python dispatcher.py*. The dispatcher continuously reads metrics from the profiler output. Based on this profile, it dynamically switches to the most appropriate configured SCX scheduler. This allows seamless, low-latency adaptation to workload changes without user intervention, enabling performance optimization in rapidly changing environments such as personal computing systems or developer workstations.

The full source code for SCX Ba-Bawm is available on GitHub at https://github.com/EddieFed/scx_ba_bawm.

6. Future Work

Our SCX Ba-Bawm implementation is developed using Python and the BPF Compiler Collection (BCC), which provides us with rapid prototyping capabilities and an accessible interface for working with eBPF. However, this approach comes with limitations in terms of performance, portability, and dependency overhead. Because Python is our host environment, the resulting eBPF system may incur higher memory usage and slower startup times—factors that can be critical in production environments or resource-constrained systems.

An extension of this work would be to reimplement the profilers as standalone bpf.c programs compiled using libbpf or Clang/LLVM. This would provide a fully native eBPF bytecode, allowing the entire system to be deployed without Python, reducing runtime overhead, and improving compatibility with environments where Python's BCC is unavailable or undesirable. In doing so, SCX Ba-Bawm could also take advantage of additional kernel features and hooks through eBPF directly, potentially enabling more fine-grained and/or low-latency profiling.

7. Conclusion

As more specialized schedulers continue to be developed, the need for management of these scheduler systems becomes increasingly important in dynamic, heterogeneous workload environments. SCX Ba-Bawm works to address this by introducing a novel, automated, profiler-driven approach for selecting and dispatching schedulers. By using the power of eBPF and sched_ext, our system shows that it is possible to achieve low overhead, scheduler switching on real-time dynamic workloads.

7. References

[1] Love, R. (2010). Linux Kernel Development (3rd ed.). Addison-Wesley.

[2] Linux Kernel Documentation: sched_ext (SCX) https://docs.kernel.org

[3] Høiland-Jørgensen, T. et al. (2018). *The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel*. SIGCOMM. https://dl.acm.org/doi/pdf/10.1145/3281411.3281443

[4] Brendan Gregg. (2019). *BPF Performance Tools: Linux System and Application Observability*. Addison-Wesley. https://www.brendangregg.com/bpf-performance-tools-book.html

[5] iovisor/bcc: BCC - Tools for BPF-based Linux IO analysis, networking, monitoring, and more. GitHub. Retrieved May 9, 2025 from https://github.com/iovisor/bcc

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

[7] The Linux Foundation. 2024. More features and use-cases for sched_ext (David Vernet). Video. (9 July 2024) Retrieved May 9, 2025 from https://www.youtube.com/watch?v=skCBvHVrVhc

[8] sched-ext/scx: sched_ext schedulers and tools. GitHub. Retrieved May 9, 2025 from https://github.com/sched-ext/scx

[9] 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 *Seventeenth European Conference on Computer Systems (EuroSys '22), April 5–8, 2022, RENNES, France*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3492321.3519585

[10] Jack Tigar Humphries, Neel Natu, Ashwin Chaugule, Ofir Weisse, Barret Rhoden, Josh Don, Luigi Rizzo, Oleg Rombakh, Paul Turner, Christos Kozyrakis. ghOSt: Fast & Flexible User-Space Delegation of Linux Scheduling In *SOSP '21: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles.* 17 pages. https://doi.org/10.1145/3477132.3483542

[11] Feitong Qiao, Yiming Fang, Asaf Cidon. Energy-Aware Process Scheduling in Linux In *ACM SIGENERGY Energy Informatics Review, Volume 4, Issue 5*. 7 Pages. https://doi.org/10.1145/3727200.3727214