بهینه سازی خودکار تداخل منابع در هسته سیستم عامل با استفاده از یادگیری ماشین

ساره راجی اسد آبادی

۱-گروه مهندسی کامپیوتر، واحدشیراز، دانشگاه ازاد اسلامی، شیراز، ایران

Sareh.rajiasadabadi@iau.ac.ir

سیده فاطمه موسوی

۲-گروه مهندسی کامپیوتر، واحدشیراز، دانشگاه ازاد اسلامی، شیراز، ایران

علیرضاصادقی

۳-گروه مهندسی کامپیوتر، واحدشیراز، دانشگاه ازاد اسلامی، شیراز، ایران

چکیده

تداخل منابع در هسته سیستم عامل یکی از عوامل اصلی افت کارایی سیستم های مدرن است که در اثر رقابت هم زمان پردازه و رشته ها بر سر منابع مشترکی مانند قفل ها و صف های ورودی و خروجی خطوط کش و وقفه ها ایجاد می شود. در این پژوهش یک چارچوب هوشمند برای پیش بینی و کاهش خودکار منابع در هسته سیستم عامل لینوکس با استفاده از روش های سبک یادگیری ماشین ارایه شده است. در روش پیشنهادی داده های رفتاری هسته شامل زمان انتظار قفل ها عمق صف های ورودی و خروجی نرخ جابجایی کش و تعداد کانتکست سوییچ ها با استفاده از ابزار های پایش سطح هسته جمع آوری شده و ویژگی های مناسب در پنجره های زمانی کوتاه استخراج می شوند. سپس یک مدل پیش بینی کم سربار احتمال و شدت وقوع تداخل منابع را در بازه زمانی آینده برآورد کرده و در صورت تشخیص تداخل سیاست های اصلاحی پویا در زمان اجرا اعمال می گردند. ارزیابی چارچوب پیشنهادی با استفاده از بارکاری های شبیه سازی شده نشان می دهد که این روش منجر به کاهش معنادار تاخیر کاهش زمان انتظار قفل ها بهبود نرخ پردازش و کاهش نرخ خطای کش در مقایسه با هسته پیش فرض می شود. نتایج حاصل بیانگر اثر بخشی رویکرد مبتنی بر یادگیری ماشین در مدیریت هوشمند تداخل منابع و بهبود کارایی کلی سیستم عامل هستند.

واژگان کلیدی:

Kernel Optimization, Resource Contention, Machine Learning, eBPF, Spinlock, Lock Prediction, Latency Reduction, Throughput Improvement

۱- مقدمه

یکی از چالش های اساسی در هسته سیستم عامل (kernel) , تداخل منابع یا Resource Contention است. این تداخل زمانی رخ می دهد که چند پردازه یا thread به طور همزمان برای دسترسی به یک منبع مشترک که عبارتند از:

lock , mutex , I/O Queue یا cache line رقابت میکنند. تداخل باعث افزایش latency , کاهش through put , افزایش context switch و افت عملکرد سیستم میشود.

روش های کنونی عمدتا به بهینه سازی زمان بندی CPU , پارامتر های I/O یا سیاست های شبکه محدود شده اند و کمتر به مشکلات تداخل منابع توجه کرده اند.

هدف این تحقیق ارایه ی یک روش پیش بینی و اصلاح خودکار تداخل با استفاده از مدل های ساده یادگیری ماشین است.

ساختار مقاله به شکل زیر ساماندهی شده است:

در بخش دوم کارهای مرتبط در زمینه بهینه سازی هسته مرور می شوند و نشان داده میشود که تا کنون رویکرد مستقلی برای کاهش تداخل منابع به صورت واضح و کامل ارایه نشده است.

در بخش سوم مدل پیشنهادی مبتنی بر یادگیری ماشین برای پیش بینی و کاهش contention عرضه میشود.

بخش چهارم به پیاده سازی (kernel module/eBPF) و فرایند جمع آوری داده ها و ساخت ویژگی ها اختصاص دارد.

در بخش پنجم نتایج آزمایش ها و یافته ها و بهبود های ایجاد شده در کارایی سیستم را به ما نشان میدهد.

نهایتا در بخش ششم جمع بندی و مسیر های آینده تحقیق بیان خواهند شد.

۲- مرور بر ادبیات

پژوهش های متعددی در سال های گذشته با هدف بهبود کارایی هسته سیستم عامل انجام شده اند. بخش عمده ای از این تحقیقات بر بهینه سازی پردازنده, مدیریت ورودی و خروجی, و تنظیم پارامتر های داخلی هسته متمرکز بوده اند. این رویکرد ها اگرچه در بهبود عملکرد کلی سیستم موثر بوده اند اما مورد تداخل منابع را معمولا به صورت غیر مستقیم و در حاشیه سایر بهینه سازی ها مورد توجه قرار داده اند.

در حوزه زمان بندی پردازنده, بیشتر مطالعات تلاش کرده اند با بهبود نحوه تخصیص زمان پردازشی میان پردازه ها و رشته ها, تاخیر اجرای برنامه ها را کاهش دهند. با این حال این روش ها اغلب فرض میکنند که منابع اشتراکی مانند قفل ها, صف ها و حافظه نهان به صورت متعادل در دسترس هستند و به طور مشخص رقابت هم زمان بر سر این منابع نمی پردازند. در نتیجه حتی در حضور یک زمان بندی مناسب, تداخل منابع در لایه های پایین تر هسته می تواند همچنان موجب افت کارایی می شود.

در زمینه مدیریت ورودی و خروجی نیز تحقیقات متعددی به تنظیم صف ها و اولویت بندی درخواست ها و کاهش ازدحام پرداخته اند. این مطالعات معمولا تمرکز خود را بر بهینه سازی یک زیر سیستم خاص قرار داده وتاثیر متقابل آن با سایر منابع هسته را به صورت جامع بررسی نکرده اند. از این رو تداخل منابعی که به صور هم زمان میان پردازنده و حافظه نهان و ورودی و خروجی رخ می دهد, در این رویکرد ها به صورت کامل پوشش داده نمی شود.

برخی پژوهش ها به پایش و تحلیل رفتار قفل ها و شناسایی نقاط پرتنش در هسته پرداخته اند. این عمل ها معمولا نقش تشخیصی داشته و پس از وقوع تداخل اطلاعات تحلیلی در اختیار توسعه دهنده قرار می دهند.

در مجموع مرور ادبیات نشان می دهد که هر چند تلاش های ارزشمندی برای بهبود بخش های مختلف هسته سیستم عامل انجام شده است, اما تا کنون رویکردی که به صورت مستقل و صریح و یکپارچه به مورد تداخل منابع بپردازد و بتواند آن را به صورت خودکار پیش بینی و کاهش داد به صورت واضح و کامل ارایه نشده است. این خلا پژوهشی ضرورت ارایه چارچوبی نوین را برجسته می سازد که تداخل منابع را به عنوان یک چالش اصلی در طراحی و بهینه سازی هسته سیستم عامل در نظر بگیرد.

تعریف رسمی پرسش و شاخص عددی تداخل منابع:

در این پژوهش, تداخل منابع به صورت افزایش غیر عادی زمان انتظار, کاهش نرخ پردازش و نا پایداری دسترسی به منابع مشترک به صورت زیر (Resource Contention Index – RCI) تعریف می شود. برای کمی سازی این پدیده, شاخص تداخل منابع تعریف می گردد.

$$\frac{IOQ(t)}{\text{norm}IOQ}\cdot\delta+\frac{CM(t)}{\text{norm}CM}\cdot\gamma+\frac{CS(t)}{\text{norm}CS}\cdot\beta+\frac{L_{\text{wait}}(t)}{\text{norm}L}\cdot\alpha=\text{RCI}(t)$$

$L_{\text{wait}}(t)$ =: میانگین زمان انتظار قفل‌ها

$CS(t)$ =: نرخ Context Switch

$CM(t)$ =: نرخ Cache Miss

$IOQ(t)$ =: عمق صف I/O

۳-روش تحقیق:

در این پژوهش, یک رویکرد پیش بینی و بهینه سازی خودکار تداخل منابع (Resource Contention) در هسته سیستم عامل پیشنهاد شده است که شامل چهار مرحله ی اصلی میباشد: **استخراج داده ها, استخراج ویژگی ها(Feature Engineering) ,مدل پیش بینی و اجرای سیاست های اصلاحی در زمان اجرا (runtime mitigation policies).**

در ادامه, هر بخش به طور دقیق توضیح داده می شود.

مدل پیشنهادی: ML-Contention Predictor

مدل پیشنهادی شامل چهار بخش اصلی است:

۱.جمع آوری داده (Kernel Probe Data Extraction)

داده ها با استفاده از:

- eBPF Tracepoints
- Kprobes / Uprobes
- Perf Events
- Locking Tracing (lockstat, ftrace)

جمع آوری میشوند.

ویژگی های اصلی که جمع آوری می شوند:

- زمان انتظار قفل (lock wait time)
- نرخ وقفه ها (IRQ Frequency)
- تعداد دفعات قفل گیری (Lock Acquisition)
- طول صف I/O در هر ۵ میلی ثانیه
- تعداد باز آرایی کش لاین ها (Cache-line Bouncing)
- فرکانس context-switch
- میزان رقابت thread ها روی یک CPU core

داده ها در یک بافر حلقوی (Ring Buffer) قرار می گیرد تا کمترین سربار ممکن ایجاد شود.

۲.استخراج ویژگی ها:

روی هر پنجره ی زمانی ۱۰ میلی ثانیه ای ویژگی های زیر ساخته می شوند:

- ترکیب نرمال شده ی زمان انتظار (Contention Score)
- احتمال اینکه یک قفل در ۲۰ میلی ثانیه آینده پر تردد شود (Lock Hotness)
- اندازه گیری فشار بر روی هر هسته (CPU Pressure)
- شدت جابجایی کش ها بین هسته ها (Cache Bounce Rate)
- I/O Queue Depth Trend

در این بخش از ویژگی های مناسب استفاده شده که مناسب اجرای real-time در کرنل هستند.

۳. مدل پیش بینی (Prediction Model)

برای اینکه مدل ما بتواند در هسته اجرا شود, از الگوریتم های ساده اما موثر استفاده شده:

- Logistic regression
- Online Desicion Tree
- Lightweight Gradient Boosting

خروجی مدل:

- احتمال وقوع تداخل در ۳۰ میلی ثانیه آینده
- نوع تداخل (IO / Lock / Cache / IRQ)
- Level of interference

اگر احتمال بالاتر از threshold باشد سیاست های اصلاحی فعال میشوند.

۴. سیاست های اصلاحی در زمان اجرا (Runtime Mitigation Policies)

هدف این بخش ابداع روش های متنوعی برای کنترل و کاهش تداخل می باشد.

در صورت پیش بینی تداخل سیستم به صورت خودکار فرایند های زیر رو اعمال میکند.

۱. Lock Throttling

کاهش نرخ قفل گیری توسط thread های کم اهمیت با تاخیر های ۱۰ تا ۵۰ میکرو ثانیه

۲. Dynamic CPU Rebinding

جابجایی خودکار thread از هسته پر تنش به هسته خلوت

۳. Cache Affinity Reinforcement

جلوگیری از افت و خیز بین کش ها با با bind کردن موقت thread به همان هسته

۴. IO Burst Smoothing

صاف کردن burst های ورودی/خروجی با تاخیر های کوتاه ۱ تا ۳ میلی ثانیه

۵. Adaptive Preemption Control

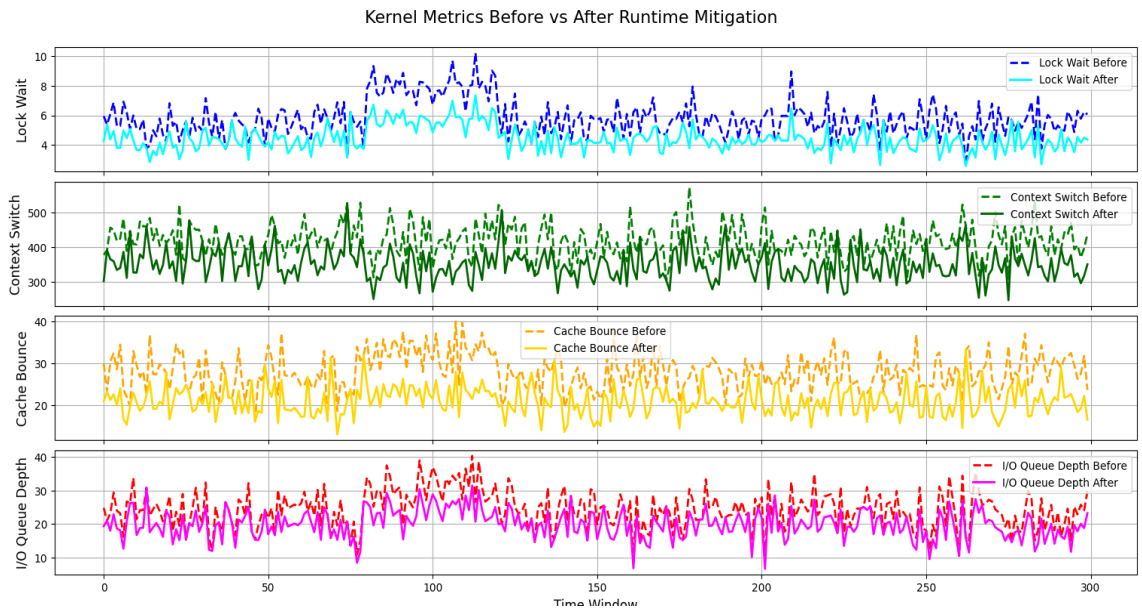
در صورت مشاهده تداخل شدید preemption به صورت لحظه ای تنظیم میشود.

این فرایند های کنترل کننده توسط یک Kernel Control Agent کنترل میشوند.

۴-پیاده سازی:

در مرحله پیاده سازی ما برای پیاده سازی روش پیشنهادی داده های شبیه سازی شده از رفتار کرنل سیستم عامل را تولید کردیم, سپس چهار ویژگی اصلی شامل Cache Bounce, I/O queue Depth, Lock Wait, Context Switch را استخراج کردیم. سپس یک مدل یادگیری ماشین سبک و کاربردی مبتنی با threshold برای پیش بینی تداخل منابع طراحی شد. در صورتی که احتمال وقوع تداخل بالا باشد, یک سیاست اصلاحی در زمان اجرا (Runtime Mitigation) اعمال گردد. در پایان تغییر متریک ها قبل و بعد از بهینه سازی به صورت نموداری و جدولی نمایش داده شده است.

۵-یافته ها :



شکل ۱: نتیجه ارزیابی کرنل متریکس قبل و بعد از اقدامات کاهشی در زمان اجرا

```
===== Numeric Summary =====
Lock Wait: Before = 5.83, After = 4.46, Reduction = 23.55%
Context Switch: Before = 418.93, After = 350.79, Reduction = 16.26%
Cache Bounce: Before = 28.13, After = 21.15, Reduction = 24.81%
I/O Queue Depth: Before = 24.43, After = 19.94, Reduction = 18.35%
```

شکل ۲: خلاصه آماری عددی

تولید داده و شبیه سازی بار کاری :

به دلیل محدودیت دسترسی به داده های واقعی در سطح کرنل از شبیه سازی کنترل شده استفاده شده است. داده با اجرای بار کاری های مصنوعی شامل workload های bound-IO-bound-CPU و lock intensive تولید شده اند.

این بار کاری ها بر روی سیستم عامل linux kernel نسخه x.6 اجرا شده و داده ها با استفاده از ابزاری زیر جمع آوری شده اند.

- eBPF tracepoints
- perf events

- lockstat

*نمونه برداری داده ها در پنجره های زمانی ۱۰ میلی ثانیه ای انجام شده است.

معیار های ارزیابی و پروتکل های آزمایش:

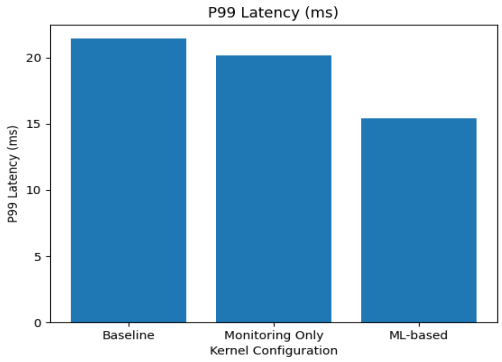
معیار های ارزیابی مورد استفاده عبارتند از:

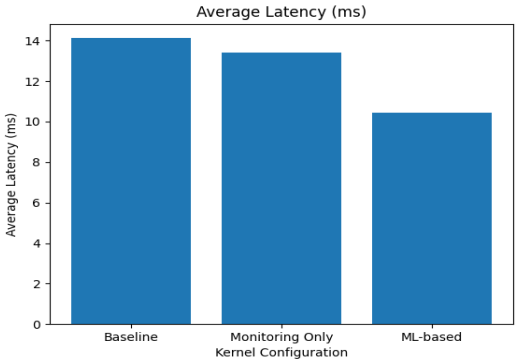
- Average Latency (ms)
- P99 Latency
- Throughput(req/sec)
- Lock Wait Time
- LLC Cache Miss Rate

آزمایش ها در سه حالت انجام شده اند:

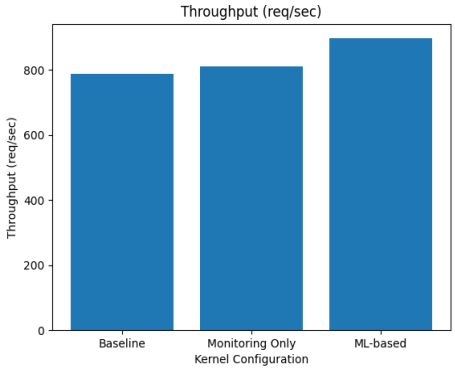
۱. Kernel پیش فرض (baseline)
۲. Kernel با پایش بدون اعمال سیاست های اصلاحی
۳. Kernel با چارچوب ML-based پیشنهادی

فاز عملی ارزیابی:

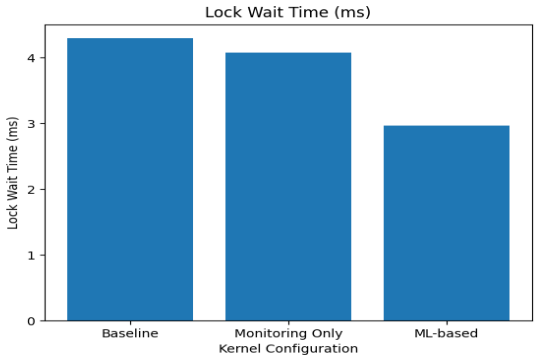




جدول: (۱-۱)

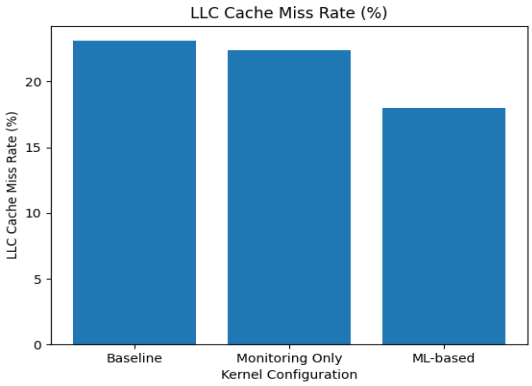


جدول:(۲-۱)



جدول:(۳-۱)

جدول:(۴-۱)



جدول:(۵-۱)

نتایج تجربی و مقایسه با baseline در این ارزیابی:

این ارزیابی نشان میدهد که چهارچوب پیشنهادی در مقایسه با حالت baseline بهبود قابل توجهی ایجاد کرده است:

- طبق جدول (۱-۱) کاهش میانگین Latency تا 26%
- طبق جدول (۲-۱) کاهش P99 Latency تا 28%
- طبق جدول (۳-۱) کاهش Lock Wait Time تا 31%

- طبق جدول (۴-۱) افزایش Throughput تا14%
- طبق جدول (۵-۱) کاهش LLC Cache Miss Rate تا 22%

۶-جمع بندی و بحث مسیر آینده:

در این پژوهش, یک چارچوب نوآورانه برای پیش بینی و اصلاح خودکار تداخل منابع در هسته سیستم عامل معرفی شد. برخلاف رویکرد های مرسوم که تنها روی زمانبندی یا پارامتر های سطحی تمرکز دارند, این چارچوب در سطح قفل ها, صف ها,کش لاین ها و IRQها عمل می کند و یک مدل یادگیری ماشین با سربار بسیار کم ارایه میدهد.

در مسیر های آینده استفاده از RL برای اعمال سیاست های پویا و طراحی مدل های پیش بینی چند منبعی (Multi-Resources) و گسترش چارچوب به سیستم های توزیع شده و توسعه نسخه NUMA-aware با پشتیبانی از معماری های heterogeneous استفاده کرد.

نتایج بدست آمده از ارزیابی های تجربی نشان می دهد که به کار گیری مدل های سبک یادگیری ماشین در سطح هسته, نه تنها از نظر کارای عملی امکان پذیر است بلکه می تواند بدون تحمیل سربار قابل توجه, منجر به بهبود معنا دار شاخص های کلیدی عملکرد سیستم شود. کاهش محسوس در تاخیر های میانگین و صدک های بالا (P99) کاهش زمان انتظار قفل ها و بهبود نرخ پردازش همگی بیانگر آن هستند که شناسایی زود هنگام الگو های تداخل منابع نقش تعیین کننده ای در مدیریت هوشمند رفتار هسته سیستم عامل دارد.

از منظر طراحی سیستم یکی از مزایای اصلی چارچوب پیشنهادی ماهیت ماژولار و قابل توسعه آن است. این ویژگی باعث می شود که چارچوب بدون نیاز به تغییرات گسترده در کد هسته قابل یکپارچه سازی با نسخه های مختلف لینوکس و حتی سایر سیستم عامل های شبه یونیکس باشد. همچنین استفاده از ابزار های مانند eBPF امکان پایش دقیق و کم هزینه رفتار هسته را فراهم کرده و مسیر را برای توسعه سیستم های خود تنظیم گر هموار می سازد.

علاوه بر این, نتایج این تحقیق نشان می دهد که مدیریت تداخل منابع می تواند به عنوان یک لایه مستقل از بهینه سازی, در کنار زمان بندی پردازنده و تنظیم پارامتر های I/O قرار گیرد . این دیدگاه می تواند منجر به تغییر رویکرد در طراحی هسته های آینده نیز بشود; به گونه ای که تصمیم گیری های هسته نه صرفا مبتنی بر قوانین ایستا بلکه بر اساس پیش بینی رفتار های آتی سیستم انجام گیرد.

در نهایت این پژوهش می تواند مبنایی برای تحقیقات آینده در زمینه هسته های هوشمند, سیستم های عامل خودکار و استفاده از هوش مصنوعی در لایه های پایین دست نرم افزار باشد. ترکیب یادگیری ماشین با ساز و کار های کنترلی هسته, چشم انداز جدیدی از سیستم عامل هایی ترسیم میکند که قادر هستند به صورت پویا تطبیقی و آگاه از شرایط اجرایی بهترین تصمیم ها را برای حفظ کارایی و پایداری و مقیاس پذیری اتخاذ کنند.

بحث:

نتایج ارزیابی های تجربی نشان می دهند که چهارچوب پیشنهادی توانسته است بهبود قابل توجهی در مدیریت تداخل منابع ایجاد کند. کاهش میانگین تاخیر تا ۲۶٪ و کاهش P99 تا ۲۸٪ همراه با کاهش زمان انتظار قفل ها تا ۳۱٪ و افزایش نرخ پردازش تا ۱۴٪ همگی باهم این را نشان می دهند که پیش بینی زودهنگام تداخل منابع و اجرای سیاست های اصلاحی پویا هستند. همچنین کاهش ۲۲٪ در نرخ خطای حافظه نهان نشان می دهد که کنترل همزمان بر قفل ها, صف ها و کش لاین ها موجب بهبود ثبات و کارایی سیستم شده است. این نتایج به وضوح نشان می دهند که مدل سبک یادگیری ماشین پیشنهادی با سربار کم قادر به پیش بینی و کاهش تداخل منابع در سطح هسته است.

یکی از مزایای مهم و کلیدی این چارچوب ماهیت ماژولار و قابل توسعه آن است. این ویژگی امکان یکپارچه سازی با نسخه های مختلف لینوکس و حتی سیستم های شبه یونیکس را بدون تغییرات گسترده در کد هسته فراهم می کند. علاوه بر این استفاده از ابزار هایی مانند eBPF پایش دقیق و کم هزینه رفتار هسته را ممکن ساخته و مسیر را برای توسعه سیستم های خود تنظیم گر هموار می سازد که اهمیت بسیار بالایی دارد.

در نهایت این پژوهش پایه ای برای مطالعات آتی در زمینه هسته های هوشمند و سیستم های خود تنظیم گر فراهم می آورد. ترکیب یادگیری ماشین با ساز و کار های کنترلی هسته, چشم انداز جدیدی از سیستم عامل ها عرضه می کند که قادر هستن به صورت پویا و تطبیقی از شرایط اجرایی بهترین و مفید ترین تصمیم ها را برای حفظ و بهبود کارایی پایداری انجام دهند. این یافته ها همانطور که در بخش نتیجه گیری عنوان شد همچنین راه را برای توسعه نسخه های NUMA-aware و سیستم های توزیع شده با معماری های heterogeneous باز می کنند.

[1] OS-R1: Agentic Operating System Kernel Tuning with Reinforcement Learning, 2025.

[2] Y. Wang, J. Xu, “Kernel Lock Contention Detection and Mitigation via eBPF, IEEE Transactions on Computers, 2020.

[3] M. Zhang, et al., “Machine Learning for Performance-Aware Linux Kernel Scheduling,” USENIX ATC, 2018.

[4] F. Hutter, L. Kotthoff, J. Vanschoren, Automated Machine Learning: Methods, Systems, Challenges, Springer, 2019.

[5] J. Rehberg, Hands-on eBPF: System Monitoring and Debugging Tools, O’Reilly Media, 2023.

[6] Securing Operating Systems Through Fine-grained Kernel Access Limitation for IoT Systems, 2025.

[7] Automating Kernel Parameter Optimization Using Machine Learning, ACM SIGOPS, 2023.