STM JOURNALS

Journal of
Operating Systems
Development & Trends

ISSN: 2454-9355
Volume 12, Issue 3, 2025
September–December
DOI (Journal): 10.37591/JoOSDT

https://journals.stmjournals.com/joosdt

Research JoOSDT

Designing Self-Optimizing Operating Systems: Information-Theoretic Approaches to Thread Scheduler Implementation

Anantham Srujana Jyothi[1,*], K.V.V. Subba Rao[2], Manas Kumar Yogi[2]

Abstract

Thread Level Scheduling (TLS) in multi-core and many-core processor environments represents a critical frontier in next-generation operating system design. As computing systems grow increasingly heterogeneous and concurrent, traditional scheduling strategies often rely on heuristics or localized resource metrics, frequently overlooking the deeper, quantifiable relationships and uncertainties inherent in complex concurrent workloads. This study explores the application of information-theoretic approaches, specifically entropy-based task allocation, mutual information-driven dependency analysis, and channel capacity-inspired throughput bounds, to advance TLS models within modern operating system kernels. By developing a rigorous framework grounded in entropy and mutual information, we systematically characterize thread dynamics, unravel hidden dependencies, and optimize resource allocation for maximal efficiency and minimal contention. The findings reveal that information-driven strategies offer promising directions to improve prediction, adaptability, and fairness in operating system schedulers, ultimately forging a more robust foundation for high-performance and real-time computing environments.

Keywords: Thread scheduling, operating systems, entropy, information theory, mutual information, OS kernel design

INTRODUCTION

The evolution of operating systems has necessitated increasingly sophisticated thread management strategies as system scale and complexity continue to expand. Thread Level Scheduling (TLS) stands at the heart of multi-core processor management within modern OS kernels, balancing concurrency, resource utilization, and execution predictability. However, conventional OS scheduling models frequently fall short in quantifying and responding to the stochasticity and interdependence typical of large-scale concurrent applications [1].

Recent years have witnessed growing interest in incorporating information-theoretic concepts into operating system research and development. Information theory, pioneered by Claude Shannon, provides a powerful mathematical framework to measure uncertainty, analyze system dependencies, and establish performance bounds. Within the context of OS-level TLS, information theory illuminates the degree of unpredictability in thread behaviors, the intricate web of dependencies

***Author for Correspondence**
Anantham Srujana Jyothi
E-mail: srujanajyothi.cse586@gmail.com

[1]Assistant Professor, Department of Computer Science and Engineering-Artificial Intelligence and Machine Learning Department, Pragati Engineering College (A), Surampalem, Andhra Pradesh, India
[2]Assistant Professor, Department of Computer Science and Engineering Department, Pragati Engineering College (A), Surampalem, Andhra Pradesh, India

Received Date: November 10, 2025
Accepted Date: November 12, 2025
Published Date: November 19, 2025

Citation: Anantham Srujana Jyothi, K.V.V. Subba Rao, Manas Kumar Yogi. Designing Self-Optimizing Operating Systems: Information-Theoretic Approaches to Thread Scheduler Implementation. Journal of Operating Systems Development & Trends. 2025; 12(3): 31–39p.

among tasks, and the theoretical limits of scheduler efficiency relative to available hardware resources.

Next-generation operating systems must address challenges beyond mere task execution: they must intelligently manage heterogeneous computing resources, adapt to dynamic workload characteristics, provide security guarantees, and maintain quality-of-service commitments across diverse application domains. Information-theoretic approaches offer a principled foundation for addressing these multifaceted requirements within unified OS architectures.

This study critically analyzes emerging literature, develops an analytical framework for information-theoretic TLS within operating system contexts, and proposes concrete avenues for future OS research and kernel development [2].

ENTROPY-BASED TASK ALLOCATION IN OPERATING SYSTEM SCHEDULERS

Understanding Entropy in OS Scheduling

Entropy, a fundamental concept from Claude Shannon's information theory, quantitatively measures uncertainty or disorder within a system. In the context of operating system thread scheduling, entropy reflects the unpredictability of thread execution patterns over time, the distribution of workloads across processing units, and the contention for shared system resources [3]. A scheduling algorithm with high entropy implies that thread execution order and core allocation are highly variable and less predictable. Conversely, lower entropy suggests more deterministic, stable scheduling patterns.

Understanding entropy provides OS kernel designers a critical lens to evaluate and adapt scheduling policies, balancing randomness, which can encourage fairness and load distribution, against determinism, which promotes predictability and optimized cache locality.

Quantifying Thread Assignment Uncertainty

In dynamically varying or heterogeneous computing environments, thread-to-core assignments within an operating system are rarely static or fully predictable. Entropy can be mathematically quantified over scheduling intervals by analyzing task allocation matrices, which record probability distributions of threads mapped to available cores. This quantification offers an actionable metric by which OS schedulers can assess the randomness of their allocation decisions.

For instance, an entropy-maximizing scheduler intentionally disperses thread assignments to avoid overloading specific cores, reducing the likelihood of bottlenecks or systemic stalls. Conversely, when workload predictability or affinity is paramount, a low-entropy schedule may be preferable, tightly coupling threads to particular cores to exploit cache locality or minimize migration overhead [4].

The key insight from an entropy-based perspective is that operating system schedulers can dynamically tune their policy objectives based on observed system metrics or workload demands, leveraging entropy as an optimization dimension within the kernel's decision-making framework.

Minimizing Bottlenecks and Stragglers in OS Kernel Execution

System inefficiencies often arise from performance bottlenecks and stragglers, threads or cores that significantly lag behind others, hindering overall application progress and degrading system responsiveness. Entropy measurements across different levels of the processor hierarchy: per-core, per-socket, or per-cluster, equip OS schedulers to detect early signs of imbalance and emerging hotspots.

By continuously monitoring entropy gradients within the kernel, scheduling algorithms can respond preemptively, reallocating threads or rebalancing workloads to diffuse contention points. In hybrid computing systems with multiple clusters or heterogeneous cores, entropy provides a unified

framework within the OS to ensure balanced resource usage both within and across processing clusters. This multi-level entropy monitoring supports dynamic OS scheduling policies that maintain balanced workloads during varying application phases, mitigating localized bottlenecks that compromise scalability and system throughput [5].

Adaptive OS Scheduling through Real-Time Entropy Analysis

One of the most promising applications of entropy in next-generation operating systems is adaptive, feedback-driven scheduling. As system workloads evolve, due to changes in concurrency levels, execution phases, or external inputs, the entropy of the scheduling state shifts correspondingly.

By continually quantifying entropy in real time, adaptive OS schedulers can fine-tune control parameters such as time slices, preemption thresholds, and thread affinities to sustain optimal system performance. For example, if entropy increases, signaling growing unpredictability and workload variability, the scheduler may widen time quanta or redistribute tasks to prevent core overload. Conversely, dropping entropy may trigger more aggressive affinity strategies to capitalize on stable workload patterns [6].

Such entropic tuning enables operating systems to function efficiently across diverse workload mixes without manual intervention or static configuration, leading to improved resource utilization, system responsiveness, and overall throughput. Entropy thus serves both as a diagnostic metric and a control mechanism within next-generation OS kernel frameworks.

MUTUAL INFORMATION FOR DEPENDENCY ANALYSIS IN OPERATING SYSTEMS

Decoding Thread Dependencies at the OS Level

Mutual information, a fundamental concept in information theory, quantitatively measures the amount of information shared between two random variables. Applied to operating system thread scheduling, these "variables" correspond to threads or tasks executing within the OS context. Mutual information represents the degree of dependency or correlation between execution patterns, data accesses, or communication behavior of different threads.

Understanding these dependencies is crucial for OS kernel designers because they directly impact scheduling effectiveness. Strongly dependent threads often share data, synchronize on common resources, or impose ordering constraints; co-scheduling such tasks can maximize data locality, minimize expensive communication across processor boundaries, and reduce synchronization overhead managed by the OS. Conversely, scheduling independent threads on separate cores improves system utilization by avoiding unnecessary contention [7].

Measuring Inter-Thread Communication in OS Environments

Modern concurrent applications, ranging from server workloads to data-intensive analytics, exhibit complex communication and synchronization patterns managed by the operating system. These include shared memory accesses, signaling via OS-provided locks or semaphores, message passing through kernel facilities, or indirect influences via shared cache or memory subsystems.

Mutual information calculations can be estimated from diverse runtime observations collected by the OS kernel, such as shared memory access traces, cache coherence traffic, time-stamped lock usage, or inter-thread messaging volumes. By computing mutual information matrices representing pairwise dependencies between threads, OS schedulers can construct empirical dependency graphs or topologies.

Threads with high mutual information values are identified as strongly coupled and become natural candidates for co-location on the same physical node, socket, or NUMA domain by the OS scheduler. This co-scheduling reduces overhead due to cross-node communication latency and cache misses. Additionally, mutual information reveals latent dependencies that may not be explicit in program

logic but manifest dynamically during runtime, enabling adaptive and data-driven OS scheduling policies [8].

Reducing Synchronization Overhead in OS Kernels

While synchronization primitives such as mutexes, barriers, and condition variables, provided by the operating system, are essential to ensure correctness in concurrent programs, their misuse or poor management can cause severe performance degradation. Waiting on OS-managed locks or entering busy-wait loops causes thread stalls that reduce overall system throughput [9].

Mutual information-based dependency analysis helps OS schedulers identify critical synchronization bottlenecks by highlighting which pairs or groups of threads strongly influence each other's execution timings. For example, threads that repeatedly contend for the same OS lock exhibit high mutual information through their serialized execution patterns.

By recognizing these hotspots, operating system schedulers can prioritize co-scheduling these threads to reduce remote memory accesses and interconnect traffic, or even explore speculative execution and optimistic concurrency approaches where feasible. Additionally, threads with moderate dependency levels may be candidates for increased asynchrony or relaxed synchronization managed by the OS, balancing consistency requirements against performance. Mutual information thus provides a quantitative foundation for nuanced synchronization management within the OS kernel, minimizing wasted computation and idle wait times [10].

Real-World OS Implementations and Evaluation

Recent advances in OS profiling tools and runtime tracing enable the collection of fine-grained execution data necessary for mutual information estimation in complex systems. State-of-the-art operating system thread schedulers integrate this analysis to move beyond static heuristics into system-wide awareness [11]. They employ global dependency graphs rather than isolated, thread-local views to optimize scheduling holistically within the OS kernel.

Empirical studies demonstrate that mutual information-centric OS scheduling policies consistently outperform simpler baseline strategies such as naive round-robin, first-come-first-served, or basic affinity schemes. This is especially pronounced in workloads with high contention or data sharing, such as database transaction processing, concurrent graph algorithms, or lock-free data structure manipulations.

By dynamically adapting thread placement based on measured dependencies, these OS approaches reduce cache misses, synchronization delays, and communication overhead, yielding improved scalability and system throughput. The integration of information-theoretic dependency metrics into operating system scheduling frameworks marks a significant advancement, emphasizing data-driven optimization rooted in rigorous mathematical characterization of concurrency relationships, transforming how OS kernels allocate resources in heterogeneous and multi-core processor systems.

CAPACITY AND THROUGHPUT BOUNDS IN OS SCHEDULER DESIGN

The Channel Capacity Analogy for Operating Systems

In information theory, channel capacity fundamentally measures the maximum rate at which data can be reliably transmitted over a communication channel. Applied metaphorically to operating system design and thread-level scheduling, the entire OS scheduler-resource ecosystem can be conceptualized as an information processing channel. In this analogy, tasks and threads represent the information to be processed, while hardware resources such as CPU cores, memory bandwidth, and interconnects form the channel infrastructure with inherent constraints [12].

Resource contention, core interconnect bandwidth, cache coherence overheads, and architectural bottlenecks limit the effective throughput of the system, analogous to noise and interference in a

communication channel. Therefore, channel capacity defines an upper bound on the number of tasks that an operating system can process reliably (i.e., completed successfully within constraints) per unit time.

This perspective emphasizes the need to measure OS scheduler efficiency not merely as raw throughput but with respect to theoretical maximums dictated by system design and workload characteristics [13].

Estimating OS Scheduler Efficiency

Using the channel capacity analogy, researchers develop capacity-inspired scheduling strategies that estimate theoretical upper bounds of system throughput for any given hardware configuration. Modeling thread execution as stochastic processes passing through noisy, capacity-limited channels allows capturing the variability in task lengths, arrival patterns, and dependency constraints within the operating system [14].

These models facilitate benchmarking of existing OS scheduling policies by comparing their achieved throughput against capacity limits. Furthermore, such modeling enables prediction and simulation of how potential scheduling optimizations, like improved task partitioning, affinity assignment, or load balancing within the OS kernel, could push system utilization closer to theoretical bounds.

By viewing OS scheduling as an optimization problem constrained by channel capacity, system designers are equipped to evaluate trade-offs rigorously, ensuring that improvements in one dimension (e.g., latency) do not degrade overall utilization or throughput beyond acceptable thresholds.

Maximizing Information Flow in Operating System Kernels

Achieving throughput near the system's capacity requires synchronization of several factors within the operating system. OS scheduling algorithms must optimize both task allocation and dependency management in tandem. Entropy and mutual information metrics can assist in delineating which tasks are independent and can therefore be scheduled more flexibly by the OS, spreading workload to avoid bottlenecks [14].

Conversely, dependent threads, those whose execution order impacts correctness or performance, must be scheduled by the OS within low-latency proximities, such as the same cache cluster or NUMA node, to minimize communication overhead. Balancing these local and global constraints is critical for OS kernels to avoid excessive synchronization delays or costly context switches, which reduce overall system throughput.

Operating system schedulers that dynamically adapt to changes in dependency graphs and workload entropy can sustain near-maximum utilization, pushing operational points closer to the channel's theoretical capacity without instability or saturation.

Resilience to Resource Variability in OS Design

Real-world systems face resource variability: CPU frequencies change dynamically due to power management policies managed by the OS, thermal limits affect performance, core failures may occur, and hardware runtime characteristics fluctuate [15]. Capacity-oriented OS TLS models explicitly incorporate these uncertainties by modeling the system as a time-varying channel with dynamically changing capacity.

By continuously estimating channel capacity and throughput bounds based on real-time system feedback within the OS kernel, scheduling policies can adjust thread placement, preemption frequencies, and load balancing strategies to maintain stable performance. This adaptive behavior

averts performance degradation during adverse conditions, enabling graceful degradation rather than catastrophic failure modes [16].

Such robustness is vital for next-generation operating systems serving large-scale, heterogeneous, or cloud environments where resource variability is the norm. Information-theoretic OS TLS therefore supports not only peak performance but also reliability and quality-of-service guarantees across diverse operational contexts.

APPLICATIONS, CHALLENGES, AND FUTURE DIRECTIONS IN OS DEVELOPMENT

Hybrid Scheduling Architectures in Next-Generation Operating Systems

Emerging computing architectures are increasingly hybrid, integrating general-purpose processors with specialized units such as AI accelerators, graphics processing units (GPUs), neural processing units, or programmable accelerators. These hybrid systems present unique challenges for operating system schedulers due to the diverse performance and resource profiles of different processing units.

Information-theoretic frameworks can provide an effective foundation for designing unified OS scheduling metrics that guide resource allocation across these heterogeneous components. By quantifying the amount of information processed or transferred during execution, these frameworks enable OS kernels to adapt dynamically.

For instance, entropy measures can evaluate uncertainty or variability in workload characteristics across different cores or accelerators, guiding OS decisions on load balancing. Mutual information can reveal dependencies between tasks assigned to different units, facilitating co-scheduling of dependent workloads by the OS to optimize data locality and reduce communication overhead. Capacity models, inspired by Shannon's channel capacity, can estimate maximum information flow achievable given hardware bandwidth and latency constraints, guiding OS allocation of resources to maximize throughput.

This approach ensures that operating systems leverage the entire system's capabilities efficiently, with scheduling policies aware of both data dependencies and information processing limits of the hardware [17]. By unifying diverse metrics under an information-theoretic umbrella, next-generation OS hybrid scheduling architectures can balance control and flexibility, leading to improved performance, energy efficiency, and system responsiveness.

Integration with Machine Learning in OS Kernel Design

The integration of information-theoretic metrics into machine learning (ML) environments opens promising avenues for adaptive, data-driven OS scheduling strategies. Reinforcement learning (RL) agents, for example, can incorporate entropy and mutual information as intrinsic rewards or cost functions within OS kernels, encouraging policies that minimize uncertainty and optimize task dependencies dynamically.

Such systems enable operating systems to learn resource allocation based on real-time feedback about workload characteristics and system state, instead of relying solely on static heuristics. For example, an OS scheduler might estimate mutual information between tasks to determine optimal co-location strategies, or evaluate the entropy of task execution paths to identify critical or risky workloads requiring priority handling.

Over time, an RL-based OS scheduler can adapt to workload changes, hardware heterogeneity, and system faults, providing robust and resilient performance. Recent research demonstrates that ML models augmented with information-theoretic measures can outperform traditional methods in complex, non-linear scheduling problems, particularly in cloud and data center operating system environments. This integration fosters operating systems capable of self-optimization, reducing manual tuning and improving efficiency in heterogeneous and dynamic computing landscapes.

Real-Time and Embedded Operating Systems

In real-time and embedded operating systems, scheduling must guarantee strict deadlines while accommodating resource constraints and unpredictable workloads. Traditional schedulability analysis in real-time OS kernels, based primarily on fixed deadlines and worst-case execution times, can be overly conservative or insufficiently informative.

In this context, information-theoretic analysis adds a new dimension by quantifying uncertainty and dependency structures among concurrent tasks within the OS. Entropy measures the unpredictability in task execution patterns, which directly impacts the likelihood of deadline misses. Mutual information can reveal inter-task dependencies that influence critical paths within task graphs managed by the real-time OS.

Applying these measures allows for more fine-grained, adaptive scheduling policies in real-time operating systems that can proactively mitigate risks associated with uncertainty and dependency violations. For example, if high mutual information indicates tightly coupled tasks, the OS can coordinate their scheduling to reduce contention or latency. Conversely, low mutual information suggests independence, allowing more flexible, asynchronous execution by the OS scheduler.

This multi-metric approach helps optimize safety margins, reduce over-provisioning, and improve the overall reliability of real-time operating systems. Furthermore, lightweight algorithms that estimate information metrics online can be integrated into embedded OS controllers, providing real-time feedback without significant computational overhead.

Security and Side-Channel Resilience in Operating Systems

Side-channel attacks, exploiting system resource sharing, timing, cache footprints, or power consumption, pose significant threats to secure multi-threaded operating systems. Information theory offers tools for analyzing and mitigating such vulnerabilities by measuring and minimizing mutual information between sensitive execution traces and observable system metrics.

Operating system schedulers designed to reduce mutual information between threads' execution traces can obscure data patterns and make it harder for adversaries to infer sensitive information. For example, OS scheduling policies that randomize thread placement or timing intervals, guided by entropy measures, can mask correlations exploitable through timing analysis or cache attacks.

Additionally, by tightly controlling mutual information between sensitive and non-sensitive tasks within the OS kernel, system designers can contain potential leakage pathways, enhancing overall security posture. This approach complements existing cryptographic measures, providing layered defense in secure operating system environments, particularly in cloud or multi-tenant infrastructures.

OPEN PROBLEMS AND RESEARCH GAPS IN OS DESIGN

Despite promising directions, several challenges remain in next-generation operating system development. Scaling information-theoretic models to large, multi-core systems requires efficient and accurate estimators of entropy and mutual information capable of running within OS kernels with minimal overhead. Reconciling conflicting objectives, such as fairness and throughput, remains complex; for example, maximizing overall throughput might lead to resource monopolization, while fairness policies could suppress high-priority workloads.

Another critical gap lies in integrating hardware counters, performance metrics, and resource monitoring data into information-theoretic OS models. This integration can enrich the representation of system states but demands careful calibration and real-time computation strategies within the kernel [18].

Finally, bridging theoretical insights with practical, deployable OS schedulers requires developing scalable algorithms, robust to noisy measurements and varying workloads, that can operate without

introducing excessive overhead or latency. As system architectures evolve toward heterogeneous, reconfigurable, and adaptive platforms, research in operating system design must continuously refine and extend these models to meet the engineering challenges of next-generation systems.

CONCLUSION

The application of information-theoretic methods to Thread Level Scheduling represents a paradigm shift in how concurrency, resource allocation, and dependency management are approached in next-generation operating system design. Entropy-based task allocation provides quantifiable guidance for dynamic load balancing and anti-bottleneck strategies within OS kernels, while mutual information-centric analysis unlocks nuanced insights about inter-thread relationships critical for reducing synchronization and communication costs managed by the operating system. Channel capacity and throughput-inspired scheduling further push OS TLS models toward theoretical limits of system efficiency, ensuring robust performance even as architectures evolve. By grounding scheduling decisions in the rigorous mathematics of information theory, operating system architects and researchers can transcend limitations of heuristic or static policies, opening pathways to more adaptive, secure, and fair scheduling frameworks within modern OS kernels. Future work should focus on developing scalable, hybrid models that blend information theory with learning-based techniques and hardware-level counters, ensuring that rapid advances in computing systems are matched by equally sophisticated OS resource management methodologies. As multi-core systems continue to proliferate in both consumer and critical infrastructure domains, the relevance and urgency of information-theoretic approaches to operating system design are only expected to grow.

REFERENCES

1. Lee J, Chwa HS, Lee J, Shin I. Thread-level priority assignment in global multiprocessor scheduling for DAG tasks. J Syst Softw. 2016 Mar 1; 113: 246–56.
2. Anjaria K, Mishra A. Thread scheduling using ant colony optimization: An intelligent scheduling approach towards minimal information leakage. Karbala Int J Mod Sci. 2017 Dec 1; 3(4): 241–58.
3. Olivier SL, Porterfield AK, Wheeler KB, Spiegel M, Prins JF. OpenMP task scheduling strategies for multicore NUMA systems. Int J High Perform Comput Appl. 2012 May; 26(2): 110–24.
4. Duran A, Corbalán J, Ayguadé E. Evaluation of OpenMP task scheduling strategies. In: Eigenmann R, de Supinski BR, editors. OpenMP in a New Era of Parallelism. IWOMP 2008. Lecture Notes in Computer Science. Vol. 5004. Berlin: Springer; 2008. p. 100–110. doi:10.1007/978-3-540-79561-2_9.
5. Chen Y, Qiu K, Chen L, Jia H, Zhang Y, Xiao L, Liu L. Smart scheduler: an adaptive NVM-aware thread scheduling approach on NUMA systems. CCF Trans High Perform Comput. 2022 Dec; 4(4): 394–406.
6. Ghazzawi HA. Scheduling approaches for large-scale complex task management. In The Proceedings of the 2nd Large Scale Complex IT Systems (LSCITS) Postgraduate Workshop. 2010; 60–66.
7. Chen F, Kodialam M, Lakshman TV. Joint scheduling of processing and shuffle phases in MapReduce systems. In 2012 Proceedings IEEE INFOCOM. 2012 Mar 25; 1143–1151.
8. Arora NS, Blumofe RD, Plaxton CG. Thread scheduling for multiprogrammed multiprocessors. In Proceedings of the tenth annual ACM symposium on Parallel algorithms and architectures. 1998 Jun 1; 119–129.
9. Madan HT, Manjunatha H, Vidyashankar M. CPU scheduling algorithms performance analysis in the RISC-V xv6 operating system environment. J Integr Sci Technol. 2025; 13(3): 1053.
10. Rajagopalan M, Lewis BT, Anderson TA. Thread Scheduling for Multi-Core Platforms. In Proceedings of HotOS'07: 11th Workshop on Hot Topics in Operating Systems, San Diego, California, USA. 2007 May 7.
11. González-Rodríguez M, Otero-Cerdeira L, González-Rufino E, Rodríguez-Martínez FJ. Study and evaluation of CPU scheduling algorithms. Heliyon. 2024 May 15; 10(9): e29959.
12. Qian J, Jiang H, Srisa-An W, Seth S, Skelton S, Moore J. Energy-efficient I/O thread schedulers

for NVMe SSDs on NUMA. In 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). 2017 May 14; 569–578.
13. Ramanathan N, Constantinides GA, Wickerson J. Concurrency-aware thread scheduling for high-level synthesis. In 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). 2018 Apr 29; 101–108.
14. Jang J, Lee H, Kim H. EDeN: Enabling low-power CNN inference on edge devices using prefetcher-assisted NVM systems. In: Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design. New York (NY): Association for Computing Machinery; 2024. p. 1–6. doi:10.1145/3665314.3670801.
15. Mu S, Deng Y, Chen Y, Li H, Pan J, Zhang W, Wang Z. Orchestrating cache management and memory scheduling for GPGPU applications. IEEE Trans Very Large Scale Integration (VLSI) Syst. 2013 Sep 11; 22(8): 1803–14.
16. Cover TM, Thomas JA. Entropy, relative entropy and mutual information. Elem Inf Theory. 1991 Mar; 2(1): 12–3.
17. Mou J, Gao K, Duan P, Li J, Garg A, Sharma R. A machine learning approach for energy-efficient intelligent transportation scheduling problem in real-world dynamic circumstances. IEEE Trans Intell Transp Syst. 2023;24:15527–39. doi:10.1109/TITS.2022.3183215.
18. Zou X, Cheng AM. Real-time multiprocessor scheduling algorithm based on information theory principles. IEEE Embed Syst Lett. 2017 Oct 10; 9(4): 93–6.

