Optimizing Energy Efficiency in Modern Computing Systems through Advanced Architectural Techniques

by Yu-Yang Ting and Wei Chen

Introduction

In the contemporary era of rapid technological advancement, optimizing energy efficiency in modern computing systems has emerged as a critical challenge and an area of intensive research. The push towards more sustainable and energy-efficient computing practices is driven by the exponential growth in data processing needs, the proliferation of cloud computing, and the ubiquitous deployment of internet-connected devices. The quest for minimizing energy consumption while maximizing performance has led to the exploration of advanced architectural techniques that promise significant improvements in energy efficiency. This literature review aims to dissect the myriad of strategies and innovations in computer architecture that contribute to energy conservation, ranging from novel processing units and memory management techniques to predictive models and energy-aware algorithms. By synthesizing findings from cutting-edge research, including studies on Non-Uniform Memory Access (NUMA) optimizations, Simultaneous Multithreading (SMT) enhancements, dynamic voltage and frequency scaling (DVFS), and the implementation of machine learning algorithms for energy prediction, this review provides a comprehensive overview of the state-of-the-art approaches to energy-efficient computing. Through the lens of recent advancements and empirical studies, we aim to highlight the effectiveness of these architectural innovations, identify prevailing challenges, and propose future directions for research in optimizing energy efficiency in modern computing systems.

Methodology

Our methodology for the literature review on optimizing energy efficiency in modern computing systems encompasses a systematic search across academic databases, applying inclusion and exclusion criteria to select relevant studies, extracting and assessing data from these studies, synthesizing findings to identify trends and gaps, and analyzing the impact of architectural techniques on energy efficiency. This approach ensures a comprehensive and critical examination of current research in the field.

Result

In this section, we delve into the multifaceted strategies and innovations that have been explored to enhance energy efficiency in modern computing systems through advanced architectural techniques. Our discussion encompasses a range of approaches from hardware optimizations, such as Non-Uniform Memory Access (NUMA) adjustments and Simultaneous Multithreading (SMT) enhancements, to software interventions like dynamic voltage and frequency scaling (DVFS) and predictive modeling. Each strategy is examined for its potential to reduce energy consumption while maintaining or improving performance, illustrating the complexity and necessity of cross-disciplinary efforts in achieving sustainable computing practices.

NUMA optimization

The article "Energy-efficient I/O Thread Schedulers for NVMe SSDs on NUMA" [1] presents a thorough investigation into the performance and energy efficiency of using a large number of parallel I/O threads. This study highlights that when managing I/O operations on Non-Uniform Memory Access (NUMA) architectures with NVMe Solid State Drives (SSDs), the impact of CPU contention—where multiple processes compete for CPU time—is significantly less detrimental than the impact of remote access. Remote access refers to the process of accessing data from SSDs that are not directly connected to the CPU attempting the access, which is a common scenario in NUMA systems. These findings suggest that optimizing for reduced remote access to NVMe SSDs could lead to more substantial performance and energy efficiency improvements than merely focusing on minimizing CPU contention in systems employing numerous parallel I/O threads.

So, they present a new algorithm on NVMe SSDs on NUMA - ENERGY-EFFICIENT I/O SCHEDULER(ESN)

Background-NUMA

In the class, we have learned about Symmetric Multiprocessing (SMP). SMP involves a system where all CPUs share the same memory resources via the same memory bus, ensuring equal access speeds for all processors, hence the term "symmetric." However, as the number of CPUs increases, memory access conflicts also rise, leading to a rapid decline in the efficiency and performance of the CPUs. On the other hand, Non-Uniform Memory Access (NUMA) divides CPUs into multiple nodes, with each node possessing its own independent memory space, allowing for high-speed interconnect communication between nodes. In NUMA architectures, the speed at which a CPU accesses memory varies depending on the node; accessing local node memory is fastest, while accessing remote node memory is slower, with speed decreasing as the distance increases. This

design inherently suffers from slower access speeds when a node's memory is insufficient and data must be retrieved from a remote node.

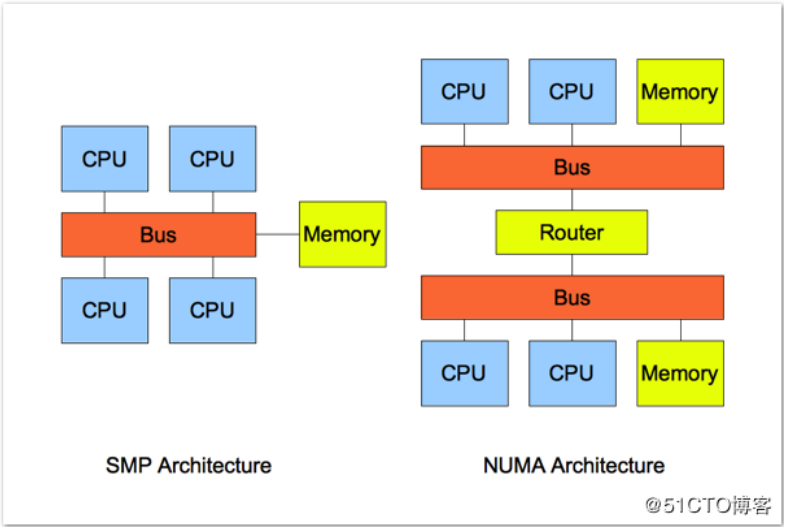


Figure 1 SMP vs NUMA

NVMe SSDs on NUMA

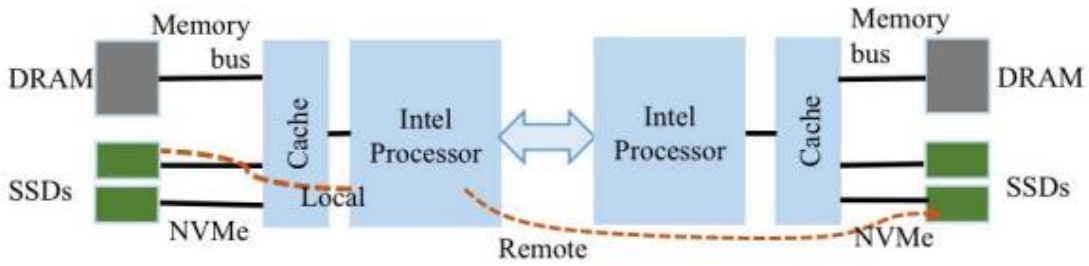


Figure 2 NVMe SSDs on NUMA

In the nuanced landscape of NUMA (Non-Uniform Memory Access) architectures, the placement of NVMe (Non-Volatile Memory express) Solid-State Drives (SSDs) becomes pivotal to system performance. Referencing the visual aids provided in the figures, we explore three settings: The first shows a single SSD coupled to one CPU node (figure 3), potentially decreasing latency and memory contention through direct local access. The second setting involves a dual-SSD setup, each connected to individual CPU nodes (figure 5), harnessing parallelism and diminishing memory traffic across nodes, which is beneficial for high-throughput demands. The third setting depicts two SSDs connected to a single CPU node (figure 4), which might enhance storage bandwidth at the cost of increasing contention for memory access. As the figures suggest, remote memory access, a fundamental aspect of NUMA systems, often entails augmented latency and access disputes. Consequently, the strategic arrangement of SSDs relative to CPU nodes

is a critical consideration, with the goal of reducing the detriments of remote access, thereby underlining the importance of tailored SSD settings to enhance the efficiency and speed of the overall system.

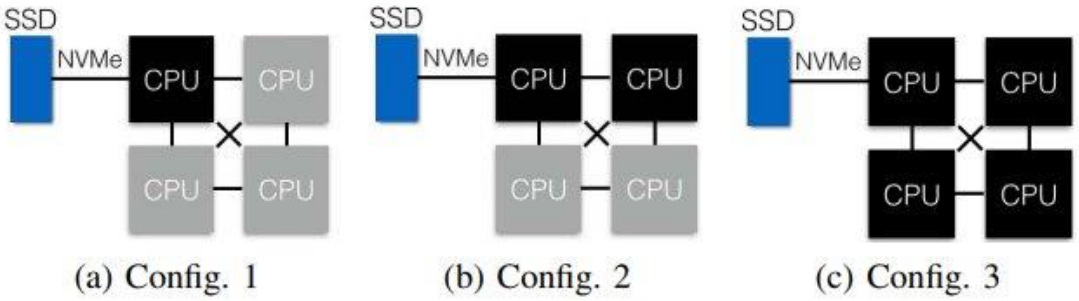


Figure 3 Single SSD NUMA setting [1]

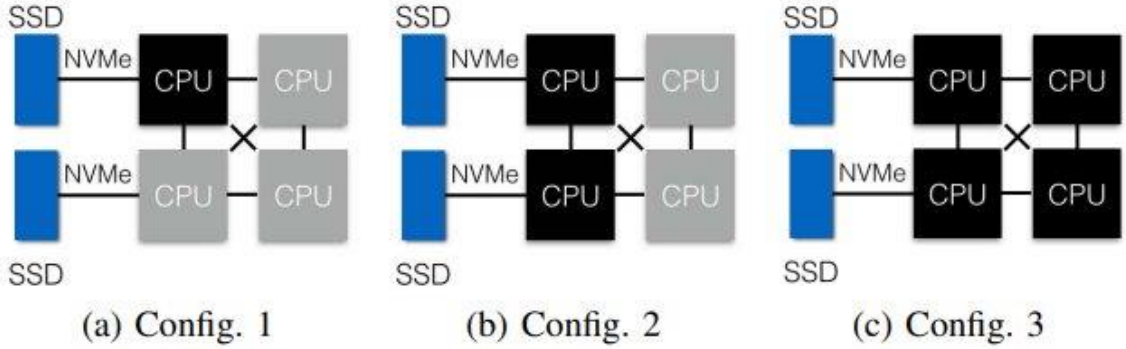


Figure 4 Dual SSDs NUMA setting 1 [1]

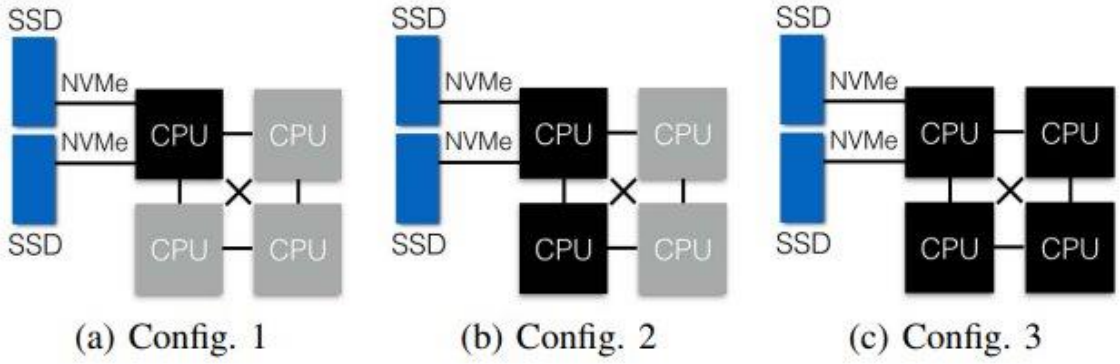


Figure 5 Dual SSDs NUMA setting 2 [1]

Expanding upon the insights presented in Figure 6, it becomes apparent that system throughput experiences a decline when the number of I/O threads surpasses the threshold of 512 in Setting 1 of Configuration 1. This specific setting involves a solitary SSD bound to a single CPU node. The degradation in performance at this juncture can be primarily attributed to a heightened contention penalty, which, in this scenario, imposes

a more substantial bottleneck than the penalty incurred through remote access. The contention penalty arises as multiple I/O threads vie for access to the limited resources of the single SSD, resulting in an increased number of conflicts and, consequently, latency. This outcome underlines the intricate balance between resource allocation and scalability within NUMA architectures. It emphasizes the need for meticulous planning in the architectural design phase to accommodate the anticipated load and access patterns, thereby ensuring that the system is not only well-tuned for current demands but also scalable for future increases in load without significant performance trade-offs.

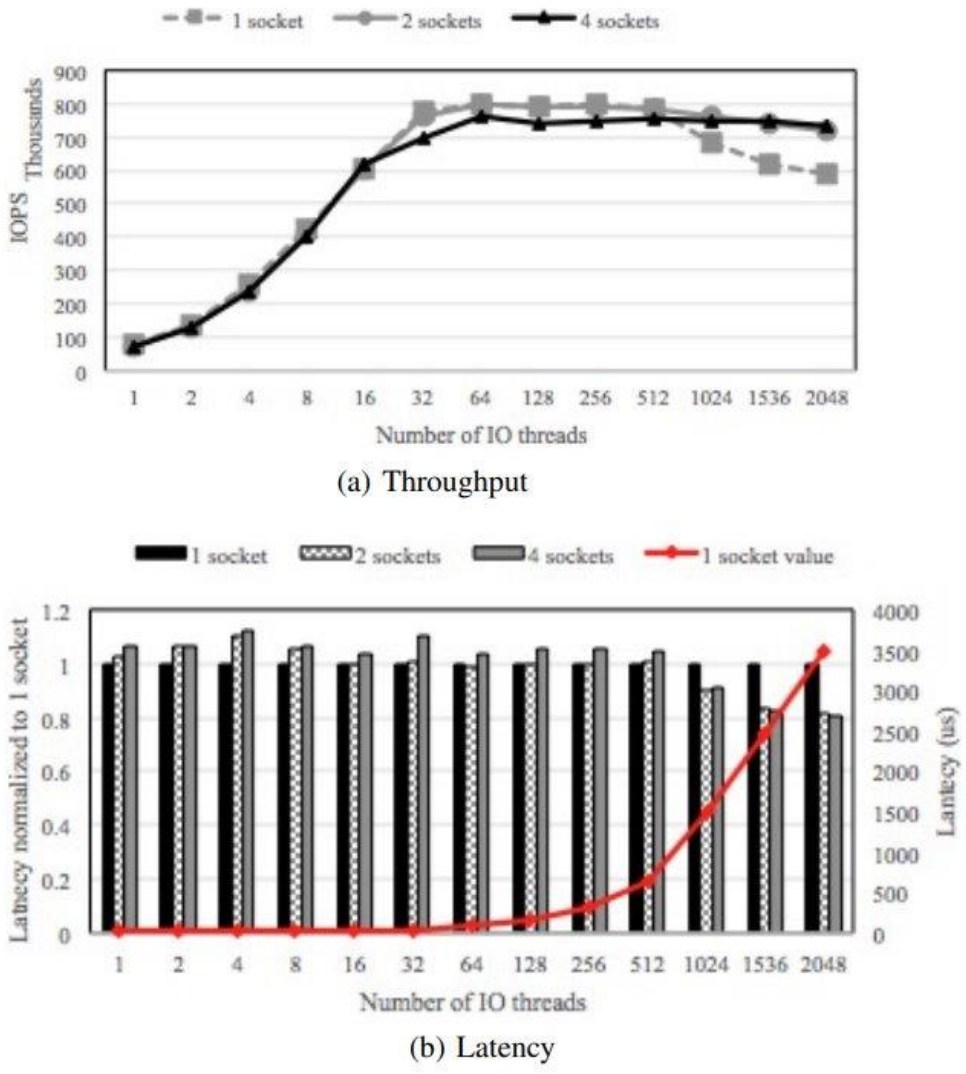


Figure 6 single SSD on NUMA benchmark

Turning our attention to the dual-SSD NUMA configurations across different settings, a distinct divergence in performance becomes evident. As illustrated in the data, Setting 1 achieves a superior maximum throughput, approximately 1600 IOPS, whereas Setting 2

peaks at merely 1200 IOPS. The critical factor contributing to this discrepancy is the unique configuration of Setting 2, where both SSDs are affiliated with a singular CPU node. Although this node has the advantage of localized access to the data stored within the SSDs, it is simultaneously hampered by the dual penalties of contention and remote access. Contention arises as the two SSDs on the same node compete for bandwidth, while other nodes incur a performance hit when attempting to access these SSDs remotely, leading to a compounded delay. In contrast, Setting 1, by distributing the SSDs across two CPU nodes, alleviates such contention and minimizes remote access requirements, thus facilitating a higher throughput. This comparison underscores the necessity of considering both the placement of storage resources and the intended access patterns to optimize NUMA system performance comprehensively.

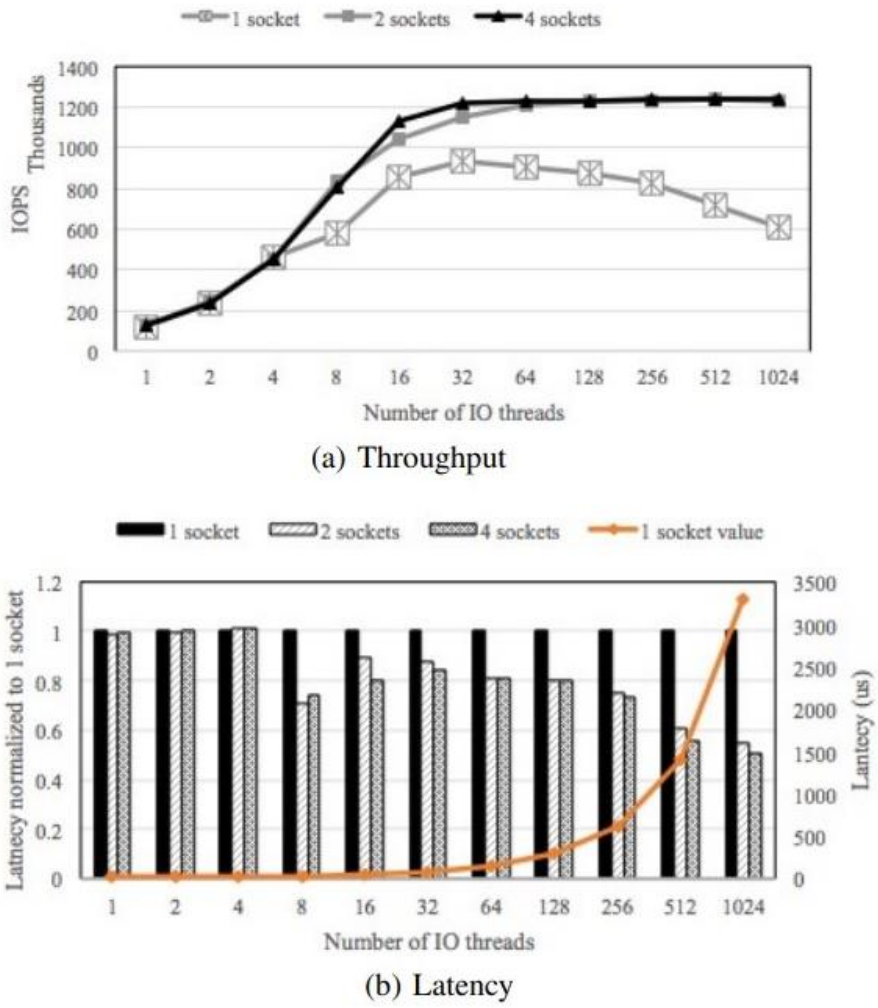


Figure 7 Setting 2 benchmark [1]

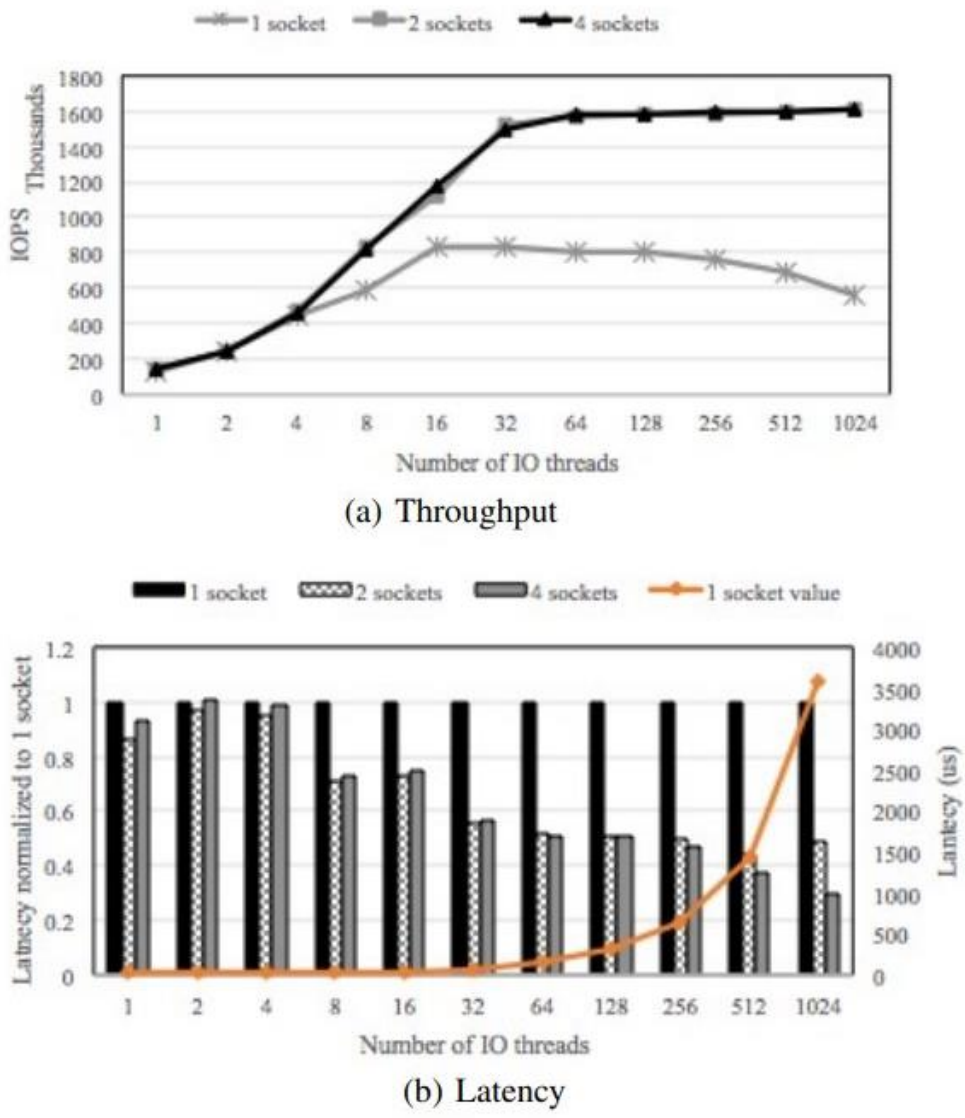


Figure 8 setting 1 benchmark [1]

Linux default scheduler-CFS

In the referenced article [1], the "default scheduler" denotes the Linux kernel's default I/O scheduler, known as the Completely Fair Scheduler (CFS). The CFS is designed to allocate time slices for process execution in a manner that aims to provide equitable CPU time to running processes, thereby ensuring a balanced distribution of computing resources. This article delves into the performance metrics of I/O operations within systems employing a single NVMe SSD and notes that prior assessments of I/O performance have predominantly been under conditions where the CFS is operational.

CFS plays a pivotal role in managing how tasks are prioritized and executed, extending its influence to the I/O performance by managing access to the storage subsystem. In the context of NUMA systems, where multiple processors are involved, CFS must also account for the additional complexity introduced by remote accesses—where data must be fetched across different memory nodes. The scheduler thus faces the dual challenge of managing equitable CPU access (its primary function) while also navigating the penalties associated with remote storage access. These penalties can exacerbate the latency issues, especially when numerous I/O threads concurrently request access to storage resources that are not locally available.

Therefore, the scheduler's behavior can significantly impact system performance, particularly in NUMA configurations. Understanding and potentially optimizing the default scheduler's strategy for handling I/O requests is crucial for enhancing overall system efficiency, particularly in settings with high I/O demands and in systems that are sensitive to latency variations due to remote access penalties.

Energy-efficient I/O scheduler (ESN)

Algorithm 2: The proposed energy-efficient scheduler

Platform identification: Identify the number of CPU cores per socket (N_{cpus}), the local\nearest_neighbor\remote CPU sockets;
Pre-scheduling: Calculate the balance point (the number of I/O processes per local CPU socket, $N_{I/O}$) that the contention penalty equals the NUMA penalty;
Initialization: Launch the I/O application (A) and the helper script (S) concurrently;
Scheduling: 1. Identify the number of existing I/O processes on local CPU socket, $N_{existing}$;
if $N_{existing}$ *is less than* $N_{I/O}$ **then**
    Map part of the I/O processes in A to local CPU socket, $N_{I/O} - N_{existing}$;
    while *not all I/O processes in A are mapped* **do**
        Map $n * N_{I/O}$ of the I/O processes in A to n adjacent neighbor CPU sockets;
    end
else
    while *not all I/O processes in A are mapped* **do**
        Map $n * N_{I/O}$ of the I/O processes in A to n adjacent neighbor CPU sockets;
    end
end
Periodically re-balance the load when one or more I/O processes on local CPU socket finish.

Figure 9 ESN algorithm

The ESN (Energy-efficient Scheduler for NUMA systems) algorithm presents a set of enhancements tailored for the unique demands of NUMA architectures. It focuses on distributing loads equitably across nodes to avert bottlenecks, thereby optimizing resource utilization. By prioritizing local memory access, ESN reduces the latency typically associated with remote accesses, leading to speedier data retrieval and diminishing energy expenditure. The algorithm also curtails resource contention, which is essential for maintaining consistent performance levels, especially under the strain of multiple concurrent processes.

Furthermore, ESN showcases impressive scalability. It adapts to the demands of escalating parallel processes without the necessity for manual intervention, a critical feature for dynamic high-performance computing environments. In terms of energy conservation, ESN strategically aligns I/O threads with CPU sockets, taking into account both the number of processes and the location of memory accesses. This alignment ensures optimal system performance while concurrently reducing power consumption by permitting the idling of CPUs not engaged in active tasks. The core objective of ESN is thus to intelligently map an increased number of I/O threads to the same CPU, while judiciously balancing performance considerations. The culmination of these mapped threads allows for the potential downscaling of energy usage by placing unused CPUs in a low-power state, thereby enhancing the overall energy efficiency of the system.

Improvement

Figures 10 through 13 provide a comparative performance analysis between the ESN scheduler and the traditional CFS scheduler for systems utilizing a single NVMe SSD. These figures highlight that the ESN scheduler not only preserves a similar level of throughput to that of the CFS but also excels in diminishing latency. Moreover, ESN demonstrates a notable reduction in energy consumption, an advantage achieved by its ability to idle CPUs when they are not in active use. This capability of ESN to maintain operational efficiency while enhancing power management underscores its suitability for energy-conscious computing environments.

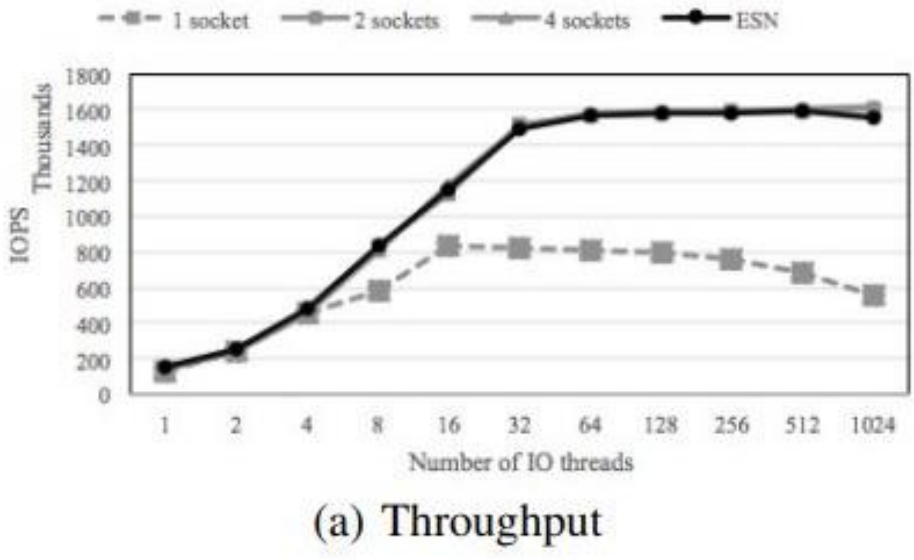
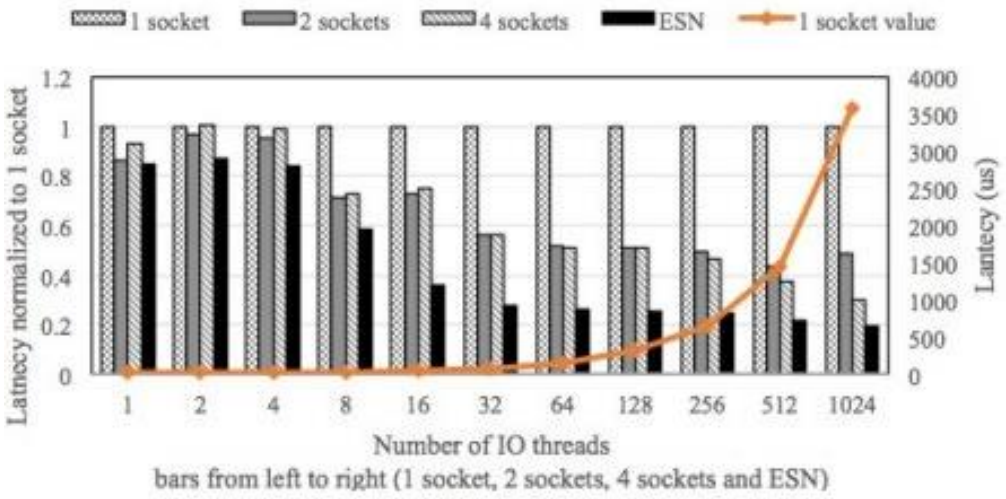
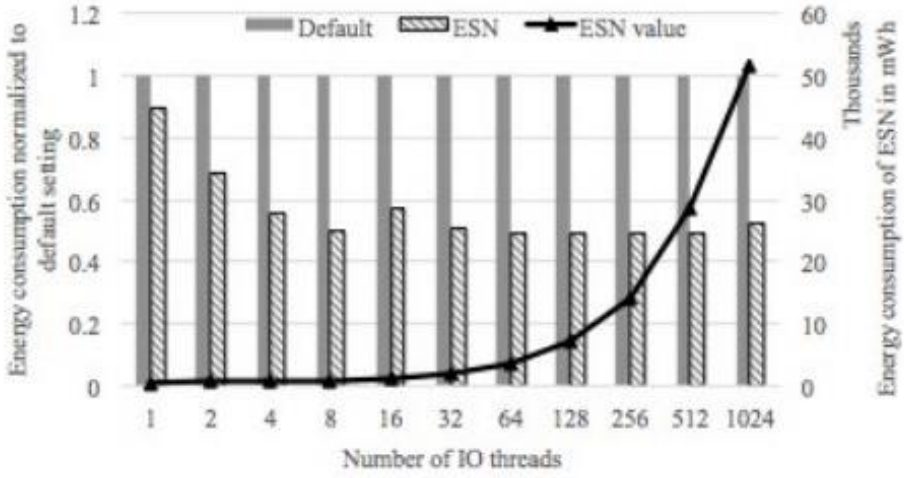


Figure 10 CFS vs ESN [1]



(b) Latency

Figure 11 CFS vs ESN [1]



(c) Energy

Figure 12 CFS vs ESN [1]

	32 I/Os	128 I/Os	512 I/Os	1024 I/Os
Throughput	1	1	1	0.9
Latency	0.625	0.7	0.7	0.3
Energy	0.6	0.3	0.67	0.75

Figure 13 CFS vs ESN [1]

Discussion & Conclusion

References

[1] H. J. W. S.-a. a. S. S. J. Qian, Energy-efficient I/O Thread Schedulers for NVMe SSDs on NUMA, 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2017.