SPDK+: Low Latency or High Power Efficiency? We Take Both

Endian Li[1], Shushu Yi[1], Li Peng[1], Qiao Li[2], Diyu Zhou[1], Zhenlin Wang[3], Xiaolin Wang[1]
Bo Mao[2], Yingwei Luo[1], Ke Zhou[4], Jie Zhang[1]
Computer Hardware and System Evolution Laboratory,
Peking University[1], Xiamen University[2], Michigan Tech[3], Huazhong University of Science and Technology (HUST)[4]

ABSTRACT

SPDK, as one of the most efficient I/O storage software, is capable of delivering the lowest I/O latency. Unfortunately, the polling mechanism in SPDK wastes tremendous CPU clock cycles, especially under small I/O operations and low queue depths. Although SPDK supports the conventional interrupt method, it does not improve power efficiency under such circumstances. To address this issue, we propose SPDK+, which enables the user interrupt feature in the SPDK to achieve both low latency and high power efficiency. Specifically, SPDK+ employs user interrupt handling to directly process MSI-X interrupts from SSD devices and utilizes user wait instructions during IO wait periods to conserve power. The comprehensive evaluation results show that SPDK+ achieves up to 49.5% power efficiency improvement while keeping the I/O latency almost unchanged compared with SPDK.

CCS CONCEPTS

• **Computer systems organization** → *Special purpose systems*; • **Hardware** → **Chip-level power issues**.

KEYWORDS

Power Efficiency, SPDK, Polling, Interrupt

1 INTRODUCTION

Over the past decade, solid-state drives (SSDs) have been continuously evolving towards extremely high-performance storage devices, which meet the high demands for low I/O latency in diverse computing domains such as clouds, high-performance computing, and databases [1, 2, 12, 26]. While SSDs can deliver promising low I/O latency (e.g., 5~25 us in PCIe 5.0 SSDs [36]), the traditional storage software stack implemented in the kernel space imposes significant overheads in terms of latency due to the intensive user-kernel context switch, lock-based contentions, and tedious control path [9, 11, 34, 35].

User-space storage stack designs, such as *Storage Performance Development Kit (SPDK)* [35], have become inevitable candidates to replace traditional kernel-space ones, which can unleash the full performance potential of the latest PCIe 5.0 SSDs [8]. SPDK is both lightweight and high-performance as it employs a user-space I/O engine, facilitates concurrent multi-thread accesses based on a lock-free principle, and follows the run-to-complete thread model [35]. In addition, it adopts an I/O polling mechanism, which employs a dedicated CPU core to keep checking the status of I/O requests submitted to the underlying SSDs until their completion. While the polling mechanism can significantly shorten the communication latency between the host and SSD devices, it, unfortunately, burdens CPU usage all the time, resulting in poor CPU power efficiency [9, 19].

One common practice to reduce CPU usage is to employ the interrupt-based communication method. Specifically, once an I/O request is submitted, the working core switches to sleep or execute other tasks until the underlying SSD informs it of I/O completion via an interrupt. In this manner, the CPU power cost for polling can be eliminated. However, the conventional interrupt mechanism is implemented in the kernel space and SPDK handles the NVMe completion queue (CQ) [25] is implemented in the user space. In other words, after an interrupt is triggered, the working CPU must schedule SPDK in order to process the I/O completion.[32]. It is worth noting that this tedious control path causes 76% higher latency for the interrupt service routines and thread scheduling (cf. Section 2.2 for details). To sum up, the existing host-SSD communication methods (i.e., polling and interrupt) cannot simultaneously achieve low latency and high power efficiency, called *latency-energy dilemma*.

HotStorage '25, July 10–11, 2025, Boston, MA, USA

ACM ISBN 979-8-4007-1947-9/25/07…$15.00
https://doi.org/10.1145/3736548.3737824

The user interrupt feature supported by 4^{th}-generation Intel Xeon Scalable processors and its successors [24] offers a favorable opportunity to resolve this dilemma. To be specific, when a user interrupt is triggered, the CPU core only reserves three registers (i.e., *rip*, *rsp*, and *rflag*) rather than the entire process context (i.e., register files and memory stack). By doing so, the user interrupt feature reduces the interrupt response latency to only 2 us.

Inspired by the aforementioned features, we propose *SPDK+*, a user-interrupt enhanced user-space storage stack design, which can achieve both low latency and high power efficiency simultaneously. Specifically, considering that NVMe SSDs by default adopt the MSI-X interrupt [18, 21] to notify the host of I/O completion, SPDK+ maps the MSI-X interrupt issued by the NVMe SSDs to a specific entry of the interrupt remapping table [4, 5]. When NVMe SSDs trigger the MSI-X interrupt, the CPU maps the interrupt to the user interrupt based on the interrupt remapping table. The CPU then saves the current state and passes the control flow to the user interrupt handler. The handler then processes the interrupt. Upon completion, it executes the UIRET instruction to restore the saved context and resume normal execution.

Note that the user interrupt feature is incompatible with traditional power-saving methods such as sleeping or task switching. This is because the registration and response of the user interrupt should be handled by the same kernel-level thread. Since existing power-saving mechanisms make the I/O process inactive, they can result in interrupt failure. We observe that Intel ISA has added a set of new instructions, such as *tpause*, *umonitor*, and *umwait*, for the purpose of low power. SPDK+ utilizes such instructions to construct a low-power user-level thread within the I/O process. When the working CPU core submits an I/O request, it switches to the low-power thread, waiting for the incoming MSI-X interrupt. Our evaluation results show that SPDK+ enhances power efficiency by up to 49.5% with only a negligible latency increase compared to SPDK.

2 BACKGROUND AND MOTIVATION

2.1 SPDK and Its Implementations

SPDK. *Storage performance development kit*, also known as *SPDK* [35], is a user-space storage engine that can mitigate the software overheads induced by the traditional storage stack. Specifically, SPDK employs a block device abstraction called *Bdev* to perform the same functions as the block device layer in the kernel. SPDK also implements a user-space, asynchronous, polling-based NVMe driver [35]. Therefore, users can directly access NVMe SSDs in user space without trapping into the intricate kernel.

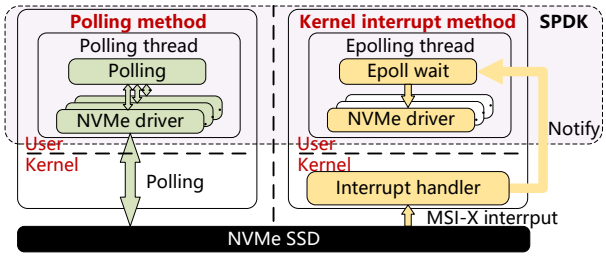
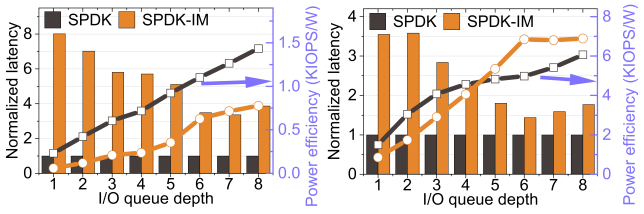


Figure 1: Internal details of SPDK.

Polling method. The left side of Figure 1 illustrates the operation of the polling method. The polling method in SPDK enables the NVMe driver to continuously check for the completion of I/O operations. This is done by running tight busy-wait loops on dedicated CPU cores. Each CPU core typically executes one polling thread, which has one or more pollers. These pollers are essentially functions that perform specific tasks, such as checking the CQ for I/O operations. The polling thread will repeatedly call these pollers in a loop, ensuring that any completed I/O requests are processed with minimal delay. However, the polling threads consume dedicated CPU cores for polling, which is infeasible in systems with limited resources. Furthermore, continuous polling leads to high CPU usage, which is not ideal in scenarios where power efficiency is a concern (e.g., data centers [17] and HPC [16]).

Kernel-space interrupt method. SPDK has recently introduced an interrupt method to adapt to fluctuating I/O loads in real-world scenarios, optimizing its power efficiency [30]. The interrupt method integrated in SPDK is, in general, similar to its implementation in other interrupt-based I/O software stacks [28]. Both require SSD devices to send an MSI-X interrupt upon I/O completion [18, 21, 28]. They also yield the current core, enabling the kernel to schedule other tasks or put the core into a power-saving state while waiting for I/O completion. The key difference is that the traditional I/O storage software stack processes the CQ directly in the kernel upon receiving MSI-X interrupts from the SSD devices, and then schedules the user thread to return the results. In contrast, SPDK processes the CQ in the user space. The right side of Figure 1 shows how the SSD device notifies SPDK of I/O completion via interrupts. Specifically, SPDK registers an MSI-X interrupt handler through the *VFIO* [33] interface and binds it to an *eventfd* file descriptor [22]. The epolling thread is an interrupt-based version of the polling thread, supporting blocking waits for interrupts. After handling an MSI-X interrupt, the kernel changes the epolling thread state from blocked to runnable. The epolling thread will execute the NVMe driver to process the CQ when it is running. Therefore, SPDK cannot avoid the significant overheads of interrupt handling and thread scheduling. To sum up, the interrupt method aids in reducing CPU power consumption to some extent but severely degrades performance,



(a) 4KiB random read @ 1 core. (b) 4KiB random write @ 1 core.

Figure 2: Latency and power efficiency comparison.

thereby failing to achieve gains in CPU power efficiency. For simplicity, we use the terms "CPU power efficiency" and "power efficiency" interchangeably.

2.2 Preliminary Study

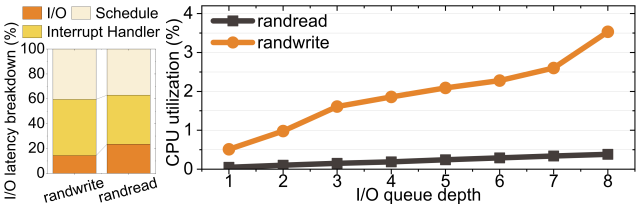
To quantitatively investigate the impact of these methods on I/O latency and CPU power efficiency, we set up an experiment, which configures the microbenchmark to 4 KiB I/O with low queue depths and employs 1 core with 1 SSD. We examine them on SPDK with the polling (`SPDK`) and interrupt (`SPDK-IM`) methods (cf. Section 4.1 for experiment details).

Key observation 1. *Contrary to common sense, the polling method exhibits superior power efficiency than the interrupt method at low I/O queue depths.*

Figure 2 presents the latency and power efficiency of the polling and interrupt methods. In this article, we define CPU power efficiency as the ratio of the average number of I/O processes per second to the average power consumption of the CPU. CPU power efficiency and power efficiency are interchangeable in this paper. The polling method achieves superior performance (32.1 us and 4.9 us for random read and write, respectively) but has higher power consumption (137.6 W). In contrast, the interrupt method shows lower performance (257.1 us and 17.3 us for random read and write, respectively) with reduced power consumption (68.9 W). Interestingly, Figure 2 shows that the interrupt method does not improve power efficiency compared to the polling method at low I/O queue depths. The poor power efficiency of the interrupt method is attributed to its long latency, which exceeds that of the polling method too much, significantly negating any power benefits brought by the interrupt method.

To further analyze the latency difference, we use perf [6] to decompose the I/O latency of random requests in the interrupt method, which is shown in Figure 3(a). We categorize the latency into three parts: `I/O` is the SSD service time, `Interrupt Handler` covers interrupt issuance to handler completion, and `Schedule` spans scheduling start to CQ completion. Experiments show that over 76% of I/O latency stems from interrupt handling and scheduling delays.

Key observation 2. *Despite its superiority in small I/Os, the polling method does not fully utilize CPU clock cycles, leaving significant room for power efficiency optimization.*



(a) Interrupt I/O latency analysis. (b) CPU utilization of polling method.

Figure 3: Interrupt I/O latency analysis and CPU utilization of polling method.

To investigate the potential for enhancing polling power efficiency, we perform an experiment to quantify the actual CPU utilization during I/O polling. The results are shown in Figure 3(b). The findings indicate that only 3.5% of CPU clock cycles are effectively utilized, leaving the remaining CPU cycles wasted by waiting. Even when a single core simultaneously polls seven disks, with each disk having a queue depth of 8, only 15.17% of its clock cycles are actively utilized. The low utilization is mainly attributed to the fact that our tests consist solely of I/O-related operations, for which SPDK incurs minimal software overhead. Consequently, I/O waiting becomes the primary source of latency, significantly lowering overall utilization. These results suggest that polling, even in scenarios involving small I/O operations, exhibits severe power inefficiency, implying the importance of designing optimization strategies to enhance power efficiency.

3 DESIGN

3.1 Key Insights

Opportunity 1. *Intel user interrupt technique simplifies the conventional kernel interrupt mechanism, which, in turn, effectively reduces the I/O latency.*

Intel recently introduced *User Interrupt* (UINTR), a new hardware feature for its Sapphire Rapids processors and successors, to offer a low-cost mechanism for inter-processor communication [24]. UINTR allows interrupts to be passed directly to an interrupt handler without switching address spaces or privilege levels. To support inter-processor communication, each core is equipped with a dedicated set of hardware for triggering and handling user interrupts.

Compared to the traditional kernel interrupts, UINTR processing is faster and more streamlined. The user directly sends a user interrupt to the target core from the user space via the x86 instruction *senduipi*. Before sending the user interrupt, `senduipi` sets the *Posted-Interrupt Requests* (*PIR*) field in the *User Posted Interrupt Descriptor* (UPID) to indicate the sender's identity. The target core then compares the received interrupt vector with the *User-Interrupt Notification Vector Register*. If matched, the process continues; otherwise, the interrupt is treated as a standard interrupt. Next, the

CPU hardware sets the *User Interrupt Request Register* based on the PIR. If the PIR is empty, the hardware ignores the interrupt. When the CPU executes in user mode, the user interrupt is triggered. Several registers, including *rsp* and *rip*, are saved to the stack, and control is handed over to the user interrupt handler. Finally, the handler will execute the *UIRET* instruction to resume normal execution.

Opportunity 2. *Intel user wait instructions enable CPU power reduction within the user space.*

User-space processes always need to wait for an event for just a few microseconds, such as I/O completion or inter-thread communication. Traditional kernel-based mechanisms, like *sleep* or *pipe*, are inefficient in this circumstance due to their prolonged scheduling latency. Therefore, developers use busy-waiting to solve this problem. However, busy-waiting continues to occupy CPU resources and consume more power. To address this, Intel introduced a new set of low-power instructions (e.g., *umonitor*, *umwait*, and *tpause*), which allow efficient waiting with lesser CPU power, thereby improving performance and power efficiency. Umonitor and umwait provide a mechanism to suspend CPU execution until a write operation accesses data at the specified address, while tpause enables the current thread to sleep for a specified amount of time (up to 100 us). The umwait and tpause instructions allow the CPU to enter the power-saving states *C0.1* and *C0.2* during execution, thereby reducing CPU power consumption [3]. In practice, DPDK has utilized these instructions to achieve up to a 20% improvement in power efficiency during idle periods [14].

3.2 Overview

Inspired by the aforementioned analysis and key insights, we propose SPDK+, which enables the user interrupt feature in the SPDK to achieve both low latency and high power efficiency. Figure 4 illustrates the overall design of SPDK+, which integrates the user interrupt and user wait instruction technologies. To be specific, SPDK+ modifies the existing user interrupt mechanism to take over the role of processing MSI-X interrupts sent by SSD devices. It also leverages the Intel wait instructions to create low-power threads, which can achieve power saving during I/O completion.

3.3 Design Details of SPDK+

MSI-X interrupt handling. In the traditional kernel interrupt method, the CPU core switches to other user processes while waiting for I/O completion. After handling the interrupt, it schedules SPDK to resume execution at an appropriate time. This procedure introduces long interrupt handling latency and scheduling delays. In order to reduce interrupt handling and scheduling overheads in the kernel interrupt mode, one straightforward approach is to enable direct user-space processing of MSI-X interrupts. This process can be divided into two steps: routing MSI-X interrupts to the interrupt vector corresponding to the user interrupt, and preparing the prerequisites for triggering the user interrupt. The former is achieved by modifying the VFIO module, which sets the interrupt remapping table entry of the MSI-X interrupt to the user interrupt vector (①). The latter requires that the PIR should not be empty. Unfortunately, MSI-X interrupts do not automatically update the PIR. As PIR remains empty, the CPU hardware ignores the interrupt. To address this, SPDK+ proactively executes the `senduipi` instruction to populate the PIR during user interrupt handler registration and after the interrupt handling procedure. Nevertheless, this design introduces an unnecessary interrupt. Fortunately, `senduipi` checks the *Suppress Notification* (*SN*) bit in the UPID before sending a real interrupt. If *SN* is set to 1, the interrupt is not sent. Therefore, we set the SN bit in advance before sending `senduipi`. By doing so, SPDK+ directly processes device-generated MSI-X interrupts in user space via user interrupt hardware features [15, 24].

User-level thread switching and power saving. Similar to kernel interrupt, the user interrupt technique leaves CPU time for waiting I/O completion, which should be carefully utilized to save CPU power. However, the user interrupt mechanism cannot make the core sleep or switch to other user processes because user interrupt must be handled on the same thread, which defines the user interrupt handler. To address this, we propose to switch the CPU core to a low-power thread instead of other user processes, which not only reduces the intrusion into SPDK, but also increases the possibility of further optimization. Specifically, we design a user-level thread framework that includes two types of user-level threads: the original polling thread and our custom low-power thread. When there are no I/O operations to process, the polling thread can proactively switch to the low-power thread and wait for an MSI-X interrupt. The low-power thread repeatedly executes the tpause instruction until an MSI-X interrupt arrives. umwait is not adopted in our implementation, as it would necessitate intrusive modifications to the lower layers of SPDK in order to monitor changes in the completion queue, thereby compromising the integrity of SPDK's architectural design. Upon interrupt arrival, indicating that I/O processing is required, the interrupt handler directly calls the thread-switching function to switch back to the polling thread for further processing. After processing, it switches back to the low-power thread. Through this design, we can directly and easily use idle time to save power with minimal thread-switching overhead.

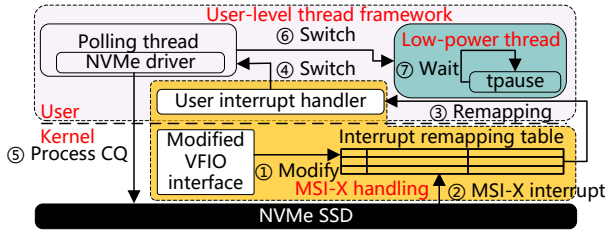


Figure 4: Architecture of SPDK+.

3.4 Put All Pieces Together

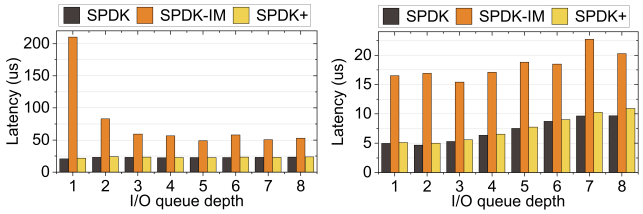
By integrating the aforementioned techniques, SPDK+ processes the I/O completion as follows. When the SSD completes an I/O operation, it sends an MSI-X interrupt to notify the CPU (①). This interrupt will be mapped to the user interrupt vector according to the interrupt remapping table (②). This interrupt preempts the currently executing low-power thread and activates the user interrupt handler. The handler then executes the user-level thread-switching function, transferring control to the polling thread (④). The polling thread processes the CQ (⑤), retrieves the completed NVMe commands, and performs subsequent processing. Once all operations are completed, the polling thread calls the thread-switching function to switch back to the low-power thread (⑥). While waiting for I/O completion, the low-power thread continuously executes the tpause instruction, pausing the CPU to reduce power consumption until the next interrupt arrives (⑦). Through this workflow, SPDK+ maximizes the use of low-power instructions while waiting for I/O completion, significantly improving CPU power efficiency.

4 EVALUATION

4.1 Experimental Setup

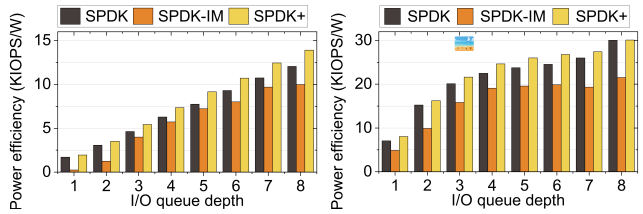
Methodology. We conduct the experiments on a server equipped with a 46-core Intel® Xeon™ PLATINUM 8558 CPU [13], and up to seven 1 TB PCIe 5.0 TiPro9000 SSDs [36]. We disable the CPU hyper-threading feature [23] to reduce power consumption and set the CPU frequency scaling governor as ondemand [27]. Our implementation is based on Linux v6.8.10 [7], with modifications to its VFIO interface to support our design. Additionally, we utilize spdk_nvme_perf v24.09 [31] to evaluate the performance of different designs and use *Model Specific Register* [10] to monitor the CPU power consumption. By default, we use seven I/O cores, each accessing one SSD.

Subjects. We compare SPDK+ with two other designs. (1) SPDK [32]: a state-of-the-art user-space storage stack, which employs the polling mechanism to obtain I/O completion information; (2) SPDK-IM [32]: the interrupt-based variant of SPDK, which enables CPU switch to another user process during waiting for I/O completion.



(a) 4KiB random read @ 7 cores. (b) 4KiB random write @ 7 cores.

Figure 5: Average latency comparison.



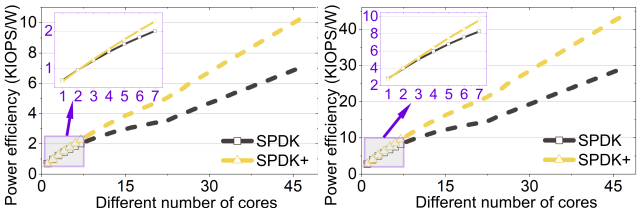
(a) 4KiB random read @ 7 cores. (b) 4KiB random write @ 7 cores.

Figure 6: Power efficiency comparison.

4.2 Overall Comparison

Latency. We measured the latency of different designs at varying queue depths, the results of which are shown in Figure 5. Note that the sequential read and write test results are similar to the random ones. We omit the results for space limitations. Benefiting from the internal cache of the SSD, the write latency of the SSD is very low at low queue depths. Compared with SPDK, SPDK-IM experiences 249% and 170% higher latency on average in read and write tests, respectively, due to the excessive time spent in the interrupt handling procedures and scheduling routines. In comparison, the latency of SPDK and SPDK+ is very similar. This is because the SPDK+ process always runs in the foreground. This prevents SPDK+ from waiting for thread scheduling after IO completion, effectively eliminating the scheduling delay seen in SPDK-IM. Additionally, SPDK+ leverages user interrupt technology, which significantly speeds up interrupt handling and further reduces overall latency.

Power efficiency. Figure 6 shows the power efficiency of different methods with varying queue depths. SPDK+ achieves the best power efficiency across all scenarios. To be specific, it outperforms SPDK by 14.6% ~ 18.2% in random reads and 0.1% ~ 13.6% in random writes. The power efficiency improvement of SPDK+ is due to its use of UINTR and user wait instructions. This allows the majority of the time spent waiting for IO completion to be executed with the *tpause* instruction, reducing CPU power consumption and ultimately enhancing overall power efficiency.



(a) 4KiB random read @ 1-46 cores. (b) 4KiB random write @ 1-46 cores.

Figure 7: Power efficiency with increasing core count.

4.3 Sensitivity Study

We measured the power efficiency trends of SPDK and SPDK+ with varying core counts, the results of which are shown in Figure 7. In this experiment, each core managed one SSD, each SSD had a single queue pair, and each queue was configured with a depth of 1. We measured the actual power efficiency of SPDK and SPDK+ with 1~7 CPU cores and SSDs. The power efficiency of more cores is derived based on the theoretical performance (IOPS) of 8~46 SSDs and the measured power consumption of 8~46 CPU cores. The power efficiency gap between SPDK+ and SPDK widens as the core count increases, with SPDK+ achieving improvements of up to 15.4% (7 cores) and up to 49.5% (46 cores). This is because as the number of cores increases, the disadvantage of low CPU utilization in the polling method is further amplified. This, in turn, makes the power efficiency gap larger.

5 CONCLUSION

In this paper, we propose SPDK+, which enhances SPDK with UINTR to achieve both low latency and high power efficiency. Evaluation results show that SPDK+ can improve power efficiency by up to 49.5% at varying queue depths.

In future work, we plan to investigate methods of SPDK full-stack optimization to improve power efficiency, especially for NVMe-oF. We will extend UINTR and revamp the scheduling of user-level threads with more optimizations from the I/O path of SPDK, which is aimed at improving power efficiency further while maintaining overall I/O latency. We also consider the collaboration of SPDK+ and hybrid polling. Note that restricted by the communication overhead between the user and kernel space, the hybrid polling [9, 19, 20, 29] implemented in the kernel cannot achieve optimal performance and power efficiency [11]. UINTR in SPDK+ provides high-precision timing directly in the user space, enabling the implementation of hybrid polling in the user space.

ACKNOWLEDGMENT

We sincerely thank the anonymous reviewers for their insightful comments and feedback. This work is mainly supported by the National Key Research and Development Program of China under Grant No. 2023YFB4502702, the Natural Science Foundation of China under Grant No. 62332021 and 62472007. Dr. Luo is partly supported by the National Natural Science Foundation of China under Grant No. 62032001. Dr. Mao is partly supported by the National Natural Science Foundation of China under Grant No. U22A2027. Dr. Jie Zhang is the corresponding author.

REFERENCES

[1] Jalil Boukhobza, Pierre Olivier, Wen Sheng Lim, Liang-Chi Chen, Yun-Shan Hsieh, Shin-Ting Wu, Chien-Chung Ho, Po-Chun Huang, and Yuan-Hao Chang. 2025. A Survey on Flash-Memory Storage Systems: A Host-Side Perspective. *ACM Trans. Storage* (March 2025). https://doi.org/10.1145/3723167 Just Accepted.

[2] Che-Wei Chang, Geng-You Chen, Yi-Jung Chen, Chia-Wei Yeh, Pei Yin Eng, Ana Cheung, and Chia-Lin Yang. 2017. Exploiting Write Heterogeneity of Morphable MLC/SLC SSDs in Datacenters with Service-Level Objectives. *IEEE Trans. Comput.* 66, 8 (2017), 1457–1463. https://doi.org/10.1109/TC.2017.2677425

[3] Jonathan Corbet. 2019. Short waits with umwait. *LWN.net* (2019). https://lwn.net/Articles/790920/

[4] Intel Corporation. 2021. *Intel® Virtualization Technology for Directed I/O Architecture Specification.* Technical Report. Intel Corporation. http://kib.kiev.ua/x86docs/Intel/VT-d/D51397-001.pdf

[5] Huiying Cui. 2024. I/O Processing Performance Optimization for Embedded Real-Time Virtualization Systems. In *Proceedings of the 2024 4th International Conference on Artificial Intelligence, Automation and High Performance Computing* (Zhuhai, China) *(AIAHPC '24)*. Association for Computing Machinery, New York, NY, USA, 314–319. https://doi.org/10.1145/3690931.3690984

[6] Arnaldo Carvalho de Melo. 2010. The New Linux 'perf' Tools. In *Linux Kongress*, Vol. 18. 1–42. http://oldvger.kernel.org/~acme/perf/lk2010-perf-acme.pdf

[7] Linux Kernel Developers. 2024. Linux Kernel 6.8.10 ChangeLog. https://www.kernel.org/pub/linux/kernel/v6.x/ChangeLog-6.8.10.

[8] Diego Didona, Jonas Pfefferle, Nikolas Ioannou, Bernard Metzler, and Animesh Trivedi. 2022. Understanding modern storage APIs: a systematic study of libaio, SPDK, and io_uring. In *Proceedings of the 15th ACM International Conference on Systems and Storage* (Haifa, Israel) *(SYSTOR '22)*. Association for Computing Machinery, New York, NY, USA, 120–127. https://doi.org/10.1145/3534056.3534945

[9] Assaf Eisenman, Darryl Gardner, Islam AbdelRahman, Jens Axboe, Siying Dong, Kim Hazelwood, Chris Petersen, Asaf Cidon, and Sachin Katti. 2018. Reducing DRAM footprint with NVM in Facebook. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 42, 13 pages. https://doi.org/10.1145/3190508.3190524

[10] Part Guide. 2011. Intel® 64 and ia-32 architectures software developer's manual. *Volume 4: Model-specific Registers* (2011), 136.

[11] Bryan Harris and Nihat Altiparmak. 2022. When poll is more energy efficient than interrupt. In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems* (Virtual Event) *(HotStorage '22)*. Association for Computing Machinery, New York, NY, USA, 59–64. https://doi.org/10.1145/3538643.3539747

[12] Jian Huang, Anirudh Badam, Laura Caulfield, Suman Nath, Sudipta Sengupta, Bikash Sharma, and Moinuddin K. Qureshi. 2017. FlashBlox: achieving both performance isolation and uniform lifetime for virtualized SSDs. In *Proceedings of the 15th Usenix Conference on File and Storage Technologies* (Santa clara, CA, USA) *(FAST'17)*. USENIX Association, USA, 375–390.

[13] Intel Corporation. 2023. *Intel® Xeon® Platinum 8558 Processor Specifications*. Intel Corporation. https://www.intel.com/content/www/us/en/products/sku/237255/intel-xeon-platinum-8558-processor-260m-cache-2-10-ghz/specifications.html

[14] Intel Corporation. 2023. *Power Management – User Wait Instructions Power Saving for DPDK PMD Polling Workloads Technology Guide*. Technology Guide. Intel Corporation, Santa Clara, CA, USA. https://www.intel.com/content/www/us/en/content-details/751859/power-management-user-wait-instructions-power-saving-for-dpdk-pmd-polling-workloads-technology-guide.html

[15] Yuekai Jia, Kaifu Tian, Yuyang You, Yu Chen, and Kang Chen. 2024. Skyloft: A General High-Efficient Scheduling Framework in User Space. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles* (Austin, TX, USA) *(SOSP '24)*. Association for Computing Machinery, New York, NY, USA, 265–279. https://doi.org/10.1145/3694715.3695973

[16] Shoaib Kamil, John Shalf, and Erich Strohmaier. 2008. Power efficiency in high performance computing. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. 1–8. https://doi.org/10.1109/IPDPS.2008.4536223

[17] Avita Katal, Susheela Dahiya, and Tanupriya Choudhury. 2022. Energy efficiency in cloud computing data centers: a survey on software technologies. *Cluster Computing* 26, 3 (Aug. 2022), 1845–1875. https://doi.org/10.1007/s10586-022-03713-0

[18] Babar Khan, Carsten Heinz, and Andreas Koch. 2024. The Open-source DeLiBA2 Hardware/Software Framework for Distributed Storage Accelerators. *ACM Trans. Reconfigurable Technol. Syst.* 17, 2, Article 23 (March 2024), 32 pages. https://doi.org/10.1145/3624482

[19] Gyusun Lee, Seokha Shin, and Jinkyu Jeong. 2022. Efficient hybrid polling for ultra-low latency storage devices. *Journal of Systems Architecture* 122 (2022), 102338. https://doi.org/10.1016/j.sysarc.2021.102338

[20] Gyusun Lee, Seokha Shin, Wonsuk Song, Tae Jun Ham, Jae W. Lee, and Jinkyu Jeong. 2019. Asynchronous I/O Stack: A Low-latency Kernel I/O Stack for Ultra-Low Latency SSDs. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 603–616. https://www.usenix.org/conference/atc19/presentation/lee-gyusun

[21] Xiaojian Liao, Youyou Lu, Zhe Yang, and Jiwu Shu. 2021. Crash Consistent Non-Volatile Memory Express. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 132–146. https://doi.org/10.1145/3477132.3483592

[22] Linux man-pages project. [n.d.]. eventfd(2) - Linux manual page. https://man.archlinux.org/man/eventfd.2.en.

[23] Deborah T Marr, Frank Binns, David L Hill, Glenn Hinton, David A Koufaty, J Alan Miller, and Michael Upton. 2002. Hyper-Threading Technology Architecture and Microarchitecture. *Intel Technology Journal* 6, 1 (2002).

[24] Sohil Mehta. 2021. x86 user interrupts support. https://lwn.net/Articles/869140/.

[25] NVM Express, Inc. 2021. NVM Express Base Specification 2.0. https://nvmexpress.org/wp-content/uploads/NVM-Express-Base-Specification-2_0-2021.06.02-Ratified-5.pdf Ratified.

[26] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. 2014. SDF: software-defined flash for web-scale internet storage systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 471–484. https://doi.org/10.1145/2541940.2541959

[27] Venkatesh Pallipadi and Alexey Starikovskiy. 2006. The ondemand governor. In *Proceedings of the linux symposium*, Vol. 2. 215–230.

[28] Zebin Ren and Animesh Trivedi. 2023. Performance Characterization of Modern Storage Stacks: POSIX I/O, libaio, SPDK, and io_uring. In *Proceedings of the 3rd Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems* (Rome, Italy) *(CHEOPS '23)*. Association for Computing Machinery, New York, NY, USA, 35–45. https://doi.org/10.1145/3578353.3589545

[29] Yongju Song and Young Ik Eom. 2019. HyPI: Reducing CPU Consumption of the I/O Completion Method in High-Performance Storage Systems. In *International Conference on Ubiquitous Information Management and Communication*. https://api.semanticscholar.org/CorpusID:163164960

[30] SPDK. [n.d.]. SPDK: Changelog. https://spdk.io/doc/changelog.html.

[31] The SPDK Community. 2023. *SPDK NVMe Driver*. The Storage Performance Development Kit (SPDK) Project. https://spdk.io/doc/nvme.html

[32] The SPDK Community. 2024. *Storage Performance Development Kit (SPDK) Release v24.09*. https://github.com/spdk/spdk/releases/tag/v24.09

[33] Alex Williamson. 2011. VFIO - "Virtual Function I/O". The Linux Kernel documentation. https://docs.kernel.org/driver-api/vfio.html

[34] Jisoo Yang, Dave B. Minturn, and Frank Hady. 2012. When poll is better than interrupt. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (San Jose, CA) *(FAST'12)*. USENIX Association, USA, 3.

[35] Ziye Yang, James R. Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E. Paul. 2017. SPDK: A Development Kit to Build High Performance Storage Applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 154–161. https://doi.org/10.1109/CloudCom.2017.14

[36] Ltd. Yangtze Memory Technologies Co. [n.d.]. TiPro9000 Solid State Drive. Product Page. https://www.ymtc.com/cn/products/59.html?cat=44