Disperse Access Considered Energy Inefficient in Intel Optane DC Persistent Memory Servers

***Abstract*—The Intel Optane DC Persistent Memory Module (AEP), which is the first commercial available Non-Volatile Memory (NVM) product, offers comparable performance with DRAM while providing larger capacities and data persistence. Existing researches that substituting NVM with DRAM or hybridizing them are either emulator-based or focused on how to improve the energy efficiency for writes, while the energy efficiency of the real AEP system are less explored. Based on real AEP, we observe that, even thought eliminating the DRAM-like refresh energy consumptions, AEP shows significant energy consumption differences in different performance levels. Specifically, requests with time intervals (*disperse*) underperform in both performance and the energy efficiency when compared with requests without time intervals (*compact*). This disparity and parallelism performance potentials motivate us to propose Sprint-AEP, an energy-efficiency-oriented scheduling method for AEP-equipped servers. Sprint-AEP fully activates adequate AEPs to address most of the requests by deferring the write requests and prefetching the hottest data. The remaining AEPs will stay in idle mode with a low idle power to save energy. Besides, we also utilize the read parallelism to accelerate the sync and prefetching processes. Compared with energy-unaware AEP usages, our experimental results show that Sprint-AEP saves up to 26% energy without performance degradation.**

***Index Terms*—Intel Optane, Persistent Memory, Energy Efficiency, Scheduling**

I. Introduction

Putting all data into Dynamic Random Access Memories (DRAMs) is usually regarded as the last resort for storage-bottlenecked performance problems in the age when there are significant performance gaps between memory and storage devices. However, the exponential growth of application footprints challenges this DRAM-enabled method from the capacity, data persistence, and energy limitation. First, currently the JEDEC standard [1] allows for 128 GB as the maximum capacity per DRAM, while this upper bound is far from expected capacity demands. Second, data in DRAMs is volatile while the data persistence is usually considered as a default option for storage systems. This demand unavoidably exacerbates the disadvantage of DRAM capacity limits, because persisting data to or loading data from the relatively much slower non-volatile hierarchies significantly compromises the performance benefits of residing them in DRAMs [2]. Third, limited power budgets are unable to support the free scale of DRAMs, and as much as 40% of the total power consumed in a server is attributed to DRAMs [3–5]. Many researches are trying to address these issues in both micro and macro approaches.

Micro ways continue to explore potentials of DRAMs from their internals. Optimizing DRAM energy consumptions mitigates the energy limitations and also helps to scale DRAMs to provide larger capacities. These methods [6–13] take DRAM physical structures into considerations, and process all the data mainly by only a part of the DRAM banks while keeping other banks in the energy-efficient mode. Considering that DRAM on-board memory management units are usually black-boxed to users, these approaches are difficult to be realized. Besides, periodical refreshes for valid data in DRAMs still consume 13% of the total power even at idle state [5].

Macro tryings introduce Non-Volatile Memories (NVMs) as supplements to or substitutes for DRAMs. Apart from competitive performance with DRAM, NVM devices provides lower standby power with data persistence and, more importantly, higher storage capacities. These advantages of NVM greatly alleviate the large capacity and high performance demands from large-footprint applications, and make it a promising alternative to DRAM. However, previous publicly available NVM emulators or prototypes attract research efforts on problems such as limited write endurance [14–19] or the asymmetries of read and write operations in terms of performance or power consumptions [17–23]. The problem is that, for the newly released commercially available NVM device, Intel Optane DC Persistent Memory Module (AEP) [24], its practical characteristics remain unclear and are thus needs to be explored.

With different access patterns and lots of tests, we observe that (1) AEP shows significant energy consumption differences in different performance levels. Specifically, requests with time intervals (*disperse*) underperform in both performance and the energy efficiency when compared with requests without time intervals (*compact*). (2) AEP also shows great parallelism performance potentials. These findings indicate that fully activating only a part of AEPs not all of them in a server is able to handle the majority of requests in the most of time.

We believe that the real energy consumption of AEP, just like the power budget wall for DRAM, is the major concern for the further deployment, and focusing on its energy-efficiency in this paper.

This disparity and parallelism performance potentials motivate us to propose Sprint-AEP, an energy-efficiency-oriented scheduling method for AEP-equipped servers. Sprint-AEP reshapes access patterns to selected AEPs accordingly. Sprint-AEP fully powers on adequate AEPs to absorb the coming requests at the best effort, and dispatches misses to the remaining idle AEPs. Moreover, Sprint-AEP utilizes the parallelism potentials to compensate possible performance degradations for standby AEPs and to boost data movements among different power mode AEPs.

In summary, we make the following contributions in this paper:

- We conduct experiments on real AEP devices, and find opportunities to reduce energy consumptions for AEP-equipped servers.
- We take advantage of the parallelism performance potentials to provide performance guarantees.
- We implement Sprint-AEP on a server with real AEP devices. Experiment results show that Sprint-AEP saves up to 26% energy without performance degradation.

TABLE I
MOTIVATION TEST CONFIGURATIONS

CPU	2 x Gen Intel Xeon processors (48 cores)
OS Kernel	Linux 3.10.0-957.12.1.el7.x86_64 GNU/Linux
CPU0 DIMMs	One 16GB DRAM
CPU1 DIMMs	One 128GB AEP, and one 16GB DRAM
Access method	By-pass DRAM, NUMA with DAX AEP access

TABLE II
MEASURED AEP POWER

Power Mode	**Power in Watts (W)**
Idle	1.91
Peak	3.64
Dynamic	1.91-3.3

II. BACKGROUND AND MOTIVATION

In this section, we firstly brief the AEP product, and then give experiments on servers with real AEP devices under different access patterns to quantitatively show their enery and power characteristics. Finally, we explain our designs of Sprint-AEP in a high level to clarify design principles.

A. Intel Optane DC Persistent Memory Module (AEP)

The newly released commercially available Intel Optane DC Persistent Memory Module, named Apache Pass (AEP), provides DDR4 interfaces just the same as the DRAM, and currently a dual-processor Intel server can hold up to twelve AEPs. The typical capacity of an AEP is 128/256/512 GB, therefore a server provides up to 6 TB. The 3D XPoint technology [25] enabled AEP product offers near DRAM performance and data persistence, this greatly closing the persistency boundary between CPUs and storage levels. Besides performance parameters, however, the energy consumption and parallelism capabilities of AEP are less explored.

B. Access Pattern and AEP Power Mode

1) Experiment Configurations and Methods: Tab. I lists the parameters of motivation experiments on the AEP server, and we use the Running Average Power Level (RAPL) interface [26] to measure energy usages the DIMMs area [27]. To measure the AEP power, we set a test group and a control group in a two-processor server, which is shown in Tab. I. In detail, the CPU1 DIMMs area is set as the test group, and the CPU0 DIMMs area is set as the control group. The DRAM of both groups are the same, and the AEP only exits in the CPU1 DIMMs area. Through the NUMA and by-pass DRAM methods, the requests are sent to the AEP directly. By comparing the test group and the control group, we finally obtain the energy consumption of the AEP.

2) Workload Patterns: As there is usually a time interval between two requests in the real world [28], we plan to explore the influence of different time intervals on the AEP. To reduce the disturbance, the time interval between two requests is fixed during each test. And the request size is 4 KB in our tests, as 4 KB is a common size and shows the similar results with the other request sizes (bigger than 256 bytes).

We send requests with different time intervals to simulate the heavy and light workload access patterns. Specifically, requests with time intervals are *disperse* accesses, and requests without time intervals are *compact* accesses. We start from zero-time interval and run a new test every 0.1 millisecond (ms) until the 1.6 ms time interval with random/sequential write/read cases.

Tab. II lists the measured power of AEP in watts. When time intervals between requests are long enough, AEP shows a steady energy consumption in 1.91 W, which means that AEP requires a static power supply. As time intervals becomes shorter, AEP consumes more power, ranging from 1.91 to 3.3 W, and finally reaches its fully active mode in 3.64 W under compact accesses. These energy consumption differences indicate that the AEP energy consumption is tunable with different access patterns. The following subsection then quantitative the relationship between access time intervals and performance.

C. Performance and Energy-efficency

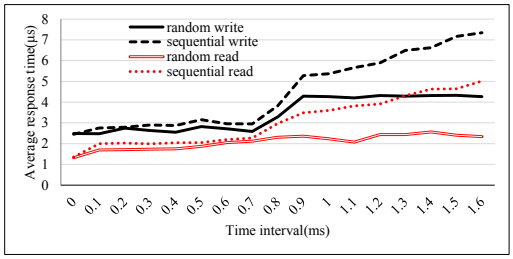


Fig. 1. The average response time of 4KB request under four R/W modes.

In Fig. 1, we can see that the average response time is closely related with the AEP power. The higher the average response time, the smaller the power of AEP. In general, the read outperforms the write. The smallest average response time is observed at zero-time interval. And the average response time changes little between o.1 ms and 0.8 ms time interval. Then the average response time reaches a higher value at the 0.9 ms time interval.

As shown in Fig. 2, for example, the average Input/Output Operations Per Second (IOPS) and AEP power under random write are introduced, as a function of the time interval. The left ordinate and right ordinate represent the IOPS and AEP power respectively. The IOPS of zero-time interval is 254131. In

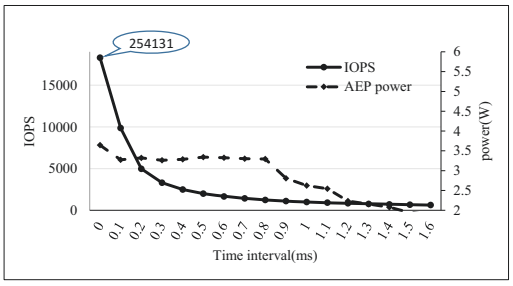


Fig. 2. IOPS and AEP power under random write.

general, IOPS decreases exponentially with the time interval. AEP power has the largest value at zero-time interval, and remains stable at 3.3 watt (W) between o.1 ms and 0.8 ms time interval. Finally, AEP power descends until 1.96 W, which is almost the same as idle power.

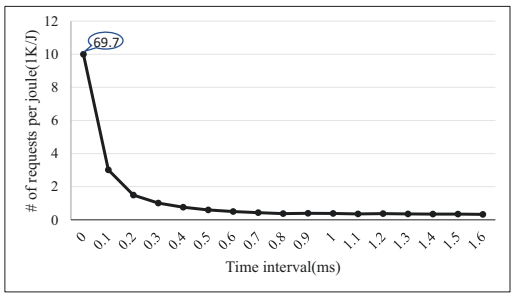


Fig. 3. Number of processed requests per joule.

As shown in Fig. 3, to figure out the efficiency, we analyze the number of requests the AEP can address with only one joule, as a function of the time interval. The value of zero-time interval (69.7) is too large to be displayed and is marked with a label. With the increase of time interval, the value descends quickly and then seems to be stable at 0. The smaller the time interval, the higher the efficiency.

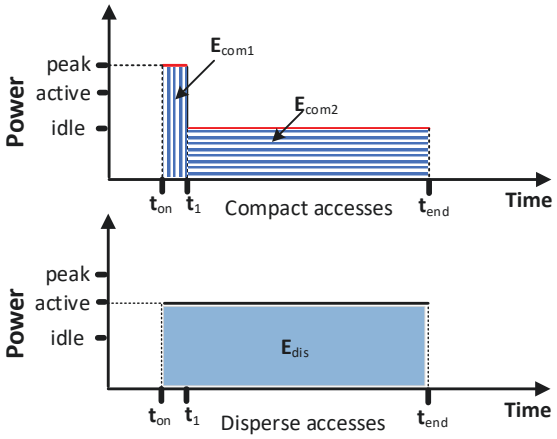


Fig. 4. Real-time power and total energy of compact and disperse accesses with the same amount of data. Compact accesses (top) and disperse accesses (bottom) over time.

In this article, we propose two concepts: compact and disperse requests. In detail, the requests with zero-time interval are named compact requests, and the requests with non-zero-time interval are named disperse requests. Fig. 4 shows the compact and disperse accesses with the same testing dataset size. Disperse accesses start with the active power at t_{on}, causing energy consumption to rise until time t_{end}, and finishes with the final energy consumption E_{dis}. On the other hand, compact accesses begin at t_{on} with the peak power, and remains until t_1, where the compact accesses have been finished with the execution-time energy consumption E_{com1}. Then the AEP becomes idle and the cumulative energy curve slowly rises during the idle period (from time t_1 to t_{end}), finally ending with the equal-time energy consumption E_{com1}+E_{com2}.

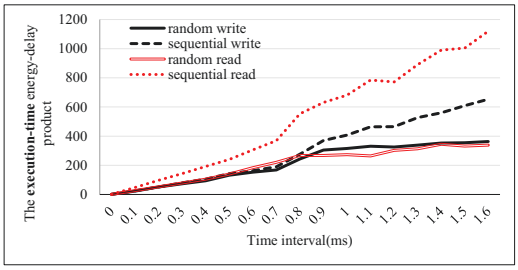


Fig. 5. The execution-time energy-delay product.

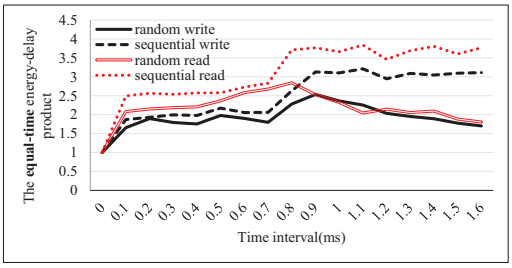


Fig. 6. The equal-time energy-delay product.

To further explore the energy efficiency, we use the **energy-delay product** as a measurement. In detail, the energy data means the **execution-time** energy and **equal-time** energy, the delay data means the average response time. All of the energy and delay value has been processed with normalization before used. As shown in Fig. 5 and Fig. 6, there are two kinds of products, which use the **execution-time** energy and **equal-time** energy respectively. We should note that the smaller the product, the higher the energy efficiency. Thus, in either case, the compact test (zero-time interval) has the highest energy efficiency.

D. Parallelism Potential

We have also tested the throughput and power of the AEP under different threads to explore the parallelism. Fig. 7 shows the throughput of AEP under different threads. With the increasing of threads, the throughput of read improves significantly. Whereas the throughput of write improves slightly and has a visible drop at eight threads. Fig. 8 shows the related AEP power. We can find that the read power is higher than the write power, which is the same as the throughput. In addition,

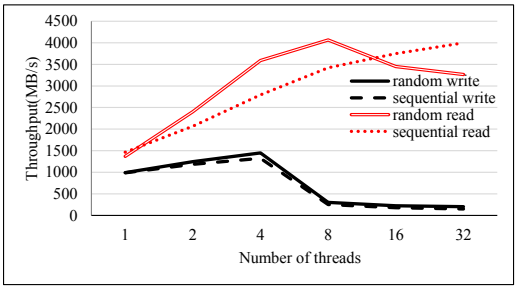


Fig. 7. Throughput of AEP with different threads.

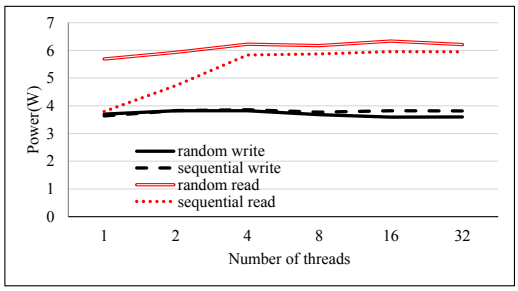


Fig. 8. AEP power with different threads.

the AEP power almost remains stable as the number of threads changes.

In short, read can benefit from multiple threads, while the write can't.

E. Motivation

Generally, part of the real-world workload traces are mainly composed of disperse requests (several tens of megabits per second), which can't fully utilize the bandwidth of an AEP. Meanwhile, even if the coming requests are all compact requests (zero-time interval), the bandwidth of the AEP server (multiple AEPs) still can't be fully utilized. But we should note that an AEP will run with high power even when dealing with disperse requests, rather than idle power. Therefore, naively distributing requests to all the AEPs is not energy efficient.

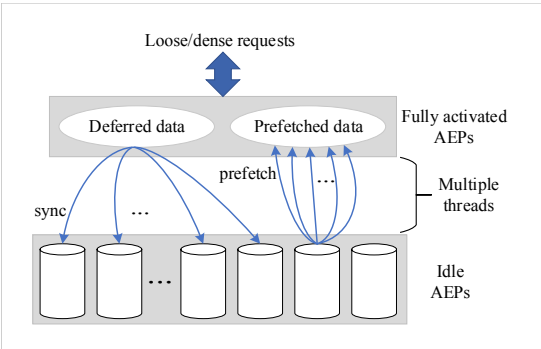


Fig. 9. The motivation of our design.

As shown in Fig. 9, to address this problem, we can fully activate only one AEP to address most of the requests by deferring the write requests and prefetching the read requests for the other AEPs. When one AEP is unable to deal with the compact requests, more AEPs will be fully activated in the AEP server. The remaining AEPs will stay in idle mode with a low idle power to save energy.

Finally, the deferred write requests will be synchronized to their corresponding idle AEPs, which offers an opportunity to leverage the internal parallelism of the AEP. In detail, as shown in section II, we can read the deferred write requests with multiple threads, which can significantly accelerate the sync process with only a small energy increase. And the parallelism can also be used in the prefetching process.

Though we can't keep only part of AEPs with idle power at present, the internal controller of each AEP designed by Intel offers an opportunity to achieve this goal in the future.

III. Design of Our System

A. System Architecture

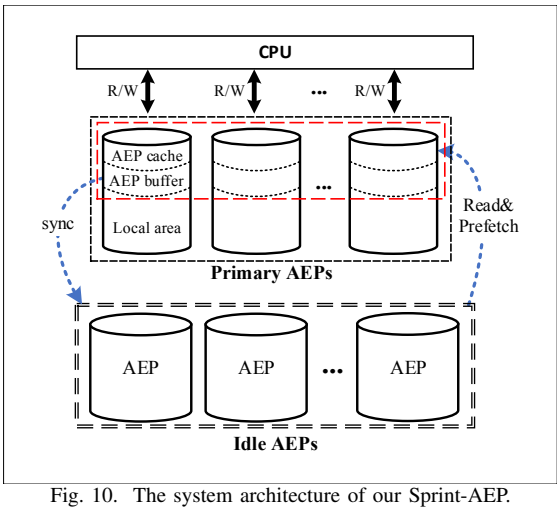


Fig. 10. The system architecture of our Sprint-AEP.

As shown in Fig. 10, the system architecture of our Sprint-AEP consists of three parts: CPU, primary AEPs, and idle AEPs.

CPU is mainly responsible for distributing requests across all the AEPs. All of the AEPs are divided into two parts: primary AEPs and idle AEPs. Primary AEPs are fully activated to guarantee the performance, and idle AEPs are always kept standby to save energy until there is a need to access an idle AEP.

There is at least one primary AEP, and the number of primary AEPs and idle AEPs varies with the intensity of workload. When the workload is too heavy for the existing primary AEPs, one idle AEP will be fully activated as a new primary AEP.

For convenience, we set a parameter α (10% in our tests) to judge whether an idle AEP should be fully activated. Firstly, we fully activate an idle AEP as primary AEP for several seconds, and calculate the rate of write and read performance increment. If the sum value of write and read throughput increment ratio can achieve α, this AEP will be kept as primary AEP.

Each primary AEP is responsible for up to M idle AEPs, which is set as 3 in our tests. Each primary AEP is logically divided into three parts: AEP cache, AEP buffer and local area.

The local area stores the data that is originally dispatched to the primary AEP. AEP buffer will absorb the write requests that are originally dispatched to idle AEPs, and data in AEP buffer will be synchronized to its corresponding idle AEPs. After data synchronization, these write data will be logically moved to AEP cache. In this way, the originally disperse write requests of idle AEPs will be gathered into compact write process, which can raise the energy efficiency. We fetch the frequently and recently accessed data from idle AEPs to AEP cache in advance, thus part of the originally disperse read requests will be turned into compact prefetching processes.

B. Data Organization

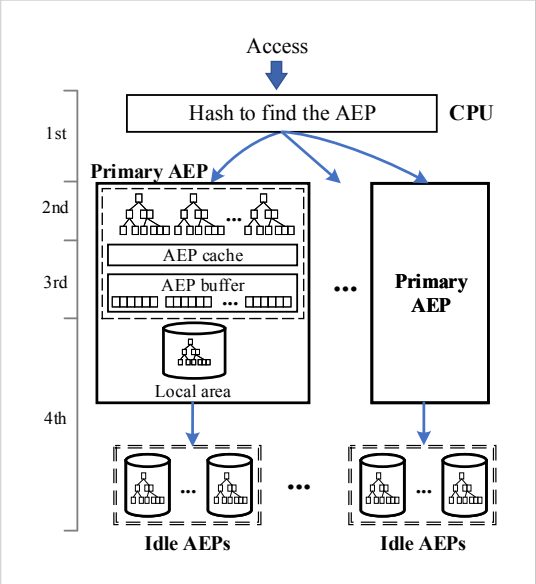


Fig. 11. The index architecture of the storage system.

For better data organization, we design an index architecture, which is shown in Fig. 11.

The first layer is a hash function, which is a function of the key for key-value requests and decides which AEP a request belongs to.

The second layer contains the B+-trees of AEP cache and AEP buffer of every primary AEP. Each B+-tree corresponding to one idle AEP and only manages the data that are temperately stored in primary AEP. There are only metadata in these B+-trees, and the related values are all stored in the AEP buffer and AEP cache.

The third layer contains the AEP buffer and AEP cache of every primary AEP. AEP cache stores the data that is most likely to be accessed, including the frequently and recently accessed data prefetched from its related idle AEPs and the clean data from AEP buffer. AEP buffer absorbs all the write requests of its related idle AEPs by many logs. Each log corresponding to one idle AEP.

The fourth layer is the final data storage area, including the idle AEPs and the local area of the primary AEPs. And each area is organized by an internal B+-tree.

When a request comes, it will be firstly processed by the hash function and then sent to one of the primary AEP. If the request is originally dispatched to one primary AEP, it will be directly addressed by the local area of the primary AEP. Otherwise, the request will visit the B+-trees of AEP cache and AEP buffer. If the request is a write process, it will be wrote into its corresponding log in the AEP buffer. Otherwise, we will read the data from the AEP cache, AEP buffer or the corresponding idle AEP. When an idle AEP is read, we will start a prefetching process, which is introduced in the next subsection.

Due to the limited space of AEP buffer, each log will be synchronized to its corresponding idle AEP when the log size reaches a threshold (80% of the AEP buffer in our test). As the read performance will increase with multiple threads, we will synchronize these logs with multiple threads to multiple idle AEPs. In this way, the sync process will be significantly accelerated.

Besides, we use key-value storage system in this article for an example, and the metadata is significant. The metadata of each key-value entry is designed as $(key, value\_address, pointer, hotness, flag)$. The key represents the logical address of the requests in our implementation, the $value\_address$ represents the physical address where the value is stored, the $pointer$ is reserved for prefetching data chain. The $hotness$ means the accessed popularity, the $flag$ means whether the data is dirty($flag$=1) or not($flag$=0) for the B+-trees of AEP cache and AEP buffer, and the $flag$ also means whether the data has been cached($flag$=1) or not($flag$=0) for B+-trees in idle AEPs. Note that B+-tree in the local area of primary AEP only needs $(key, value\_address, pointer)$.

C. Prefetching

To further reduce the energy consumption of idle AEPs, we propose an energy-aware prefetching algorithm. Since the write requests of idle AEPs can be temperately addressed by the AEP buffer, our main concern is about the read requests. We intend to design a prefetching algorithm to reduce the read access of idle AEPs.

This subsection provides an overview of our prefetching algorithm and a sketch showing how this algorithm works.

Very roughly, first, we will divide the data of AEP cache and idle AEPs into different hot levels respectively. The higher the hot level value, the larger the visit probability is. Thus, the hottest data in idle AEPs will be fetched to AEP cache when there is a primary AEP cache miss .

Fig. 12 shows the sketch of prefetching data from an idle AEP. At the beginning, we will initialize a Hot Range List (HRL) cluster based on the B+-tree of an idle AEP. We will also initialize a Cold Range List (CRL) cluster based on three B+-trees related to AEP cache and AEP buffer. The largest hot level value in HRL cluster and the smallest cold level value in CRL cluster are named $Hmax$ and $Hmin$ respectively.

To simplify the statistic, we impose certain limits on the HRL and CRL clusters. First, the smallest hot level value in HRL cluster should be larger than $Hmin$, and the largest cold level value in CRL cluster should be smaller than $Hmax$.

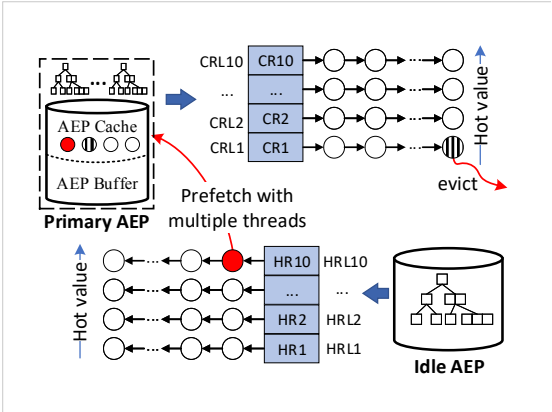


Fig. 12. The sketch of prefetching algorithm for an idle AEP.

Second, two clusters only records the metadata with $flag$ equals zero. To improve prefetching efficiency, we divide the HRL (CRL) cluster into several (up to 10) HRLs (CRLs). Every HRL (CRL) is constructed with LRU. The hot level value is sequentially incremented from HR1 (CR1) to HR10 (CR10).

When a primary AEP miss happens, we will fetch data from the corresponding idle AEP and a prefetching process will be triggered. First, data will be fetched with multiple threads from the head of HRL10 to AEP cache. If there isn't enough space in the AEP cache, data in CRL1 will be evicted. When HRL10 has been traversed, we will move on to the sequential chain HRL9 if the HR9 is still higher than CRL1, and so on. Otherwise, the prefetching process will be stopped. Besides, we should note that frequent and large data migration is likely to cause dramatic fluctuations. To this end, we set a threshold for the prefetching size at any one time, which is one-twentieth the size of AEP cache.

To guarantee the consistency of all the B+-trees, we record the metadata of the eliminative data, which will be synchronized to the corresponding idle AEPs when there is idle AEP access.

IV. Evaluation and Results

A. Configurations

TABLE III
THE CONFIGURATION OF OUR OBSERVATION TEST

CPU	2 processors with 96 cores Genuine Intel(R) CPU
System	Linux 3.10.0-957.12.1.el7.x86_64 GNU/Linux
CPU1 DIMMs	four 128GB AEPs, 16GB DRAM
Access method	By-pass DRAM, NUMA with DAX AEP access

We evaluate our Sprint-AEP on a real AEP server, which has been shown in Tab. III. This AEP server puts four 128GB AEPs on the CPU1 DIMMs, and we run our tests on CPU1 and four AEPs by NUMA, without accessing the DRAM.

The Yahoo! Cloud Serving Benchmark (YCSB) is an open-source specification and program suite for evaluating retrieval and maintenance capabilities of computer programs. It is often used to compare relative performance of NoSQL database management systems. In our tests, we use the C++ version of YCSB [29], which has low cost. Unless otherwise noted, the size of a key-value request is set as 1KB, the distribution mode of requests ($requestdistribution$) is set as $zipfian$, and we use 20 threads to distribute requests.

TABLE IV
THE WRITE RATIO OF OUR YCSB TRACES

Workload	Description	Time interval (μs)	Density
YCSB A	50% updates, 50% reads	0	high
YCSB B	5% updates, 95% reads	0	high
YCSB 3	67% updates, 33% reads	0	high
YCSB A-T	50% updates, 50% reads	50	low
YCSB B-T	5% updates, 95% reads	50	low
YCSB 3-T	67% updates, 33% reads	50	low

As shown in Tab. IV, we choose two representative YCSB workload traces: YCSB A and YCSB B, and make a new YCSB 3. The three traces represent three different write sensitivity. The time interval means the extra time we add between two requests. Therefore, three original traces (YCSB A, YCSB B, YCSB 3) are defined as high density traces with zero-time interval. Three traces (YCSB A-T, YCSB B-T, YCSB 3-T) are defined as low density traces with 50μs-time interval to simulate the real-word traces.

To evaluate the performance and energy consumption of our Sprint-AEP, we set three configurations for comparison.

- **Standard**: Four AEPs are all powered on as a small distributed key-value storage system to address requests without any power management policy.
- **Sprint-AEP**: Four AEPs are managed by our novel system scheduling scheme to save energy with little performance degradation.
- **Sprint-AEP w/o Pre**: Sprint-AEP method without prefetching algorithm.
- **Sprint-AEP w/o Parallel**: Sprint-AEP method without using the read parallelism in sync and prefetching.

B. Analysis on Low-density Workload

According to our records, Sprint-AEP only fully actives one primary AEP with three low-density traces (YCSB *-T), and the remaining three AEPs all mainly kept with idle power. Because the IOPS of YCSB *-T is far away from the maximum bandwidth of one AEP, and one primary AEP is enough.

1) Energy Consumption Analysis: As shown in Fig. 13 and Fig. 14, we record the number of read times and the execution time for one AEP (primary AEP for standard mode, idle AEP for Sprint-AEP) under different low-density YCSB *-T traces with 20GB data.

In Fig. 13, Compared to standard mode, it is clear that Sprint-AEP can significantly reduce the number of read times. Because AEP cache and AEP buffer act as a large read cache for the whole storage system in Sprint-AEP (w/o Pre). In detail, Sprint-AEP w/o Pre can reduce at least 96% of the number of read times, further more, Sprint-AEP can reduce about 99.99%. Sprint-AEP performs better than Sprint-AEP

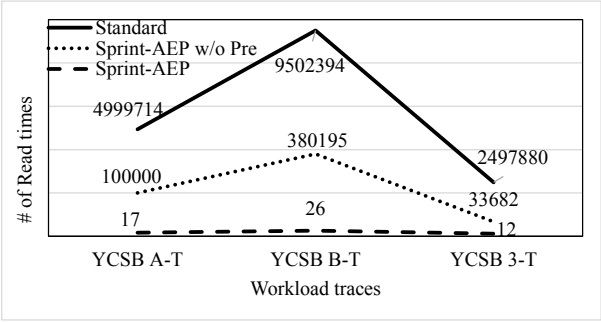
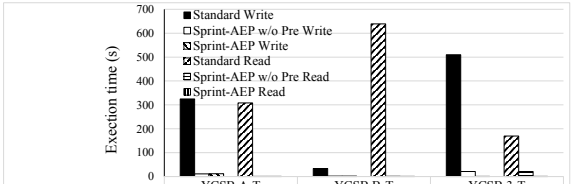


Fig. 13. Number of read times for one AEP (primary AEP for standard mode, idle AEP for Sprint-AEP) under different low-density YCSB *-T traces with 20GB data.



	YCSB A-T	YCSB B-T	YCSB 3-T
Standard Write	324.798	33.850714	509.772099
Sprint-AEP w/o Pre Write	11.2867	1.78421	21.0861
Sprint-AEP Write	11.4502	1.73962	0.114585
Standard Read	307.977	639.006192	169.526851
Sprint-AEP w/o Pre Read	0.312139	1.08313	20.7176
Sprint-AEP Read	0.292889	0.102834	0.273668

Fig. 14. Execution time for one AEP (primary AEP for standard mode, idle AEP for Sprint-AEP) under different low-density YCSB *-T traces with 20GB data.

w/o Pre, as more read requests of idle AEPs have been addressed by the primary AEP through prefetching.

In Fig. 14, compared to standard mode, Sprint-AEP reduces at least 95% of the execution time. Because most of the time has been wasted by the 50μs-time interval in the standard mode. Whereas, Sprint-AEP can address most of the disperse requests by only one fully activated primary AEP, and then gather these disperse requests into compact requests by sync and prefetching. Thus, the idle time of remaining AEPs are extended. The read time of Sprint-AEP is almost the same with Sprint-AEP w/o Pre, though there is a noticeable difference for the number of read times. Because the read time contains the prefetching time, and each prefetching operation fetches a chunk of data from the idle AEP. The pure read time of Sprint-AEP is smaller than Sprint-AEP w/o Pre, but the gap is offset by the time of prefetching.

According to Tab. II and Tab. IV, four AEPs of standard mode works at 2.8W all the time. For Sprint-AEP, when destage and prefetching happens, all of the four AEPs are kept in peak power (3.64W). The rest of the time, primary AEP works at 2.8W, and the remaining three idle AEPs work at 1.91W.

Therefore, as shown in Fig. 15, we calculate the total equal-time energy of four AEPs for different configurations under different low-density YCSB *-T traces with 20GB data. Compared to standard mode, Sprint-AEP (w/o Pre) can save about 26% energy. Sprint-AEP consumes almost the same

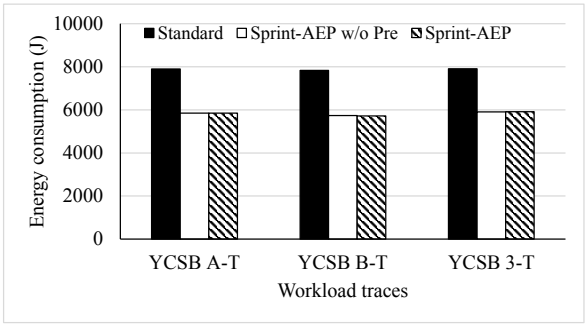


Fig. 15. Total equal-time energy of four AEPs for different configurations under different low-density YCSB *-T traces with 20GB data.

energy with Sprint-AEP w/o Pre, as they have the same read, write and idle time. There is no difference between different workload traces. Because lots of time has been wasted by the 50μ [illegible] for

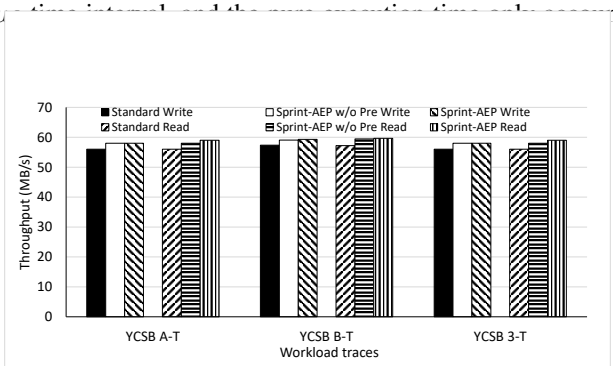


Fig. 16. Performance comparison under different low-density YCSB *-T traces with 20GB data.

2) Performance Analysis: As shown in Fig. 16, we test the performance of three configurations under different low-density YCSB *-T traces with 20GB data. Whether write or read, Sprint-AEP slightly outperforms the standard mode. The reason is that Sprint-AEP can easily gather disperse requests into compact requests to improve the performance by the primary AEP when the IOPS of YCSB *-T is far away from the maximum bandwidth of one AEP.

In detail, for write, Sprint-AEP (w/o Pre) absorbs disperse write requests by the primary AEP, and then synchronize these data to their related idle AEPs in compact mode by three parallel threads. For read in Sprint-AEP (w/o Pre), AEP cache and AEP buffer act as a large read cache for the whole storage system. Besides, for Sprint-AEP, disperse read requests can be turned into compact prefetching requests to further more increase the read performance. There is no difference between different workload traces. Because the pure execution time only accounts for a tiny fraction of the whole run time with the 50μs-time interval, and the throughput has been limited by the 50μs-time interval.

3) Energy Efficiency Analysis: To explore the energy efficiency, as described in section II, we use the **energy-delay product** as a measurement. The energy means the equal-time energy, and the delay means the sum of write and read latency. The final energy-delay products have been processed with

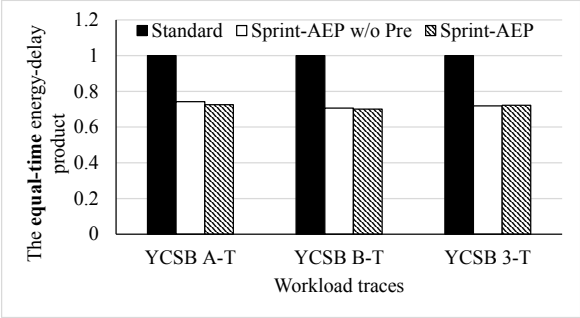


Fig. 17. Energy efficiency comparison under different low-density YCSB *-T traces with 20GB data.

normalization. As shown in Fig. 17, we compare the energy efficiency under different low-density YCSB *-T traces with 20GB data. The smaller the product, the higher the energy efficiency is. Therefore, Sprint-AEP (w/o Pre) has a higher energy efficiency than standard mode, and can improve the energy-delay product by 30%.

4) The Effect of Dataset Size: As shown in Fig. 18 and Fig. 19, we also calculate the total energy of four AEPs and record the performance for different configurations under YCSB A-T with different dataset size, which shows the same pattern.

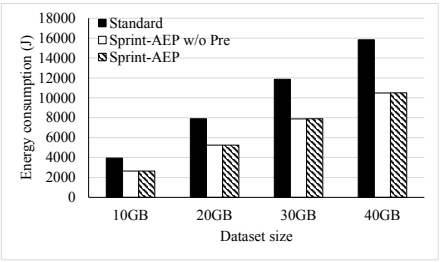


Fig. 18. Total equal-time energy of four AEPs under YCSB A-T with different dataset size.

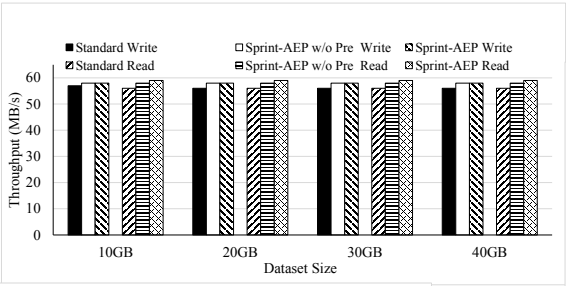


Fig. 19. Performance comparison under YCSB A-T with different dataset size.

C. Analysis on High-density Workload

Under high-density workload traces, Sprint-AEP (w/o Pre) fully activates three primary AEPs under YCSB A and B, two primary AEPs under YCSB 3. Because one fully activated AEP can't guarantee the performance, and the detail has been described in section III.

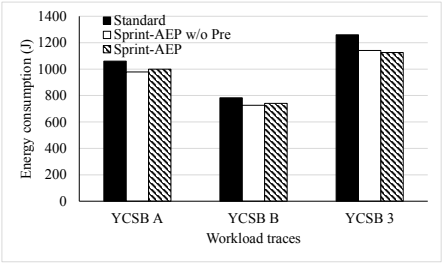


Fig. 20. Total equal-time energy of four AEPs for different configurations under different low-density YCSB *-T traces with 20GB data.

1) Energy Consumption Analysis: As shown in Fig. 20, we calculate the total energy of four AEPs of different configurations under different low-density YCSB *-T traces with 20GB data. Compared to standard mode, Sprint-AEP can save about 7% to 17% energy. Because Sprint-AEP (w/o Pre) fully activates three primary AEPs under YCSB A and B, two primary AEPs under YCSB 3. We can find that standard and Sprint-AEP (w/o Pre) consumes the least energy under YCSB B. Because the read ratio of YCSB B is the largest, and AEP performs a better read throughput than write.

2) Performance Analysis: As shown in Fig. 21, we compare the performance of different configurations under different high-density YCSB traces with 20GB data. We can find that the write throughput of standard mode is higher than Sprint-AEP (w/o Pre). Because the compact requests distributed to primary AEP has already reached its maximum bandwidth with B+tree, extra requests of an idle AEP will exceeds the maximum bandwidth and result in write performance degradation. Whereas, Sprint-AEP w/o Pre shows almost the same read throughput with standard mode, except for YCSB 3. Because the read ratio is smallest in YCSB 3, and the cache function of AEP cache and AEP buffer has been reduced by many write requests. But the read throughput is still largest under YCSB B, because of the largest read ratio. Sprint-AEP always shows a better read throughput than Sprint-AEP w/o Pre, as the prefetching algorithm really works.

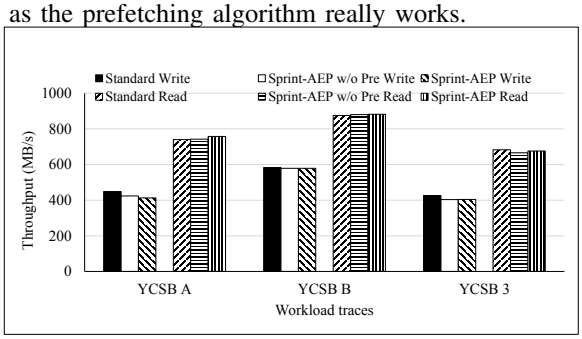


Fig. 21. Performance comparison under different high-density YCSB traces with 20 GB dataset.

To figure out why Sprint-AEP (w/o Pre) fully activates three primary AEPs under YCSB A and B, two primary AEPs under YCSB 3, Fig. 22 has been introduced. Fig. 22 shows the ratio of performance improvement under different high-density YCSB traces with 20 GB dataset. The X-X+1 Write/Read means the ratio of throughput improvement of Write/Read

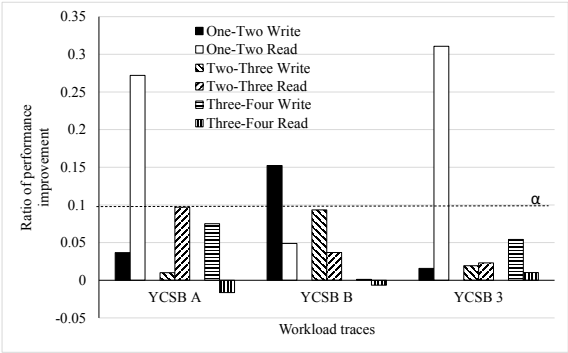


Fig. 22. Ratio of performance improvement under different high-density YCSB traces with 20 GB dataset.

when the system fully activates X+1 primary AEPs based on X primary AEPs. As shown in section III, we set a parameter α, which is 10% in our tests. If the sum of X-X+1 Write and X-X+1 Read can achieve α, then Sprint-AEP will fully activates X+1 primary AEPs. As shown in Fig. 22, Sprint-AEP will fully activates three primary AEPs for YCSB A and B, and two primary AEPs for YCSB 3.

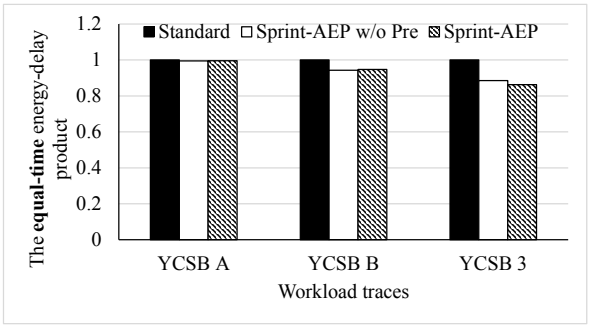


Fig. 23. Energy efficiency comparison under different high-density YCSB traces with 20GB data.

3) Energy Efficiency Analysis: To explore the energy efficiency, we still use the **energy-delay product** as a measurement. The energy means the equal-time energy, and the delay means the sum of write and read latency. The final energy-delay products have been processed with normalization. As shown in Fig. 23, we compare the energy efficiency under different high-density YCSB traces with 20GB data. The smaller the product, the higher the energy efficiency is. Therefore, Through little, Sprint-AEP (w/o Pre) still has a higher energy efficiency than standard mode.

D. The effect of Parallelism

To evaluate the effect of parallelism, we test Sprint-AEP and Sprint-AEP w/o Parallel under all the YCSB traces. First, as shown in Fig. 24, we evaluate the effect of parallelism under different high-density YCSB traces with 20GB data.

We can find that the utilization of parallelism can significantly improve the performance. In detail, Sprint-AEP outperforms 30% to 68% and 23.7% to 40% than Sprint-AEP w/o Parallel on write and read respectively. Because we read the AEP cache with multiple threads to accelerate the sync

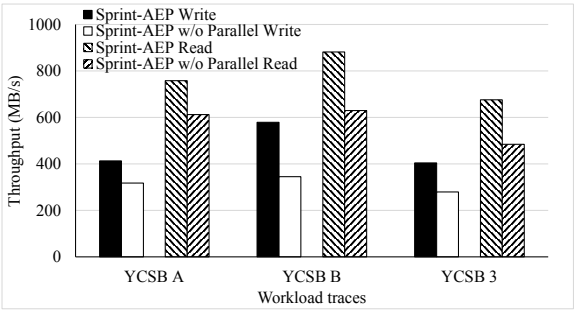


Fig. 24. The effect of parallelism under different high-density YCSB traces with 20GB data.

process with only a small increase in energy. And we read the hot data in idle AEPs with multiple threads to accelerate the prefetching process.

We also evaluate the effect of parallelism under different low-density YCSB traces with 20GB data, which is shown in Fig. 25. Different from the results under high-density YCSB traces, the parallelism can only bring little improvement on the performance. Because the throughput is mainly limited by the 50μs-time interval.

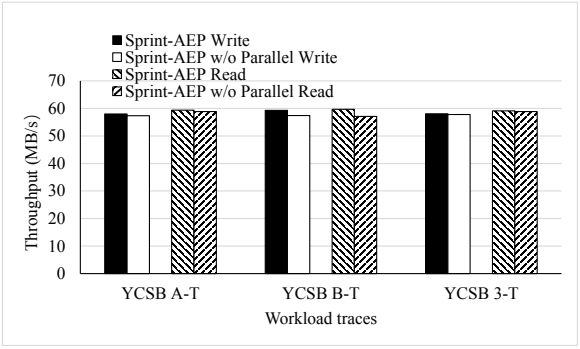


Fig. 25. The effect of parallelism under different low-density YCSB traces with 20GB data.

V. Related Work

A. Energy Saving Methods of DRAM

To reduce the energy consumption of DRAM, many methods have been proposed. For example, turning some banks of DRAM into low-power mode according to the request interval [6, 7], processing all of the data mainly by part of the DRAM [8–10], reducing the DRAM refresh rate [11, 12], reducing the activation energy by scheduling the order of requests [13], and so on [30, 31]. Although there are many energy-saving schemes for DRAM in academic, the majority of them do not seem to actually promote industrialization. Because the chip/bank of a DRAM is difficult to control. And the data reliability and capacity of DRAM are always problems[1].

B. Energy Saving Methods of NVM

The energy-saving methods of NVM usually save energy by organizing the data allocation of a hybrid DRAM/NVM memory [14–19, 32, 33], and some methods mainly focus on the consistence and performance, not the energy. Article [17]

proposes energy-aware persistence principles, save energy by modifying the logging method of a single NVM device. Article [32] saves energy by performing data allocation to different memories and task mapping to different cores. Article [33] uses DRAM to store the data with higher local I/O activity, while cache data with lower activity is placed into NVM to save the refresh energy. Most of the methods only focus on the high write energy and latency, and the read energy is always ignored.

C. Systems and products of NVM

The excellent characteristics of NVM have attracted extensive interests of industry and academia [20]. Academic researchers have proposed systems, such as SOFORT [21], Peloton [34], and FOEDUS [35], that incorporate NVM both as memory and storage. Meanwhile, database vendors like SAP, Oracle, Microsoft, and Aerospike, have raced to announce NVM-based products [22, 23, 36, 37]. However, these systems and products do not solve the energy consumption problems of NVM.

VI. Conclusion

In this paper, we observe that naively scattering data to all AEPs is energy inefficient, and an AEP can obtain the highest energy efficiency when dealing with compact requests. Therefore, we propose an energy-efficient AEP scheduling method, named Sprint-AEP. Sprint-AEP fully activates only part of the AEPs to absorb all the requests, and dispatches misses to the remaining idle AEPs. Thus, the write requests of idle AEPs will be temperately stored in the fully activated AEPs, and finally synchronized to idle AEPs in compact mode. And by a compact prefetching algorithm for idle AEPs, most of the read requests can be addressed by the fully activated AEPs. Therefore, all the write requests and most of the read requests of idle AEPs can be gathered into compact requests for better energy efficiency. Compared with naive AEPs usage, our experimental results show that Sprint-AEP saves up to 26% energy without performance degradation.

References

[1] (2020, Jan.) Jedec. [Online]. Available: https://www.jedec.org/

[2] J. Xu, J. Kim, A. Memaripour, and S. Swanson, "Finding and fixing performance pathologies in persistent memory software stacks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, 2019, pp. 427–439.

[3] C. Lefurgy, K. Rajamani, F. L. R. III, W. M. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *IEEE Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[4] M. S. Ware, K. Rajamani, M. S. Floyd, B. Brock, J. C. Rubio, F. L. R. III, and J. B. Carter, "Architecting for power management: The IBM power7$^{\text{tm}}$ approach," in *16th International Conference on High-Performance Computer Architecture (HPCA-16 2010), 9-14 January 2010, Bangalore, India*, 2010, pp. 1–11.

[5] B. Oh, N. Abeyratne, J. Ahn, R. G. Dreslinski, and T. N. Mudge, "Enhancing DRAM self-refresh for idle power reduction," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design, ISLPED 2016, San Francisco Airport, CA, USA, August 08 - 10, 2016*, 2016, pp. 254–259.

[6] V. Delaluz, M. T. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "DRAM energy management using software and hardware directed power mode control," in *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA'01), Nuevo Leone, Mexico, January 20-24, 2001*, 2001, pp. 159–169.

[7] X. Fan, C. S. Ellis, and A. R. Lebeck, "Memory controller policies for DRAM power management," in *Proceedings of the 2001 International Symposium on Low Power Electronics and Design, 2001, Huntington Beach, California, USA, 2001*, 2001, pp. 129–134.

[8] V. D. L. Luz, M. T. Kandemir, and I. Kolcu, "Automatic data migration for reducing energy consumption in multi-bank memory systems," in *Proceedings of the 39th Design Automation Conference, DAC 2002, New Orleans, LA, USA, June 10-14, 2002*, 2002, pp. 213–218.

[9] V. Delaluz, A. Sivasubramaniam, M. T. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Scheduler-based DRAM energy management," in *Proceedings of the 39th Design Automation Conference, DAC 2002, New Orleans, LA, USA, June 10-14, 2002*, 2002, pp. 697–702.

[10] H. Huang, K. G. Shin, C. Lefurgy, and T. W. Keller, "Improving energy efficiency by making DRAM less randomly accessed," in *Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005, San Diego, California, USA, August 8-10, 2005*, 2005, pp. 393–398.

[11] M. Ghosh and H. S. Lee, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked drams," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40 2007), 1-5 December 2007, Chicago, Illinois, USA*, 2007, pp. 134–145.

[12] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: retention-aware intelligent DRAM refresh," in *39th International Symposium on Computer Architecture (ISCA 2012), June 9-13, 2012, Portland, OR, USA*, 2012, pp. 1–12.

[13] D. H. Yoon, M. K. Jeong, and M. Erez, "Adaptive granularity memory systems: a tradeoff between storage efficiency and throughput," in *38th International Symposium on Computer Architecture (ISCA 2011), June 4-8, 2011, San Jose, CA, USA*, 2011, pp. 295–306.

[14] S. Pelley, P. M. Chen, and T. F. Wenisch, "Memory persistency," in *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN,*

USA, June 14-18, 2014, 2014, pp. 265–276.

[15] D. Narayanan and O. Hodson, "Whole-system persistence," in *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2012, London, UK, March 3-7, 2012*, 2012, pp. 401–410.

[16] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *36th International Symposium on Computer Architecture (ISCA 2009), June 20-24, 2009, Austin, TX, USA*, 2009, pp. 2–13.

[17] S. Kannan, M. K. Qureshi, A. Gavrilovska, and K. Schwan, "Energy aware persistence: Reducing energy overheads of memory-based persistence in nvms," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT 2016, Haifa, Israel, September 11-15, 2016*, 2016, pp. 165–177.

[18] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos, "Software-managed energy-efficient hybrid DRAM/NVM main memory," in *Proceedings of the 12th ACM International Conference on Computing Frontiers, CF'15, Ischia, Italy, May 18-21, 2015*, 2015, pp. 23:1–23:8.

[19] S. Lee, H. Bahn, and S. H. Noh, "Characterizing memory write references for efficient management of hybrid PCM and DRAM memory," in *MASCOTS 2011, 19th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Singapore, 25-27 July, 2011*, 2011, pp. 168–175.

[20] G. Psaropoulos, I. Oukid, T. Legler, N. May, and A. Ailamaki, "Bridging the latency gap between NVM and DRAM for latency-bound operations," in *Proceedings of the 15th International Workshop on Data Management on New Hardware, DaMoN 2019, Amsterdam, The Netherlands, 1 July 2019*, 2019, pp. 13:1–13:8.

[21] I. Oukid, D. Booss, W. Lehner, P. Bumbulis, and T. Willhalm, "SOFORT: a hybrid SCM-DRAM storage engine for fast data recovery," in *Tenth International Workshop on Data Management on New Hardware, DaMoN 2014, Snowbird, UT, USA, June 23, 2014*, 2014, pp. 8:1–8:7.

[22] M. Andrei, C. Lemke, G. Radestock, R. Schulze, C. Thiel, R. Blanco, A. Meghlan, M. Sharique, S. Seifert, S. Vishnoi, D. Booss, T. Peh, I. Schreter, W. Thesing, M. Wagle, and T. Willhalm, "SAP HANA adoption of non-volatile memory," *PVLDB*, vol. 10, no. 12, pp. 1754–1765, 2017.

[23] (2019, Nov.) Oracle timesten in-memory database. [Online]. Available: https://www.oracle.com/database/technologies/related/timesten.html

[24] (2019, Nov.) Intel optane dc persistent memory. [Online]. Available: www.intel.com/optanedcpersistentmemory

[25] (2019, Nov.) 3d_xpoint. [Online]. Available: https://en.wikipedia.org/wiki/3D_XPoint

[26] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: memory power estimation and capping," in *Proceedings of the 2010 International Symposium on Low Power Electronics and Design, 2010, Austin, Texas, USA, August 18-20, 2010*, 2010, pp. 189–194.

[27] S. Desrochers, C. Paradis, and V. M. Weaver, "A validation of DRAM RAPL power measurements," in *Proceedings of the Second International Symposium on Memory Systems, MEMSYS 2016, Alexandria, VA, USA, October 3-6, 2016*, 2016, pp. 455–470.

[28] (2020, Jan.) Real-world traces. [Online]. Available: http://iotta.snia.org/traces/

[29] (2018, Apr.) A c++ version of ycsb. [Online]. Available: https://github.com/basicthinker/YCSB-C

[30] O. Ozturk and M. T. Kandemir, "Ilp-based energy minimization techniques for banked memories," *ACM Trans. Design Autom. Electr. Syst.*, vol. 13, no. 3, pp. 50:1–50:40, 2008.

[31] Y. Li and T. Zhang, "Reducing DRAM image data access energy consumption in video processing," *IEEE Trans. Multimedia*, vol. 14, no. 2, pp. 303–313, 2012.

[32] Y. Wang, K. Li, J. Zhang, and K. Li, "Energy optimization for data allocation with hybrid SRAM+NVM SPM," *IEEE Trans. on Circuits and Systems*, vol. 65-I, no. 1, pp. 307–318, 2018.

[33] B. Wang, J. Tang, R. Zhang, W. Ding, S. Liu, and D. Qi, "Energy-efficient data caching framework for spark in hybrid DRAM/NVM memory architectures," in *21st IEEE International Conference on High Performance Computing and Communications; 17th IEEE International Conference on Smart City; 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019, Zhangjiajie, China, August 10-12, 2019*, 2019, pp. 305–312.

[34] J. Arulraj, M. Perron, and A. Pavlo, "Write-behind logging," *PVLDB*, vol. 10, no. 4, pp. 337–348, 2016.

[35] H. Kimura, "FOEDUS: OLTP engine for a thousand cores and NVRAM," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, 2015, pp. 691–706.

[36] (2019, Apr.) How to configure persistent memory (pmem). [Online]. Available: https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-pmem?view=sqlallproducts-allversions

[37] B. Bulkowski. (2018, Dec.) Aerospike 4.5: Persistent memory and compression. [Online]. Available: https://www.aerospike.com/blog/aerospike-4-5-persistent-memory-compression/