Dynamic Interval Polling and Pipelined Post I/O Processing for Low-Latency Storage Class Memory

Dong In Shin, †Young Jin Yu, †Hyeong S. Kim, Jae Woo Choi,
Do Yung Jung, †Heon Y. Yeom
Taejin Infotech, †Seoul National University, Korea

Abstract

Emerging non-volatile memory technologies as a disk drive replacement raise some issues of software stack and interfaces, which have not been considered in disk-based storage systems. In this work, we present new cooperative schemes including software and hardware to address performance issues with deploying storage-class memory technologies as a storage device. First, we propose a new polling scheme called *dynamic interval polling* to avoid the unnecessary polls and reduce the burden on storage system bus. Second, we propose a pipelined execution between storage device and host OS called *pipelined post I/O processing*. By extending vendor-specific I/O interfaces between software and hardware, we can improve the responsiveness of I/O requests with no sacrifice of throughput.

1 Introduction

The widening performance gap between main memory and storage devices over the past decades has limited the ability to improve overall performance of large-scale, high-performance computers. Several non-volatile memory technologies complement this performance gap between memory and storage. Deployment of storage-class memory (SCM) as a disk drive replacement and a scalable main memory alternative requires analyzing its impact on overall computer systems including the existing software stack and interfaces.

Communicating with devices via polling is known to be more efficient than interrupt-based approach in the low-latency devices including storage-class memory (SCM) as well as high-speed network. When the interrupt-based processing is used for the thousands of packets or requests per second, the overall system performance degrades significantly due to the interrupt overhead. Spinning reduces this delay significantly by busy-waiting rather than sleeping, which removes the context switch overhead and the associated delay [1, 3, 12].

Several new issues which have not been considered before arise when applying polling-based approach to the low-latency storage-class memory. The first thing is a polling interval. At first sight, shorter polling interval seems to improve the responsiveness. However, since device polling is a process which reads from I/O memory register, this leads to data transfer between host memory and the device register. Therefore, a short interval between polls imposes a heavy burden on the storage channel. Naive fixed interval polling can degrade I/O performance resulting in poorer performance compared with interrupt based approach. Another thing we consider is a shortened gap between memory and a storage device. Memory access time, which was a negligible fraction in disk-based storage system, has become substantial in the overall I/O performance for the low-latency SCM device whose response time is only a few microseconds.

In this paper, we present our collaborative work in the level of software and hardware to address some performance issues *with* deploying the advanced memory technology as a storage device. First, we studied the relationship between the polling interval and the responsiveness on the high-performance storage channel like PCI-Express. We propose a new polling scheme called **dynamic interval polling**, which adjusts the polling interval dynamically based on the response time prediction of the SCM device. By using dynamic interval polling, we can avoid unnecessary polls and thus reduce the burden on storage system bus.

Next, we reviewed the software storage stack considering the shortened gap of delay between memory and the new storage device. In modern disk drives, it is well-known that a request coalescing scheme can improve the device throughput significantly. However, this batch-style I/O degrades request latency due to the host-device interfaces: A single completion ack via hardware interrupt is given only after all merged requests are completed. The coalesced signal also makes the mem-

ory operations for each completed request to be serialized. Those memory operations include unmapping DMA memory, updating memory to notify I/O completion, and freeing memory. We call those memory operations as post-I/O processing. Compared to the low-latency of the SCM devices, the serialized memory operation time becomes no longer a negligible amount.

To reduce the latency under the coalescing scheme with little or no sacrifice of throughput, we propose **pipelined post I/O processing**, which enables both an I/O processing in storage device and the following post memory operations in host OS to be executed simultaneously. For this, we propose a new I/O interfaces called **page-unit I/O completion acknowledgement**. This enables the device controller to notify an I/O completion per page via a special I/O memory register rather than sends an I/O completion event after all merged requests are handled. Our polling-based device driver is also extended to monitor the completion of each page segment and execute the memory operations for the page even before the I/O requests is completely serviced.

We evaluate I/O performance using our DRAM SSD which emulates the next generation phase change memory. We run FIO benchmark under multi-threaded configurations to evaluate the overall performance improvements. The results show that the proposed solutions improve IOPS about 20% for 4 KB block read, corresponding to 50,000 IOPS. To evaluate the performance gain on full workloads with realistic mixed I/O, we measure OLTP performance using BenchmarkSQL tool over PostgreSQL database. Our approach increases mixed read/write performance in terms of tpmC by 15% and achieves much higher peak throughput at higher load.

As our contribution, we expect that new I/O interfaces as well as the proposed software techniques will lead to better design of future host controller interfaces and drivers [6]. In the following section, we give a brief explanation about the motivation for our work. Next, we explain a dynamic interval polling technique. In section 4, we propose a pipelined post I/O processing scheme using a page-unit I/O completion acknowledge as a new block I/O interface. Performance evaluation is presented in section 5.

2 Motivation

Deploying new memory technology as a storage device involves new strategies of design for the software and I/O interfaces. Communicating with low-latency devices via polling is one of the strategies proven to be more efficient than the existing interrupt-based approach. In this section, we focus on new features of storage class memory different from disk-based storage.

One of major differences between SCM device and disk storage is how to access the media to store and fetch data. By the electrical signals similar to load and store in main memory, data access in SCM does not involve any physical movement in disk-based storage including the head seeking and the platter rotating. Since data access latency in SCM is 3-4 orders of magnitude less than that in disk-based storage, storage using SCM can significantly narrow the performance gap with DRAM.

Another important feature of SCM is a predictability of response time. Due to its mechanical nature, the hard disk has a non-uniform access latency, which make it hard to predict its response time precisely. On the contrary, SCM itself has a uniform access latency, which is independent of access pattern [5]. There are some issues related to predictability in SCM performance, which arise from employing SCM as an I/O device; queueing, buffering, overwriting, and array architecture [4]. Although these issues contribute unpredictability in the delay, some new features on memory access can make SCM more predictable storage device than others. For example, in phase-change memory (PCM), a leading candidate among SCM technologies as a disk drive replacement as well as a scalable DRAM alternative, the memory element can be switched more quickly and at the level of single bits without needing to first erase an entire block of cells. Due to the fact that PCM allows overwriting and requires no garbage collection, unlike flash SSD, this allows for a much simpler wear-leveling techniques, which can improve predictability in SCM performance [9, 10].

Our analysis for SCM features raises some questions to the existing storage system stack as follows: Could we predict the completion events and avoid the unnecessary polling rather than by simply spinning? Would there be any inefficient execution in software storage stack which has a significant impact on I/O performance under a shortened gap between memory and SCM device? In next sections, we present our collaborative work in the level of software and hardware to address these questions.

3 Dynamic Interval Polling

3.1 Fixed Interval Polling

Device polling is more efficient than interrupt in the low-latency storage device like phase-change memory. When applying polling-based approach to the low-latency device, we found that polling interval is directly related to the response time of I/O requests. At first, we simply believed that the shorter polling interval would ensure the lower response time. However, experimental result was against our expectation as depicted in Fig 1. The main reason is that device polling itself is a process which

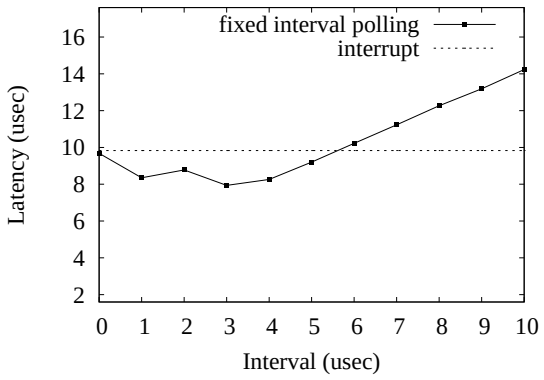

Figure 1: SCM device latency with increased polling interval for 4 KB read request. We measured the latency using our DRAM SSD which serves as a model of future SCMs.

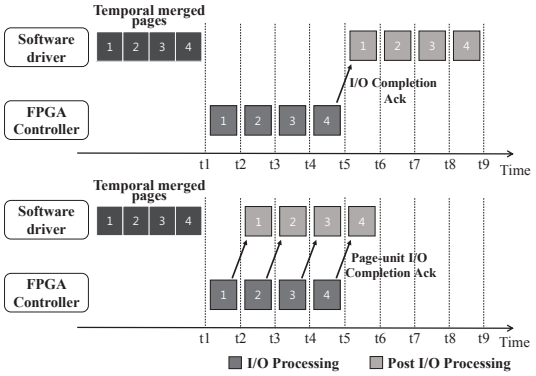

Figure 2: The pipelined post I/O processing using page-unit I/O completion ack

reads from I/O memory register. In Linux, for example, this is done by ioread32() system call which incurs DMA transfer from the device register to the host memory. Therefore, frequent polling may incur contention with the transferred data between them. As a result, a very short interval between polls imposes a heavy burden on the storage channel. Naive fixed interval polling can even degrade I/O performance resulting in inferior performance compared with interrupt based approach as shown in Fig. 1.

3.2 Dynamic Interval Polling using Response Time Prediction

To address the problem of the fixed interval polling, we propose a new polling scheme called *dynamic interval polling*. By adjusting the intervals between polls, we reduce the unnecessary polling and avoid the channel contention with the transferred data between host and device. As a result, the device responsiveness can be improved over the naive fixed interval polling.

To apply our polling scheme to an existing polling-based software driver, we divided the working cycle of the driver into two phases as follows. During an initial sampling phase, the driver runs in the fixed interval polling mode; it measures the device response times and classifies the measured times according to request type and size. After the sampling phase, the driver runs in the dynamic interval polling mode. According to the type and size of the issued I/O request, the driver calculates the predicted finish time and waits without polling until the time has come. No context switch occurs during polling.

To make this polling scheme feasible, it is crucial to achieve a highly accurate prediction. For this, we use a nano-second resolution timer in modern Linux kernel, which is used to both measure the delay and correct the prediction error. We use a feedback loop-based polling approach. If the estimated poll is earlier than the device notification, we perform polls recursively with a very short interval as much as a few dozens of nano-seconds α. Then, we correct the estimated delay as much as the margin of error. On the contrary, when the estimated poll detects the I/O completion at a time, we decrease the estimation value as much as α.

4 Pipelined Post I/O Processing

We reviewed the existing software storage stack considering the shortened gap between memory and the SCM device. Disk performance is measured by bandwidth and access time. Device polling as new interfaces of new storage device achieved a significant performance gain in terms of the latency [12]. To improve the bandwidth, we proposed new software merging scheme called **temporal merge** in our previous work [13]. This combines multiple block requests into one I/O request regardless of their spatial adjacency in disk address space.

We found that temporal merge improves throughput significantly within SCM device as the existing back/front merging scheme within disk-based storages. However, these I/O batching schemes limit the request latency due to the interfaces between host and storage device. To reduce the interrupt overhead, disk drive can withhold the interrupt with the completed messages until it gathers many of them to send at once [7]. Request latency is increased as much as the additional delay incurred by waiting for the interrupt. Similarly, for the temporally merged pages, a completion message is given only after their I/O is completed.

Besides, the memory operations for each completed

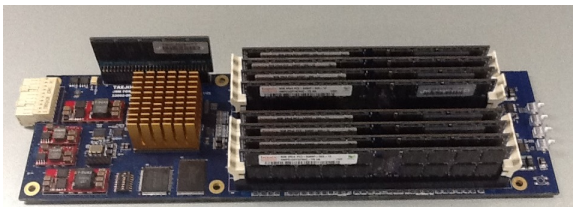

Figure 3: Our DRAM-based SSD Jetspeed^{TM}hardware used to emulate PCM device connected with PCI Express Gen2 x4 channel

request are serialized after waiting for all merged requests to be done. These memory operations include freeing memory regions mapped for DMA transfer, updating memory regions to notify I/O completion, and freeing memory regions allocated for I/O and the coalescing. We call these memory operations after I/O as a post I/O processing. Due to the shortened performance gap between memory and SCM device, the serialized memory access time becomes no longer a negligible fraction.

We present our cooperative work in the level of software and SCM hardware called **pipelined post I/O processing**. This allows the I/O processing in the device and the post I/O processing in the host run in parallel. As a result, this can reduce the latency of each request for the batched I/O with no or little sacrifice of bandwidth. To enable this pipelined execution, we propose new I/O interfaces called **page-unit I/O completion acknowledge**. This allows the device controller notify an I/O completion per page rather than send an I/O completion event after all merged requests are handled. For this, we extended our vendor-specific I/O interfaces and added a special I/O memory register to support the page-unit I/O completion acknowledge.

Fig. 2 exemplifies our scheme applied to handling temporal merged requests. We can apply this scheme to the general merging with spatial adjacency and the request queuing likewise. The two timelines show an I/O completion model which signals when all page segments are handled (the top timeline) and each page segment is handled using the page-unit I/O completion ack (the bottom timeline) respectively. Page 1 in the top spends as much as $(t5-t1)$ for its device latency, whereas the page in the bottom spends only $(t2-t1)$ for this. For all the pages in the figure, we obtain the performance gain in terms of the device latency as much as $6T$ if the I/O processing time for a single page is T.

5 Performance Evaluation

We implemented our schemes in our vendor-specific DRAM-based SSD as shown in Fig. 3 and device driver

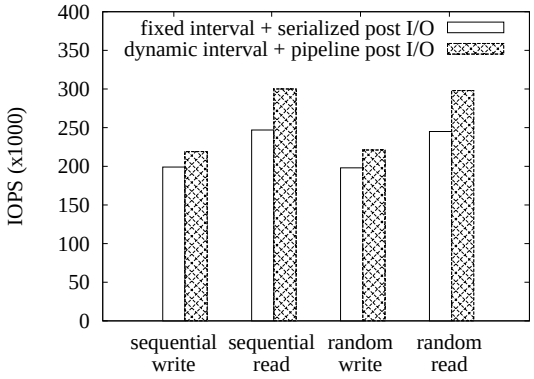

Figure 4: Scaling of storage I/Os per second (IOPS) with dynamic interval polling and pipelined post I/O processing schemes. We run fio benchmark with 16 threads.

running on a Linux 2.6.32 kernel. To emulate the future SCM storage device based on phase-change memory on our DRAM SSD, we imposed delay time as much as the performance gap between DRAM and PCM at the FPGA level as described in [8]. PCM overwrite (RESET of the cell is required before SET) is distinguished from clean write through a bitmap maintained in a device driver and PCM cell state is propagated via our customized host-device interfaces. We used PCI Express Gen2 x4 as our storage channel. Other issues on PCM including array architecture, wearing-out effect and queueing with a specific priority rule (we used FIFO command queue) are beyond of this paper. The target machine has two Xeon E5630 2.53 GHz quad core CPUs (total 16 cores) and 16 GB of RAM as it main memory.

We used the FIO benchmark to measure the performance impact of our schemes in terms of I/Os per second (IOPS). We ran the benchmark performing sequential read/write, random read/write under 16 threads and synchronous I/O configurations. Figure 4 shows the fio results in terms of IOPS. We compare our new schemes, dynamic interval polling and pipelined post I/O processing, to the existing fixed interval polling and serialized post I/O processing. Temporal merging approach is applied to both cases. We set one microsecond as its polling interval in the fixed interval polling scheme; The best performance was obtained by this polling interval. Both schemes we propose result in higher IOPS than the existing scheme as much as 10% in sequential and random write cases, corresponding to about 20,000 IOPS. The graph shows that the performance gain is doubled in read cases, corresponding to about 50,000 IOPS. The different rates of improvement results from that freeing memory regions mapped for DMA transfer and allocated for temporal merge in our device driver spend more CPU time handling read requests than write requests. In or-

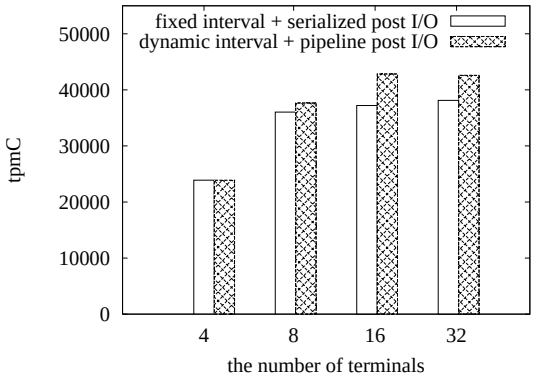

Figure 5: BenchmarkSQL OLTP performance result.

der to evaluate the effect of the page-unit I/O completion ack on bandwidth, we measure the number of pages in a coalesced I/O request. The number of those pages is increased from 6.4 to 7.5 in average. Our experimental results show our schemes improve the responsiveness of I/O requests and device throughput totally.

We used BenchmarkSQL tool, a TPC-C like benchmark, to evaluate the performance gain of our approach on full workloads with more realistic mixed I/O. We ran the benchmark on PostgreSQL database and utilized Linux EXT3 as its underlying filesystem. The graph in Fig. 5 shows the scaled tpmC with the increasing number of terminals. We find that our solutions improves the overall performance by 15% and achieves much higher peak throughput (up to 43,000 tpmC) at more higher load. However, there is no performance benefit at lower load (four terminals in the graph) due to little or no chance of temporal merge. The improved throughput on highly parallel workloads is a result of both the increased number of temporally coalesced I/O requests by pipelined memory operations and the shortened response time by the new polling scheme.

6 Conclusion and Future work

In this paper, our cooperative work explores software storage stack optimizations and new I/O interfaces in the ultra-low latency device using next-generation NVM technologies. Our conclusion is that simple changes in the existing storage software stack and I/O interfaces provide significant performance gain in terms of latency as well as throughput. Therefore, for system researchers, a deliberate review of the overall storage subsystem is required over the emerging SCM devices.

Although our implementations are based on the existing block layer for practical use, our work is independent of this layer and thus can be applied to various designs and implementations which focus on eliminating software stack overhead [1,3,11,12]. We will extend our work from block layer interfaces to kernel-bypassing interfaces considering the byte-addressable features of the future SCM devices [4]. Finally, we will also keep extending the I/O interfaces and the specialized functionality in hardware considering not only performance but also data manipulation like content-based chunking [2].

References

[1] AMEEN, A., CAULFIELD, ADRIAN M., MOLLOV, T. I., GUPTA, R. K., AND SWANSON, S. Onyx: A protoype phase-change memory storage array. In *Proceedings of the 3rd USENIX HotStorage'11* (2011), pp. 1–5.

[2] BHATOTIA, P., RODRIGUES, R., AND VERMA, A. Shredder: Gpu-accelerated incremental storage and computation. In *Proceedings of the 10th USENIX conference on File and Storage Technologies* (Berkeley, CA, USA, 2012), FAST'12, USENIX Association, pp. 14–14.

[3] CAULFIELD, A. M., DE, A., COBURN, J., MOLLOW, T. I., GUPTA, R. K., AND SWANSON, S. Moneta: A high-performance storage array architecture for next-generation, non-volatile memories. In *Proceedings of the 2010 43rd Annual IEEE/ACM MICRO'10* (2010), pp. 385–395.

[4] CONDIT, J., NIGHTINGALE, E. B., FROST, C., IPEK, E., LEE, B., BURGER, D., AND COETZEE, D. Better i/o through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 133–146.

[5] FREITAS, R. F., AND WILCKE, W. W. Storage-class memory: the next storage system technology. *IBM J. Res. Dev. 52*, 4 (July 2008), 439–447.

[6] HUFFMAN, A. Nvm express revision 1.0c. Tech. rep., Intel Corporation, 2012.

[7] INTEL, AND SEAGATE. Serial ata native command queueing. Joint whitepaper, Intel Corp. and Seagate Technology, 2003.

[8] LEE, B. C., IPEK, E., MUTLU, O., AND BURGER, D. Architecting phase change memory as a scalable dram alternative. ISCA '09, pp. 2–13.

[9] QURESHI, M. K., SRINIVASAN, V., AND RIVERS, J. A. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th annual ISCA'09* (2009), pp. 24–33.

[10] SEONG, N. H., WOO, D. H., AND LEE, H.-H. S. Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *Proceedings of the 37th annual international symposium on Computer architecture* (New York, NY, USA, 2010), ISCA '10, ACM, pp. 383–394.

[11] SEPPANEN, E., O'KEEFE, M., AND LILJA, D. High performance solid state storage under linux. In *Mass Storage Systems and Technologies (MSST'10)*, pp. 1 –12.

[12] YANG, J., MINTURN, D. B., AND HADY, F. When poll is better than interrupt. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, USENIX Association.

[13] YU, Y. J., SHIN, D. I., SHIN, W., SONG, N. Y., EOM, H., AND YEOM, H. Y. Exploiting peak device throughput from random access workload. In *Proceedings of the 4th USENIX conference on Hot Topics in Storage and File Systems* (Berkeley, CA, USA, 2012), HotStorage'12, USENIX Association, pp. 7–7.