Dynamic Near Data Processing Framework for SSDs

Gunjae Koo[1]*, Kiran Kumar Matam[1]*, Te I†, H. V. Krishina Giri Nara*, Jing Li‡,
Hung-Wei Tseng†, Steven Swanson‡, Murali Annavaram*
*University of Southern California
†North Carolina State University
‡University of California, San Diego

1. Introduction

Data movement cost from storage devices to compute nodes is extremely high for modern data processing applications. As such big data applications that rely on storage resident data pay a significant fraction of their execution time on the data input/output (I/O) time. Even though recent non-volatile memory (NVM)-based storage media exhibit enhanced access latency, the access latency to storage media is still thousands of times higher compared to DRAM. Even if the access latency of NVMs improve, the second hurdle is the bandwidth between storage and compute. Off-chip interconnects such as serial ATA (SATA) or PCI express (PCIe) are pin limited and hence their bandwidth grows rather anemically.

On the bright side, NVM-based storage systems support/need limited computing capability integrated with the storage cells. The compute node near storage acts as an intermediary between the host and storage by dealing with the vagaries of the NVM technologies, transparently to the end user. The innovation in this work is to recognize the opportunities, and limitations, of this near storage computing power and harness it in a practical way to implement a near data processing (NDP) model.

Compared to many prior works, our NDP model recognizes that the computing node near storage has limited capability, and the availability of even the small amount of compute near storage is not consistent. Moving large amounts of general purpose computing near storage may reduce the bandwidth cost, but they may incur significant performance penalties by placing an undue burden on the small compute nodes near storage. Some of the previous solutions have tackled this issue by offloading only simple functions on the SSD processors [1], [2], [3]. For this approach programmers need to partition applications in order to determine which functions may be offloaded to SSD processors. Such a manual fine tuning makes NDP unpalatable for many use case scenarios. As such, we argue that the purpose of NDP should be to offload certain computations transparent to application programmers, while at the same time the purpose of offloading should not be to reduce the computing burden on host CPUs. Rather it must be tailored primarily for reducing the communication bandwidth. As such whatever computing is done near storage must be performed with the aim of reducing the communication cost.

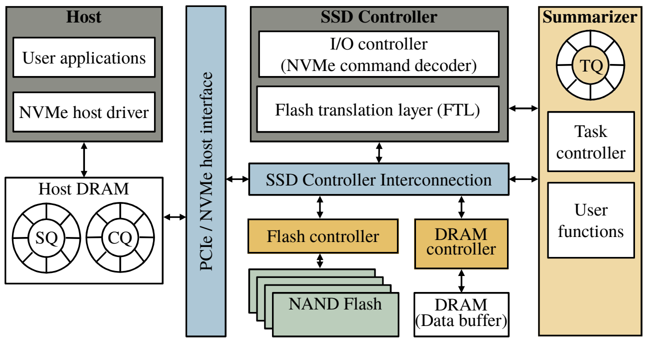

Figure 1. The overall architecture of NVMe SSD controller and Summarizer

While the approaches presented in this work can be applied to any NDP we specially target solid state disk drives (SSDs) built using flash storage chips in our implementation and evaluation. SSDs equip general-purpose embedded processors to support NAND flash management functions, such as the flash translation layer (FTL) for address translation, garbage collection, and wear-leveling. Using a set of motivational experiments on off-the-shelf SSDs we show that the computation power of the embedded processors in SSDs is underutilized on average, although there are instances when much of the available compute power is in fact needed to handle the intrinsic difficulties of flash memory – lack of in-place updates and write amplification. The dynamic nature of compute availability near storage, compounded with the dynamic nature of resource demands from the offloaded computations makes a practical NDP design challenging.

2. Summarizer NDP framework

We solve this challenge by designing a dynamic near data processing framework – *Summarizer*. Summarizer is a firmware architecture and a programming model for NDP. Summarizer firmware architecture supports opportunistic NDP control, which executes a set of user-defined offloaded functions when the computation resources are available near storage. Summarizer firmware monitors the availability of the SSD processors to enable in-SSD computation only when function offloading is beneficial. We also propose the programming model that exploits the dynamic NDP

This work is originally published as the title with "Summarizer: Trading Communication with Computing Near Storage" appeared in the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).
[1] *Gunjae and Kiran made equal contributions to the work.*

framework offered by Summarizer. To show the practicality of our approach on real systems, we implemented the Summarizer framework on an industrial-strength SSD development platform which employs non-volatile memory express (NVMe) based flash storage. By running a broad range of data-intensive applications that also need non-trivial computing resources we show that Summarizer can provide significant performance benefits.

Figure 1 depicts the Summarizer firmware architecture and its interactions with a traditional NVMe-based storage system. In current NVMe systems the host-side NVMe device driver receives an application's I/O requests as a series of NVMe commands issued by the host OS [4]. The SSD controller decodes the NVMe commands issued by the host driver, and then generates flash memory access commands. The data is then returned back to the host at a page granularity. Summarizer proposes to enhance the NVMe interface with a small subset of application programming interfaces (API) that can be used for NDP. Summarizer takes a three step approach to enhance this traditional NVMe interface for performing NDP – *initialization*, *computation* and *finalization*.

In the *initialization* step, Summarizer API transfers an in-SSD computation procedure to SSD DRAM space. In essence, Summarizer provides an API for the application to first register a set of user functions that may be executed in the SSD embedded processor opportunistically. Programmers are able to offload the custom functions to the SSD DRAM buffer using Summarizer API. Summarizer firmware allocates the memory space for the offloaded user functions, and manages the function address pointers in the user function stack. Each function ID in the user function stack entry is mapped to the address of each offloaded user function. The data structures and the temporal variables used for the in-SSD computation functions are also initialized.

Summarizer API sends the in-SSD computation flag and the function IDs alongside with data read requests in the *computation* step. This function ID is delivered to Summarizer firmware on SSD. The firmware then performs an FTL translation and then issues a page read request as is done in the traditional SSD. In addition the Summarizer firmware registers the NDP function call request in a task queue (TQ) within the SSD. The flash page requested is usually buffered in the SSD DRAM and is then transferred to the host memory through direct memory access (DMA). Summarizer firmware interacts with this data fetch path in the SSD controller to dynamically enable NDP based on the available computation resources in SSD.

The firmware decodes NDP computation requests when the fetched page data is buffered in SSD DRAM. If NDP computation is not requested, the buffered page data is transferred to the host DRAM without any further processing. If the NDP computation is requested on the buffered page data, Summarizer firmware uses the TQ entry to lookup the function call address and then launches a function call execution on the SSD core. Each TQ entry contains the function ID and the address pointer for the buffered page. Once the SSD controller finishes the user-initiated function it sends the special completion code that indicates the corresponding page data is already processed in SSD and only the results are delivered to the host.

Summarizer task controller opportunistically assigns work based on the available computation resources in the SSD controller. Rather than relying on a complex monitoring hardware Summarizer simply relies on the size of TQ to determine the availability of resources. If TQ is full Summarizer transfers the page data without NDP computation even when NDP computation is requested. In this case data movement cost is not reduced since entire page data is transferred. However, Summarizer essentially informs the host that the processing delays may overwhelm any perceived benefits of bandwidth reduction.

With Summarizer's NDP control scheme, the computation for data set is partially performed in SSD processors and the rest part is done in host CPUs. Hence partial results computed in the SSD processor is merged with the host computation results in the *finalization* step. Summarizer API sends the NVMe command to finalize the in-SSD function procedure and request the results from the Summarizer SSD.

To support each step we define a few NVMe commands. However, rather than requiring a new NVMe protocol stack we piggyback the new NVMe commands on existing NVMe commands. We essentially repurpose some of the reserved fields in the existing NVMe command structure to create additional semantics for NDP. The Summarizer NVMe commands exploit these reserved fields to deliver the information required for in-SSD computation.

3. Evaluation

We evaluated the performance of Summarizer using an industrial-strength flash-based SSD reference platform. The platform board is equipped with the multi-core ARM processor which runs the SSD controller firmware and FPGA implementing NAND flash controllers. We selected data analytics and data integration applications to evaluate Summarizer. Our evaluation shows that Summarizer's dynamic task control mode improves the average performance by 20% compared to the processing at host CPUs only, and $6\times$ compare to the processing at SSD processors only. We explored the design space by changing the SSD's internal bandwidth and computation power and show that Summarizer can be a cost-effective solution to improve the overall system performance for future SSD platforms.

References

[1] S. Boboila, Y. Kim, S. S. Vazhkudai, P. Desnoyers, and G. M. Shipman, "Active flash: Out-of-core data analytics on flash storage," in *Mass Storage Systems and Technologies, 2012 IEEE 28th Symposium on*, ser. MSST '12, April 2012, pp. 1–12.

[2] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt, "Query processing on smart ssds: Opportunities and challenges," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 1221–1230.

[3] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho, J. Jeong, and D. Chang, "Biscuit: A framework for near-data processing of big data workloads," in *Proceedings of the 43rd International Symposium on Computer Architecture*, 2016, pp. 153–165.

[4] "Nvm express revision 1.1," http://www.nvmexpress.org/wp-content/uploads/NVM-Express-1_1.pdf.