IJCSE International Journal of Computer Sciences and Engineering Open Access
Study Paper Volume-4, Issue-5 E-ISSN: 2347-2693

Study on Block Device Driver and NVMe their Implementation Impacts on Performance

Raman Kumar Kharch[1*], Vijay D. Katkar[2] and Kedar Kulkarni[3]

[1,2] *Department of Information Technology Pimpri Chinchwad College of Engineering Pune, India*
[3] *Center of Development in Advance Computing, Pune, India*

Available online at: www.ijcseonline.org

Received: Apr/27/2016 Revised: May/09/2016 Accepted: May/24/2016 Published: May/31/2016

***Abstract*—** Solid-State Drive (SSD) is also known as Solid-State Disk it contains no moving components. Attraction for SSD is due to its high throughput and scalability. It distinguishes from traditional magnetic disks like hard disk drives which contains movable head and spinning disk. SSDs are electronic circuit built on NAND-Flash/NOR-Flash and PCM. Solid-State Drive uses non-volatile memory for storage and retrieval of data or information in the form of sectors and/or pages and shows better performance than hard disks. Maximum IO performance of the used memory technology can be achieved using a properly written software device driver, which can effectively utilizes underlying hardware resources and extracts the maximum performance from the storage device. This paper is a survey on key literature on IO performance of SSD and block driver. It deals with the effort that defines what characteristics an effective solid state drive should have. The paper also discusses trends and categories in research and questions that are further open for investigation.

***Keywords*—** Dynamic Block Driver, NVMe, Solid state drive, block layer, latency.

I. INTRODUCTION

From hand-held embedded system to super-computers, data storage is massive for all computing system. Memory system plays the primary role in determining application performance, reliability, power consumption [4]. With increase in data consumption the applications demand high performance within the stringent power and cost constraint. These requirements made researchers to develop flash memory based storage devices termed as Solid State Drive (SSD). SSDs are the PCIe storage devices. Multiple NVM channel works in parallel are used in SSDs. The data retrieval process in SSD is faster and efficient than traditional HDDs [10].

Solid State Drives are more reliable, faster and more efficient than the traditional Hard Disk Drive. Various NVM technologies have different characteristics in terms of data block, need for erase, aging, rewritability and bit error rate. There are two categories of NAND flash Single-Level Cell (SLC) and Multiple-Level Cell (MLC). SLC flash stores one bit of data per cell. MLC flash stores multiple bits of data per cell.

For interfacing with the user applications at the host system, meaning no changes with the user-level, the OS software layers and I/O stack used for HDDs remain the same and the differences are encapsulated by an emulation software layer, called the Flash Translation Layer (FTL), which is added in the SSD's storage controller [6].

For handling the system calls related to filesystem from the user applications Linux kernel uses the Virtual File System (VFS). VFS is an abstraction layer of a more concrete filesystem. It is an interface between the kernel and various other filesystems [7]. One of most important feature of Linux kernel is it supports different types of filesystem. The VFS manages all this different filesystem that are mounted that the given time. The user application accesses the storage devices through the standard system calls to the filesystem. The kernel then forwards the request(s) to the VFS layer, which forward the request to the generic filesystem specific function(s). The filesystem know about the logical layout of the data, and perform functioning to the block layer on behalf of the user application.

The Block Layer is a middle layer between the Linux kernel and the storage device. It allows the application to access the storage device and it includes single point of entry from all application to the storage devices and driver. A user application uses a block device through the filesystem; the virtual file system (VFS) is the entry for all the requests. The recently read and written portions of the block devices are store in buffer/page cache of the kernel. It is then forwarded to the block layer, which allows I/O requests to the block drivers [9]. Block layer implements IO scheduler, allows merging request, reorder request. The response is send back to the user application through the same path, preserving the order.

The rest of the paper is organized as follows. Section 2 covers an overview of the PCIe-based Storage Device and Non-Volatile Memory Express. Section 3 discusses related work, the main observations and findings of the study and analysis. Finally, we conclude our work in section 4.

© 2016, IJCSE All Rights Reserved

II. LITERATURE SURVEY

A. *Storage Device*

The performance of PCIe storage devices are depend on the used NVM technology, the NVM channel, host I/O interface, internal architecture of the storage controller, and the upper layer software components that counts device driver. PCIe-based SSD was announced by Fusion-io in 2007 [5]. In single card it has 100,000 IOPS of performance, with capacity up to 320GB. NAND flash based SSD are used widely today. Unlike DRAM, NAND flash is a non-volatile storage of data which can retains data without power [8]. The emerging NVM technologies such as PCM (Phase Change Media), Memristor, are the component which can replace NAND flash. I/O latency in SSD based on NVM exhibits little difference between sequential and random IOs. On parallel IOPS single lock coarse design becomes a bottleneck to overall performance for protecting the request queues.

B. *Non-Volatile Memory Express (NVMe)*

NVMe is an interface specification for accessing the SSD attached through the PCIe bus. NVMe is a logical interface. It is a software based standard that was specifically optimized for PCIe-based interface SSDs. NVMe block driver was published by Intel for Linux [4]. In the kernel of Linux, a resizable or scalable block layer for high performance SSD are fused together. As a result of this fusion the performance of I/O submission rate increases. The main structure NVMe protocol contains two commands, namely Admin command and IO command. Admin command is used to inform NVMe-capable controller for creating, deleting IO queue. IO commands are the core of the functioning. Contents of the IO commands indicates read/write/flush request. To hold such commands, NVMe maintains two queues namely, Submission Queue (SQ) and Completion Queue (CQ). The SQs are used to submit the command to the NVMe controller and the CQs are written by the NVMe controller to indicate status of the command and error reporting if occurs. Each core can have multiple SQ and CQ. In the direct memory access (DMA) region of host memory the both queues are located. The I/O commands are executed in the NVMe-controller in four stages fetch, decode, execute, complete. The NVMe controller is implemented in hardware which is mainly responsible various stages required for the completion of IOs.

III. RELATED WORK

For the purpose of achieving high throughput and low latency dynamic block device driver has been proposed in Prototyping and Performance Evaluation of a Dynamically Adaptable Block Device Driver for PCIe-based SSDs [1]. Dynamic Block Device Driver is compatible with Linux I/O stack, and it is flexible with a PCIe-based NVM storage device. Due to which it provides high performance interface between the host computer and storage device. Figure 1 shows the flow of device driver. The functionality of Dynamic Device Driver which consists of three main components, they are I/O Request, New Descriptor Block Processor, I/O Responses Generator. The bio structure is used to represent I/O requests between block layer and device driver. The structure represent block I/O operation that are active (flight). From the block layer bios are passed to the driver space. The device driver translate bio into proper descriptor, which is then forwarded to the interface queue and finally passed to the storage device for processing. The PCIe device generates an interrupt to the device driver about signal completion of previous block. Now, the New Descriptor Block Processor function is activated for forwarding a new block to the PCIe device with descriptors, only when if there are some pending request available. The descriptors to the PCIe device are variable-size blocks. The block returned by the PCIe device with the response is processed in I/O Responses Generator function (The New Block Descriptor Processor function wakes it up). The function checks all updated contents and flag of descriptor of returned block, generates respective I/O completion and transferred it to the user application through block layer, saving the original order. Whenever a block of descriptor is returned from the PCIe device, the I/O Responses Generator function finds the respective bio pointers in 'Bio Array', generates I/O completions and removes the pointers from the array.

For pending requests from bio structure, a memory point Bio Array is defined along with two pointer variables, one point to the first descriptor of the next block that will be sent to the PCIe device, and another one which points to the first free position in the interface queue for the future descriptors.

The advantage of using dynamic device driver is that it supports variable-size blocks of descriptor. The cyclic buffer is implemented on the interface queue, and it is of fixed size.

In the kernel of Linux, a resizable or scalable block layer for high performance SSD are fused together. As a result of this fusion the performance of I/O submission rate increases. For managing I/O NVMe maintains two queues namely, Submission queue and Completion queue. Each core can have multiple submission queues. I/O latency in SSD based on NVM exhibits little difference between sequential and random IOs.

© 2016, IJCSE All Rights Reserved

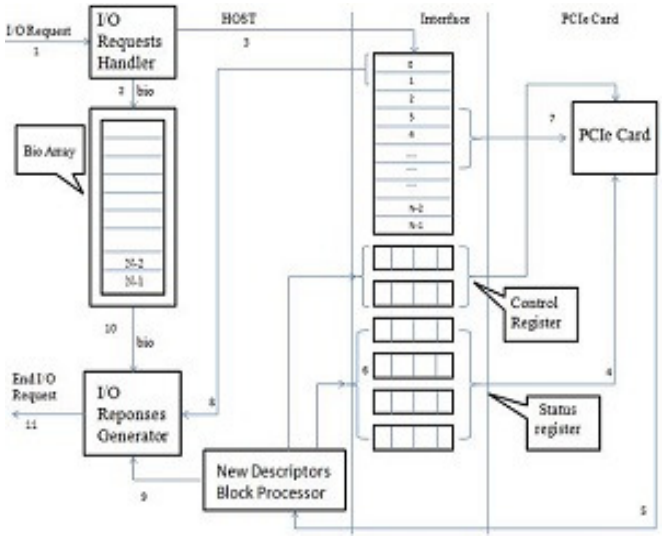

Fig. 1: Dynamic Block Driver Architecture

On parallel IOPS single lock coarse design becomes a bottleneck to overall performance for protecting the request queues. In Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems [2], to minimize cache coherency multiple IO submission/completion queues across CPU cores. The design contains idea of introducing two level of queue in the block layer: (a) Software queue, (b) Hardware queue.

Figure 2 shows the two-level queue mechanism in block layer. Block layer cannot support IO scheduling across devices in single queue per device. The block layer of SATA is evaluated with high performance NVMe-based SSD and found that block layer for Linux environment has considerable overhead for each IO. The three main problems, namely Request Queue Locking, Hardware interrupts, Remote memory access. For the multi-queue block layer, the lock contention is moved to two-level queue from single-level is designed for the Linux block layer. Three major requirements are: Single Device Fairness, Single and Multiple Device Accounting, Single Device IO staging area (place for IO scheduling). Functionality of the two-level queues are, Software Staging Queues, dispatching IO request rather in single queue it is now maintained in multiple block IO request per core, per socket on the system. Hardware dispatch queues, IO on entering into the staging area is sent to the hardware dispatch queues, instead of to the device drivers. In the queue, it matches the number of hardware contexts supported by the device driver. The number of queues supported by the device driver is anywhere from one to 2048 queues. IO ordering is not possible inside the block layer any software queue may feed any hardware queue without needing to maintain a global ordering. This* allows hardware to implement one or more queues that map onto CPU's directly and provide a fast IO path from application to hardware that never has to access remote memory on any other node.

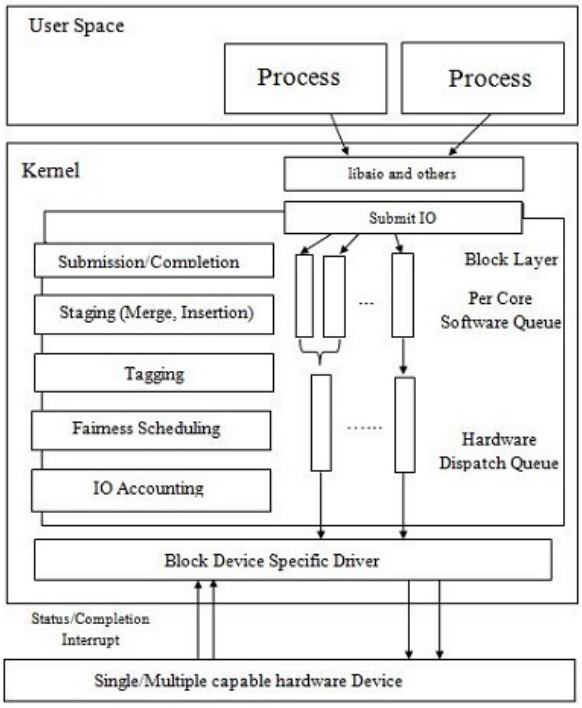

Fig.2. Two-Level Queue Block Layer

The requests in the hardware queues are not sorted but queued using FIFO policy: the incoming block IO submitted by the core i is inserted to top of the request queue attached to core i. The device performance doesn't get affected because of interleaving IOs from multiple software dispatch queues into a single hardware dispatch queues. Reason is: the two level queuing mechanisms rely on the fact that the random read and write latency are faster than the sequential for modern SSDs. The SATA based SSDs have single completions using single interrupt, single hardware dispatch queue. In the driver submission queue for uniquely identifying the position of the block IO an integer value tag is used. After completing it is passed to the driver. The tag is then re-used by the device driver. The need of linear search is eliminated for checking of completion of IOs. The block layer generates a unique tag associate to the IO i.e. inserted to the hardware dispatch queue. For supporting fine grained IO accounting the internal Linux library are modified. The modification is done to provide statistics for the state of both the software queues and the dispatch queues.

In Non-Volatile Memory Host Controller Interface Performance Analysis in High-Performance I/O Systems [3] show that, the NVMe controller performs the I/O commands in four stages. First it fetches the command from the SQ in the main memory and writes it to its Outstanding Requests Tracker (ORT). Second, it decodes the command. Third, it depends on the number of pages that needs to be read from or write to the host memory. If the number of pages more

© 2016, IJCSE All Rights Reserved

than two, we need to read a table of pointers to these pages. This table is called Physical Region Descriptor Table (PRDT). Fourth, the NVMe controller has to notify the host about the completion of the executed I/O request, command, and error occur during execution. The organization of I/O SQ and CQ can affect the performance. The system has a single SQ/CQ, shared by all cores gives the lowest memory overhead. Clearly this model has a synchronization bottleneck it needs to acquire lock for the queue when each time a core wants to submit an I/O command. Otherwise the system has a SQ/CQ per core it avoids the bottleneck of acquiring single lock to all cores. For the scalability factor large number of allocated queue affects the system performance. The overall protocol scalability is affected mainly by three factors. First, the time spent submitting I/O command. Second, time to identify a completion entry and processing it. Third, submission/completion entries, reading/writing data. The software layer is the highest contributor for affecting the overall execution time. A software layer through which any I/O request needs to go in Linux is called block layer.

The performance highly degrades when interrupting the processor each time for an I/O command is ready. A significant amount of time is spent in context switching. To the best periodic interrupts can be used. Each tau microsecond period if at least one completion occurred, the NVMe controller triggers an interrupt. The large value of interrupt detection period tau microsecond can reduce the number of interruption to the processor, but can increases the gap between the time commands and ready and processed. The small value will increase the number of interruption. The moderate period can achieve good improvement over previous two. The gap between the commands is ready and being processed is low.

CONCLUSION

In this paper we have understood that the Linux Block layer and its importance. In the second section we studied the idea about storage devices and Non-Volatile Memory Express. In the third section we studied the dynamic device driver for PCIe-based SSD that is compatible with the Linux I/O stack.

ACKNOWLEDGMENT

This work is support by Center of Development for the Advance Computing C-DAC, Pune.

REFERENCES

[1] Eleni Bougioukou, Athina Ntalla, Aspa Palli, Maria Varsamou and Theodore Antonakopoulos, “Prototyping and Performance Evaluation of a Dynamically Adaptable Block Device Driver for PCIe-based SSDs”, IEEE 2014

[2] Matias Bjørling, Jens Axboe, David Nellans, Philippe Bonnet, “Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems”, SYSTOR ACM, 2013

[3] Amro Awad, Brett Kettering, and Yan Solihin,” Non-Volatile Memory Host Controller Interface Performance Analysis in High-Performance I/O Systems”, IEEE, 2015

[4] Sivashankar, Dr. S. Ramasamy, “Design and Implementation of Non-Volatile Memory Express”, International Conference on Recent Trends in Information Technology, IEEE, 2014

[5] Mojtaba Tarihi, Hossein Asadi, Alireza Haghdoost, Mohammad Arjomand, and Hamid Sarbazi-Azad, “A Hybrid Non-Volatile Cache Design for Solid-State Drives Using Comprehensive I/O Characterization”, IEEE, 2015.

[6] Hiroko Midorikawa, Hideyuki Tan, Toshio Endo, “An Evaluation of the Potential of Flash SSD as Large and Slow Memory for Stencil Computations”, IEEE, 2014

[7] Shuichi Oikawa, Satoshi Miki, “Future Non-Volatile Memory Storage Architecture and File System Interface”, First International Symposium on Computing and Networking, 2013

[8] Myoungsoo Jung, “Exploring Design Challenges in Getting Solid State Drives Closer to CPU”, IEEE, 2013

[9] M. Wu and W. Zwaenepoel, “envy: a non-volatile, main memory storage system,” in Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS VI. New York, NY, USA: ACM, 1994, pp. 86–97. [Online]. Available: http://doi.acm.org/10.1145/195473.195506

[10] Nguyen, A. ; Satish, N. ; Chhugani, J. ; Changkyu Kim; Dubey, P., “3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs”, High Performance Computing, Networking, Storage and Analysis, 2010

© 2016, IJCSE All Rights Reserved