ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 20, April 2016

Non Volatile Memory Host Controller Interface Performance Analysis

Vinod T.

Computer Science Department, Rohini College of Engineering and Technology, Tamil Nadu, India.

vinod.eng@gmail.com

***Abstract*— Hard Disk Drives (HDD) are predominantly used in datacenters to store and retrieve data created by individuals or businesses. Due to the performance limitations of HDD, Solid State Drives (SSD) has emerged as an alternative storage option. SSD uses Non Volatile Memory (NVM) to provide faster random access and data transfer rates than electromechanical based HDD. But the host interface to SSD remains a performance bottleneck and also the Input Output (IO) subsystem causes unnecessary latencies, translations in the Read/Write commands. In order to completely utilize the performance of the NVM present in a SSD, a Non Volatile Memory Subsystem has been recently designed. The communication to this new subsystem is through PCI Express interface and the command set is based on NVM Express (NVMe) Specification and the host controller interface is based on Non Volatile Memory Host Controller Interface (NVMHCI). This new subsystem exploits the parallelism and low-latency features of emerging NVMs and demonstrates better performance than SSD and HDD. In many cases, the maximum IO performance of the used memory technology is not achieved due to limitations imposed by the software device driver that interfaces the storage card with the host's operating system. The system performance bottlenecks and overhead of using the standard state-of-the-art Non Volatile Memory Host Controller Interface (NVMHCI) are analysed and investigated.**

***Keywords*— HDD, SSD, NVM, NVMe and NVMHCI**

I. INTRODUCTION

We have become information dependents of the twenty-first century, living in an on-command, on-demand world that means we need information when and where it is required. Data created by individuals or businesses must be stored so that it is easily accessible for further processing. In a computing environment, devices designed for storing data are termed storage devices or simply storage. The type of storage used varies based on the type of data and the rate at which it is created and used. Devices such as memory in a cell phone or digital camera, DVDs, CD-ROMs, and hard disks in personal computers are examples of storage devices. Businesses have several options available for storing data including internal hard disks, external disk arrays and tapes. The types of storage devices used in the workstations and servers as well as their corresponding host controller interface are briefed in the subsequent sections.

A. Hard Disk Drive

A hard disk drive (HDD) uses a rapidly moving arm to read and write data across a flat platter coated with magnetic particles. Data is transferred from the magnetic platter through the read write head to the computer. Several platters are assembled together with the read write head and controller, most commonly referred to as a hard disk drive. Data can be recorded and erased on a magnetic disk any number of times. The types of interfaces of HDD currently being used are SATA and SAS. The host controller interfaces which are predominantly used are Advanced Host Controller Interface (AHCI), Serial Attached SCSI (SAS) Controllers, and Fiber Channel (FC) Controllers.

B. Solid State Drive

Flash based solid-state drives (SSDs) are used as storage media for delivering ultra-high performance for mission-critical applications. Solid-state drives utilize flash memory to store and retrieve data. Flash drives have no moving parts, and leverage

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 20, April 2016

semiconductor-based block storage devices, resulting in minimized response time and less power requirements to run. Flash drives use non-volatile semiconductor memory to support persistent storage and they use either single-level cell (SLC) or multi-level cell (MLC) to store bits on each memory cell. Flash drives have been tested and qualified to withstand the intense workloads of high-end enterprise storage applications.

C. PCI Express

PCI Express based SSD is the recent innovation in the field of storage. The main factors for the wide-spread adoption of PCIe as an interface for SSDs are listed below. PCIe provides high performance and it is a full-duplex system, which can support multiple outstanding requests, and out of order processing. It has a scalable port width ranging from single lane (x1) to sixteen lanes (x16). It also has scalable link speeds which include 2.5GTps, 5GTps and 8GTps. PCIe has less number of pins hence, lower area and lower cost. It is a direct attach to CPU subsystem hence, it eliminates HBA cost. PCIe also provides effective Power Management.

II. EXISTING SYSTEM

In the existing system, high performance SSDs are connected directly to the host's internal I/O architecture through the PCI Express bus, a serial interface that can utilize multiple lanes in parallel and can achieve data rates higher than 1GB/s. Regarding the interface with the user applications at the host system and to ensure transition transparency, meaning no changes at the user-level, the OS software layers and I/O stack used for the 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. A major drawback of this approach is that, since most contemporary OS storage layers and I/O stacks are developed and optimized based on functional assumptions that are valid for mechanical disks only, the SSDs are prevented from reaching their full potential performance.

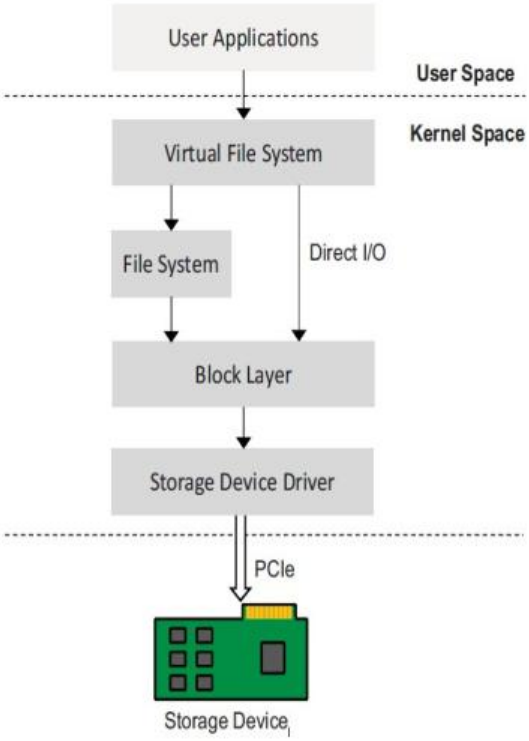

Fig 1 Existing System

The applications at the user level access storage devices through standard system calls to the file system. The kernel forwards these requests to the virtual file system (VFS) layer, which interfaces generic file system calls to file system-specific functions. The file system is aware of the logical layout of data and metadata on the storage medium and sends read and write requests of fixed-size data blocks. The block layer provides an abstract interface which conceals the differences between storage devices of different technologies. The software layer is the largest contributor to the overall execution time. The software layer includes but not limited to the I/O software stack. For example, in Linux, any I/O request needs to go through a software layer called block layer, this layer is responsible of scheduling I/O requests, merging requests and finally submitting it to the corresponding controller device driver (e.g. AHCI, FC). When the response arrives, it has to complete the corresponding I/O request.

III. NVM EXPRESS

A new storage device interface has been architected for non-volatile memory (NVM) based on Peripheral Component

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 20, April 2016

Interconnect Express (PCIe) to deliver higher performance. To realize the full benefits of NVM over the next decade, the engineering has defined NVM Express (NVMe), the software interface optimized for PCIe SSDs. NVMe was architected from the ground up for NVM to ensure a high-throughput, low latency, robust solution. SSDs based on future NVM technologies promise to deliver speeds on the order of millions of IOP Seven for client devices. NVM Express is a standardized interface is need for the easy adoption of PCIe based SSDs. NVM express specification provides this standard interface for PCIe SSDs. NVM Express is scalable host controller interface standard, which is designed for Enterprise and Client systems that use PCI express SSDs. It provides a simple, streamlined and efficient command set which eliminates the legacy HDD to SSD command conversion overhead. It provides an optimized register interface that allows the host software to communicate with the non-volatile memory Industry Support. It was developed by industry consortium of 80+ members and hence enjoys a very wide industry support.

A. NVMe Structure

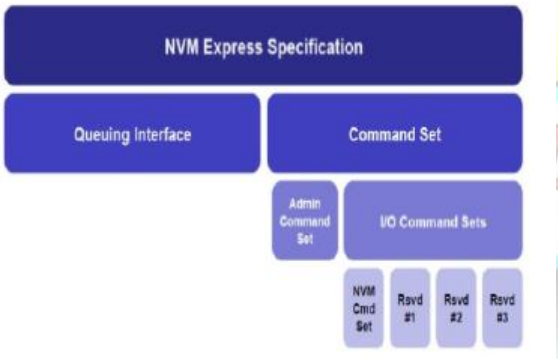

Fig 2 NVMe Structure

The NVM Express specification is composed of two important modules; the first one is the Command Set and the next one is the Queuing Interface. The command set covers both the Admin Command Set and IO Command Set. Admin commands are used for creating / deleting Submission and Completion Queues. The current version of NVM Express specification has a NVM Command Set which replaces the existing SCSI Command Set which was widely used across all major Operating Systems. This eliminates the latencies involved in translating the SCSI command to the corresponding NVM command by the NVMHCI device driver.

B. Queuing Interface

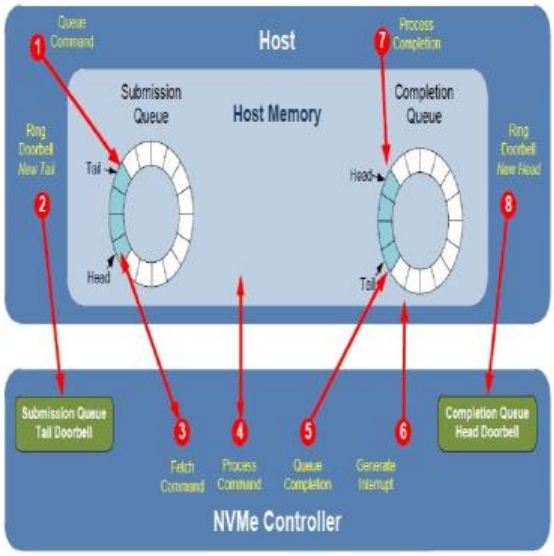

Fig 3 Queuing Interface

The basic steps involved in queuing interface are Command Submission, Command Processing and Command Completion. Commands are submitted to the NVMe controller through the Submission Queue and the responses are received through the Completion Queue. Command Submission involves 1) Host writes command to submission queue. 2) Host write updated submission queue tail pointer to doorbell. Command Processing involves 3) Controller fetches command. 4) Controller processes commend. Command Completion involves 5) Controller writes completion to completion queue 6) Controller generates MSI-X interrupt 7) Host processer completion. 8) Host write updated completion queue head pointer to doorbell. The host can ensure that the CPU core that created the commands for a particular queue pair then processes the interrupt, ensuring that the necessary data is local to that CPU core. In addition, the host can configure the system to share the interrupt processing load across cores.

C. Multicore CPU Support

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 20, April 2016

A limiting factor in SSDs' IOPS performance is the interrupt architecture. NVMe supports MSI-X interrupt steering, which efficiently allows a multicore host CPU to share the load. Host software configures the core that receives the MSI-X interrupt associated with each I/O completion queue. The host can ensure that the CPU core that created the commands for a particular queue pair then processes the interrupt, ensuring that the necessary data is local to that CPU core. This avoids the latencies which are involved in maintaining cache coherency as the data always stays in the cache of the local CPU core. If the Submission and Completion Queues are shared among multiple CPU cores, we would require locks for synchronization. But having a dedicated Submission Queue and Completion Queue per core avoids the usage of locks. In addition, the host can configure the system to share the interrupt processing load across cores. Having a single core process the interrupts limits IOPS to 200,000, but using interrupt steering and interrupt coalescing has been demonstrated to exceed 2 million IOPS.

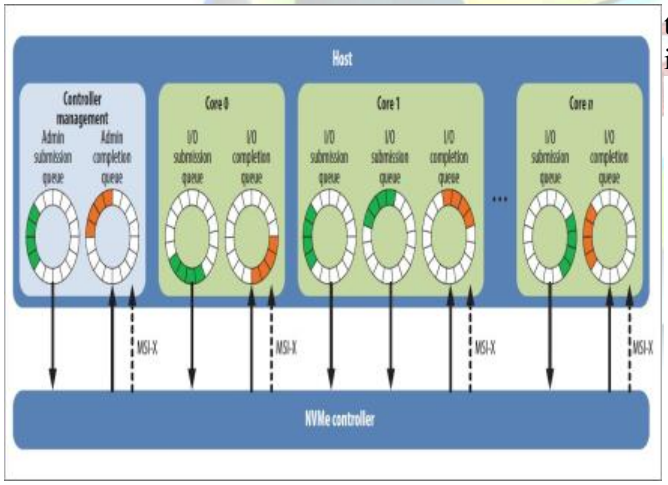

Fig 4 Multicore CPU Support

D. PRDT Table

The PRDTs allow the kernel to allocate physically non-contiguous page frames, instead of a large contiguous area starting from the address provided in the command. Note that different implementations of the direct memory access (DMA) engine can deal with PRDTs differently; for example, some DMA engines do not transfer the PRDT at all, but parse the PRDT using the DMA engine on the memory hub (MCH), which requires all the pages pointed to by the PRDTs to be transferred before moving to another request. In case of NVMe, the PRDTs are read by the NVMe controller. This allows having overlapping I/O requests at the same time and gives more flexibility to the NVMe controller firmware implementation.

IV. EXPERIMENTAL RESULTS

A. Overall Execution Time

The execution time changes when the number of processes is increased for both of the Single Queue (SQ) and Multiple Queues (MQ) queuing models. Each process reads a specific part of the file, which allows processes to proceed simultaneously. The MQ model scales better than the SQ model for up to 4 processes. However, in the case of 8 processes, it is just slightly better than the SQ model.

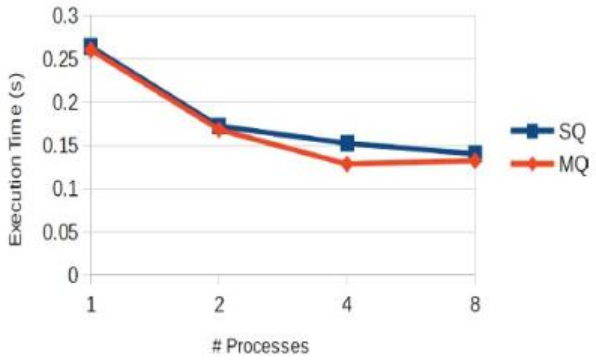

Fig 5 Overall Execution Time

B. Command Submission Time

The average time for submitting a command using the SQ model increases fast as the number of processes increases, but increases at a slower pace

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 20, April 2016

with the MQ model. Changing the number of processes from 4 to 8 increases the average submission time by 30% and 5%, for SQ and MQ models, respectively. The time is measured beginning from trying to lock the submission queue, and end when the NVMe controller is notified of the submission. The submission algorithm will involve multiple steps to transition the input command to the required format. The algorithm may need to acquire a lock for synchronizing among multiple threads running in multiple cores simultaneously.

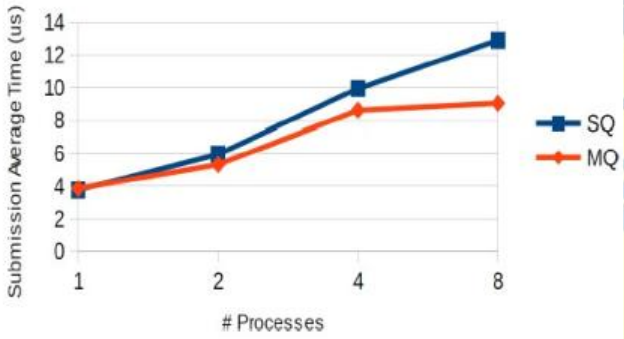

Fig 6 Command Submission Time

C. Command Completion Time

The average time of command completion, starting from trying to acquire the lock of the completion queue and the corresponding submission queues, until the NVMe controller is notified of the completion. We can observe that the SQ model has a higher average completion time when running 4 processes, which is mainly caused by the time to acquire the lock of the single submission queue in the system. Increasing the number of processes to 8 gives approximately the same average completion time for SQ and MQ.

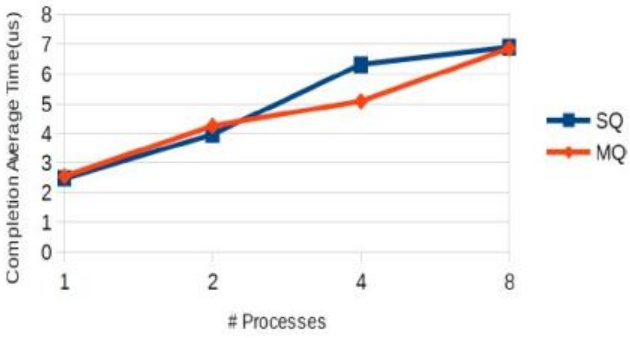

Fig 7 Command Completion Time

V. Conclusions

We have investigated the NVM host controller interfaces feature and the impact of the system software on the system performance. We studied different queuing models, and explained different system bottlenecks and performance bottlenecks for each model. Based on our study, the largest overhead is coming from submission commands, then from completion entries written by the NVMe controller to the completion queues. Reducing the overhead of the submission commands and completion notifications can be analyzed and investigated in the future.

References

[1] Amro Awad, Brett Kettering, and Yan Solihin, "NVMHCI Performance Analysis in High-Performance I/O Systems", 2015 IEEE.
[2] Amber Huffman and Dale Juenemann, "The Nonvolatile Memory Transformation of Client Storage", Published by the IEEE Computer Society 0018-9162/13 2013 IEEE.
[3] Sivashankar and S. Ramasamy, "Design and Implementation of Non-Volatile Memory Express", 2014 International Conference on Recent Trends in Information Technology
[4] Huffman, "NVM express 1.0c specification" February 16, 2012.
[5] Pci-sig. pci express specification revision 3.0. Technical report, 2012.