Maximizing Server Capability using NVMe Dims and SSD Devices

Vishwanath Burkpalli[1,], Arvind Kumar Mekin[2]

[1]Professor, P.D.A College 0f Engineering College, Kalaburgi – 585102, India.

[2]Department of Information Science, P.D.A College 0f Engineering College, Kalaburgi – 585102,

Abstract: This paper presents a mechanism to enhance the server capability using latest NVMe memory and SSD disks. The experimentations are done on current server hardware and Software components. The current Server hardware and Operating system components are optimized to use magnetic hard disks. These latest these devices are not having any mechanical components like HDDs. So there is necessity to optimize server hardware components and Software algorithms for utilizing the Server's capability to maximum extent by optimizing the hardware and operating system components

Index Terms: PCIe , Storage technology, Solid State Disk, Non Volatile Memory

1. INTRODUCTION

PCI Express (PCIe) Solid State Storage devices and memory are becoming increasingly popular in the high performance computing environment for both client and enterprise storage. The development and deployment of Solid State Drives (SSDs) has seen the introduction and adoption of SATA and SAS SSDs as replacements for or improvements on traditional magnetic Hard Disk Drives (HDDs). The introduction of even faster PCIe based SSDs with lower latencies has introduced another wave of SSD technology and architectures. The mass storage is now being optimized for SSDs. Enterprise servers have PCIe interfaces now, client systems are anticipating one or two lane PCIe devices embedded on laptop motherboards. Meanwhile, enterprise applications are seeing the evolution of PCIe SSDs as both superfast block IO devices as well as persistent memory. These developments have led to surges in hardware, software and standards works. This white paper will survey the emerging high performance PCIe storage and the implications of revolutionary applications of these new architectures. NVMe (Non-Volatile Memory express) is an PCIe extended interface for SSD devices.

Non-Volatile Memory express is an specification specially designed for accessing the Solid State Disks which are attached through PCIe bus. NVMe specification is developed in industry working group rather open standard forum. It defines optimized register interface, Command set and Features set. It's more focus is on non-volatile memory devices. Current massive storage devices like SSDs are flash based, so they can leverage the NVMe specification efficiently. The SSD devices which are designed based on these specifications are popularly called as NVMe based PCIe SSDs. Most of the latest enterprise servers are coming with PCIe lanes to accommodate these interfaces. The focus of this paper is on how to make use of PCIe SSD interface to maximize the server performance.

The current hard disks and SSDs are connected to the system via SAS/SATA interfaces. Though SSDs have the capability to connect directly use PCIe interface which have much superior transfer rate than SAS/SATA interfaces.

The below schematic block diagram represents the architecture of connecting storage devices in existing enterprise servers,

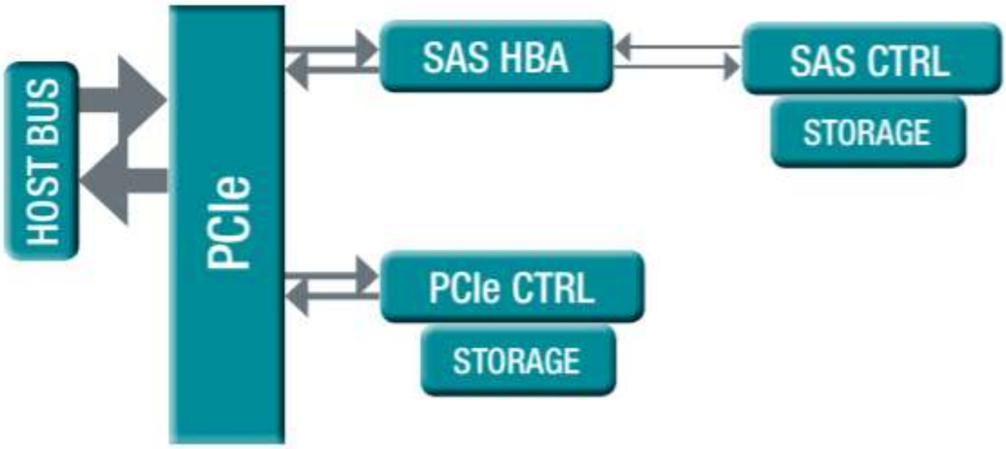


Figure – 1 : Storage connectivity in existing servers

2. PROPOSED METHODOLOGY

The proposed methodology suggests how to connect the latest technologies to utilize their strengths in current servers for improving their performance. The schematic block diagram encompassing all the stages of the proposed methodology is shown in Figure 2.

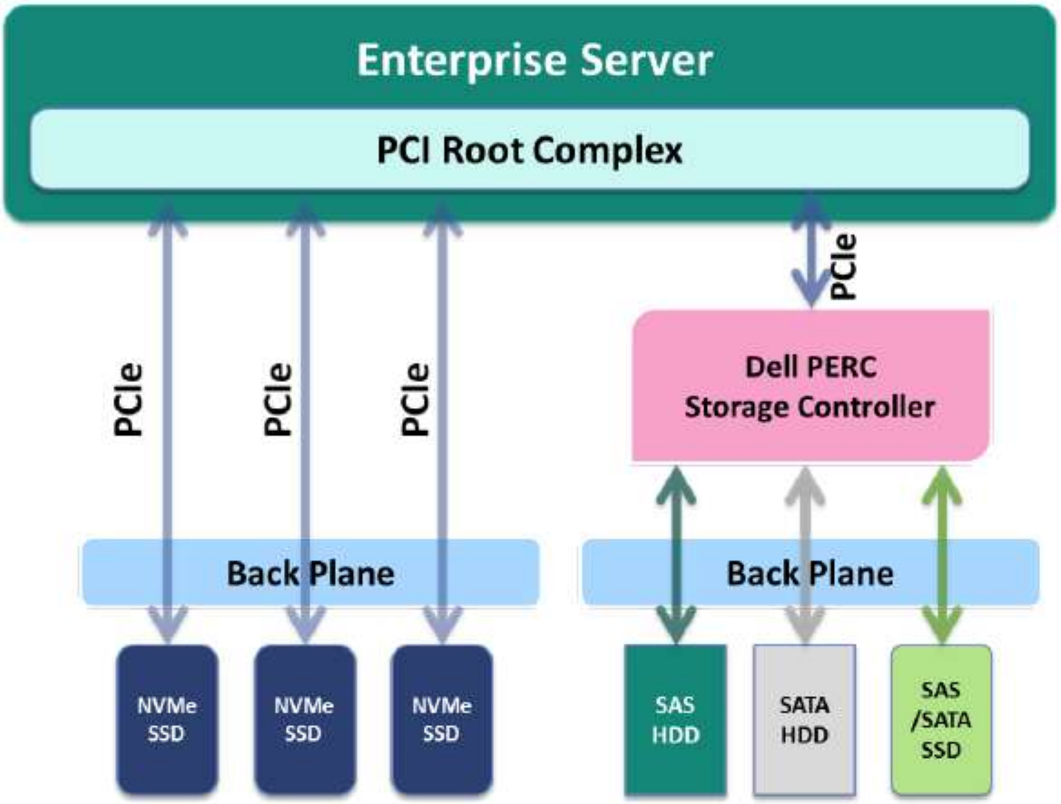


Figure – 2 : NVMe Connectivity

The operating system side support for NVMe is required in greater extent. The whole hard disk support layer follows entirely will not be relevant with respect to the NVMe based SSDs. The legacy storage interfaces are optimized for rotating media technology i.e Magnetic disks. The increasing popularity of NVM technologies such as NAND Flash created demand for a host controller interface optimized for NVM. NVM Express is a scalable host controller interface architected to meet the needs of enterprise and client systems using PCIe solid state drives. NVMe is efficient, scalable and high performance. The growth of multi-core processor systems means more multi-threading, and therefore more operations in flight. NVMe meets the needs of multi-core architectures by scaling up to 64K queues with 64K commands per IO queue. Enterprise capabilities like end-to-end data protection, virtualization and enhanced error reporting are included. The queue structures in NVMe interface are,

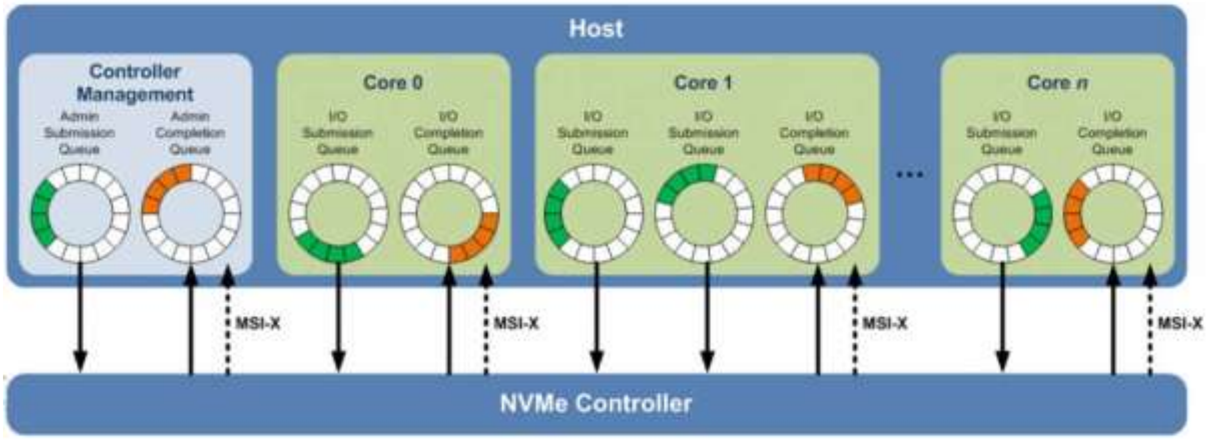


Figure – 3 : NVMe Queue Structure

Operating system layers need to tweaked to utilize the capability of NVMe devices. The various layers involved and needs to be re-arranged for talking to the storage devices are depicted as in the below picture.

Request layer, SCSI layer, and HBA layers are must HDDs, whereas these layers are bottleneck for the NVMe devises. To make use of NVMe based SSDs these layers are not necessary so operating system components of new enterprise servers can be striped off these un-necessary components which enhance the operating system performance.

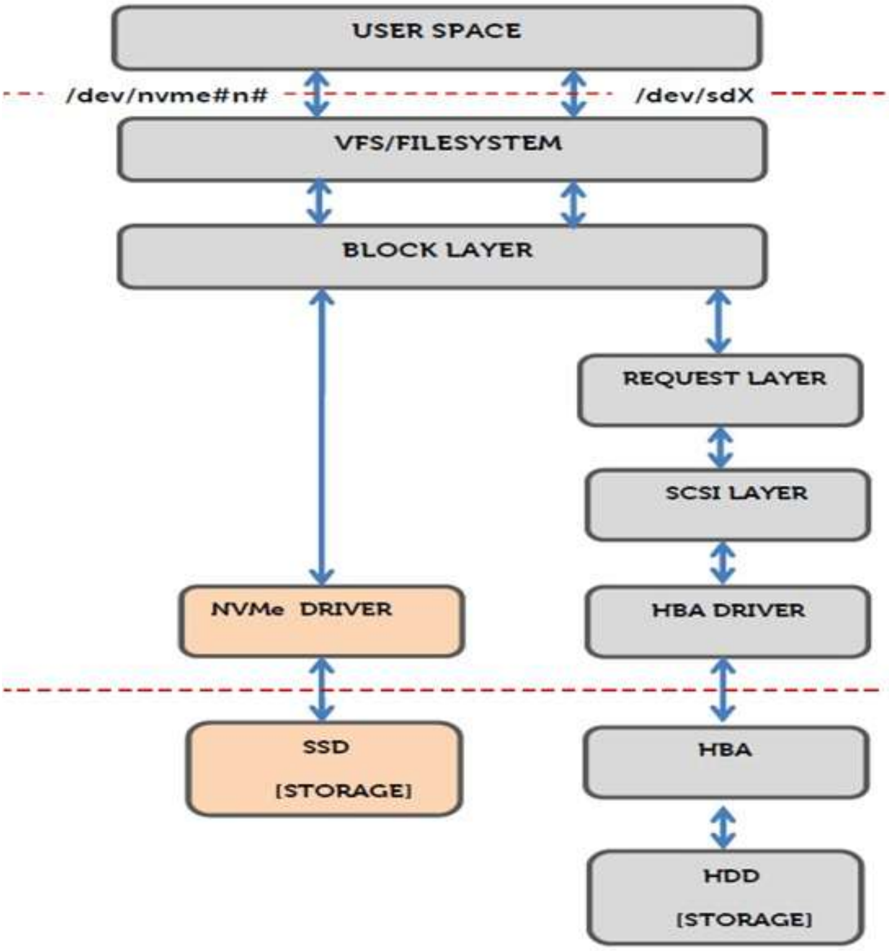


Figure – 4 : Operating System Layers for NVMe & traditional Storage

NVMe based SSD devices installed in enterprise servers can be utilized as,

1. Block Storage device for file system
2. Caching mechanism
3. Raw storage device for database repositories

2.1 The traditional Storage programming model

As a general rule, applications can operate on data of any length, starting at any address in system memory. But the data must be stored in fixed-sized blocks on disks. Many applications use a basic storage programming model where file systems manage the differences between memory and disk; file systems typically do this using a memory cache. Where possible, application read and write operations are directed to the cache rather than accessing the disk.

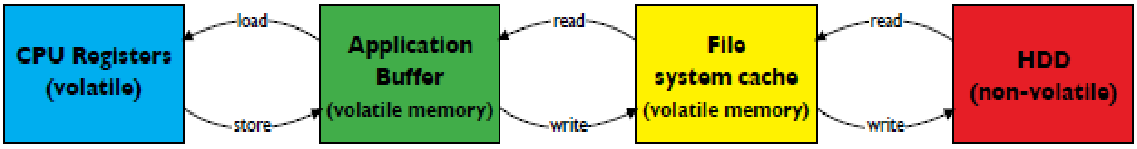

CPU Registers (volatile)
load
store
Application Buffer (volatile memory)
read
write
File system cache (volatile memory)
read
write
HDD (non-volatile)

Figure - 5 : Basic Storage data flow

The programming model used for basic storage is:
- Applications create or modify data using CPU registers
- Data is temporarily stored in volatile memory
- To assure the data is durable across system and application restarts, applications write the memory contents to files
- The file system utilizes a cache (also stored in volatile memory) to improve access to non-volatile disks

2.2 NVMe Memory Programming Models

Two programming models can adopted for NVMe devices. One approach is to provide a software layer that makes NVMe Device as only SSD disk. This provides the benefits of higher-performance NAND to existing applications created for traditional HDDs or SSDs. But emulating SSDs does not allow software to fully exploit NVMe devises persistency capabilities.

As mentioned above, the applications using disk storage programming models create/modify data in CPU registers, move the data to volatile memory, then move the data to disks for persistence across restarts. In the other approach, PM introduces the option of skipping step 3 if the memory in step 2 is persistent. This has the benefits of avoiding the overhead of moving data between volatile memory and persistent storage, avoiding the overhead introduced when file systems (or applications) adapt application-oriented data sizes to disk block sizes.

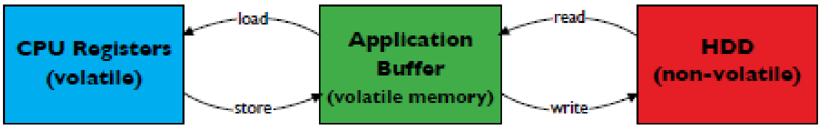


Figure - 6 : Advanced Storage data flow

Applications may need to be rewritten to use this new programming model. Volatile memory is not tied to specific applications; applications allocate memory by size and the run-time environment finds a large enough range of memory and maps it to the application's virtual memory. When the application restarts, the physical address of the allocated memory may be different. In order to use persistent memory, the application needs to access the same data (the same physical address range).. The, applications should not be allowed to access NVMe persistent memory. So new programming models are being defined to allow operating systems to control access to NVMe memory, but bypass extraneous copy-to-disk overhead.

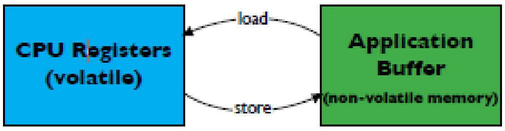


Figure - 7 : Storage data flow in NVMe Devices

3. SYSTEM SETUP AND TESTING

In this work we have identified the algorithms which need to be optimized for using in the latest enterprise server operating systems. Attempted certain kinds of testing to see the possibility of improvements.

During the experimentation , the enterprise servers are equipped with few NVMe and SSD devices. Server OS components are updated with the identified algorithms for achieving the maximum performance. The updated enterprise servers are used for

- Setting up the huge File system,
- Used for Cache device,
- Used for setting up the Database.

The main focus area was , File system. Keen observations are made using different filesystems with modified algorithms for NVMe and SSD devices.

3.1 File System

The experimentation was done on filesystems, XFS and EXT4 for various IO patterns like excusive Read, exclusive Write, Read/Write Combinations. For creating the IO work load iozone tool is used. All normal posix filesystem functionalities are all tested. NFS features are not looked into while testing.

The System configuration and other tools used for the experimentation are as below,

- **System**
    - DELL Enterprise server - PowerEdge R730, 64G
    - Intel(R) Xeon(R) Processor
    - PERC Controller, Seagate 300G 6Gbps SAS
    - Flash NVMe - SSD 400GB
    - Operating system RHEL 7.1

- **System Tunables**
    - # blockdev --getra | --setra
    - Set /sys/block/XXX/queue/rq_affinity to “1”

- **IO Tool and Options Used**

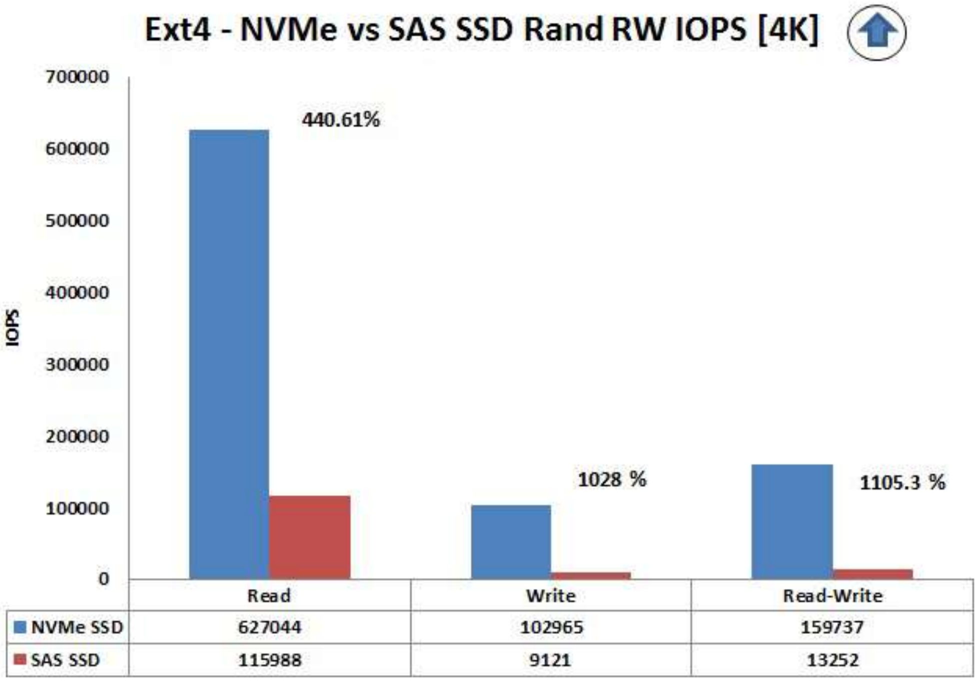
iozone –a –g 4G –b file.xls

a = Auto mode

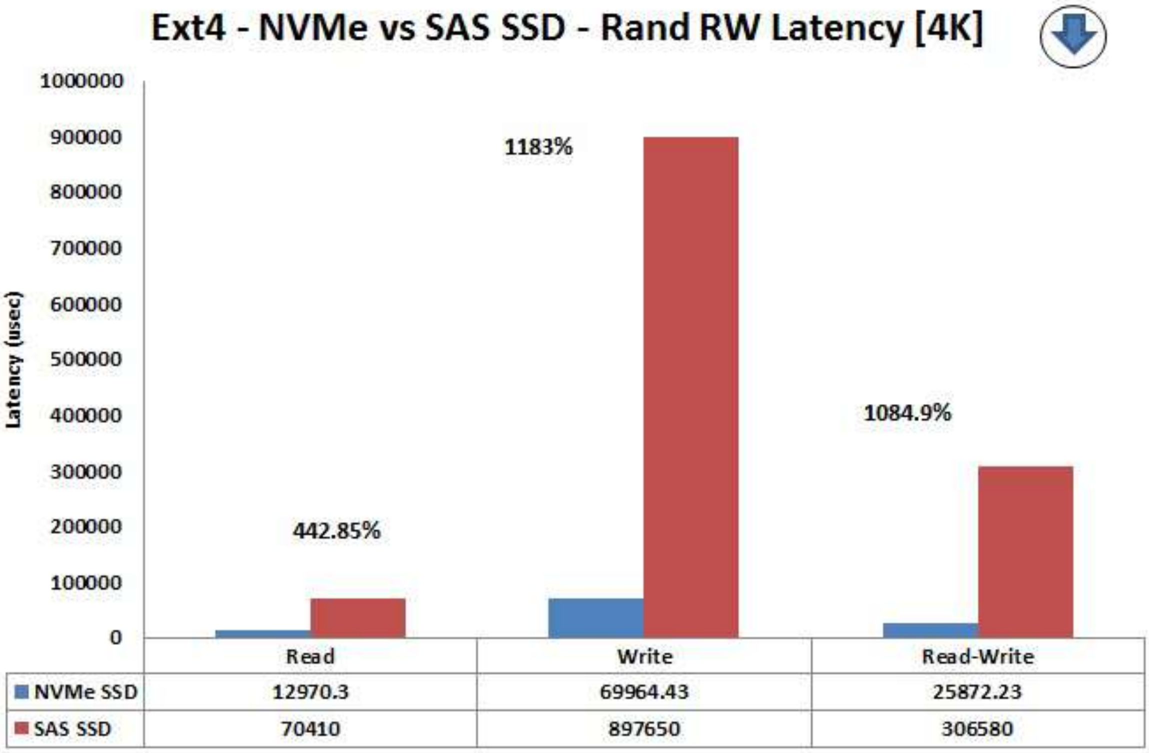
g = Set maximum file size (KB)

b = Filename Create Excel worksheet file

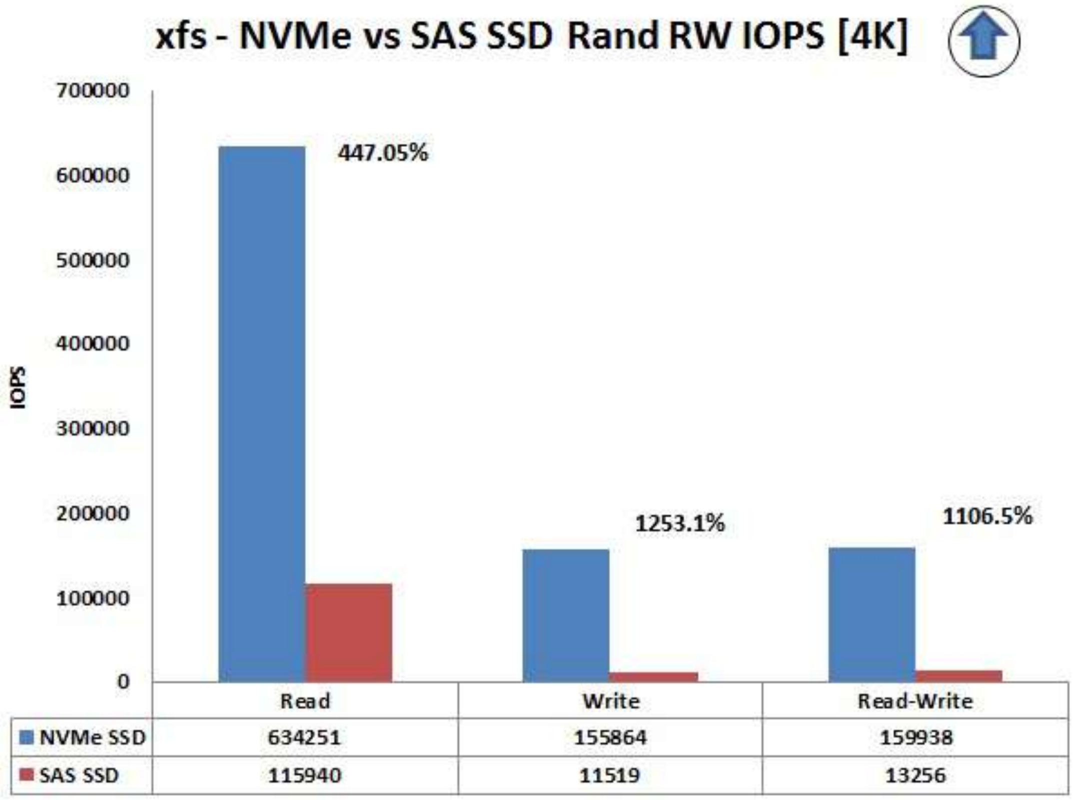
```
nvme_4k_read.fio:
[global]
bs=4k
ioengine=libaio
iodepth=512
size=4g
direct=1
directory=/mnt/nvme
name=nvme_4k_read
numjobs=16
group_reporting
rw=randread
```



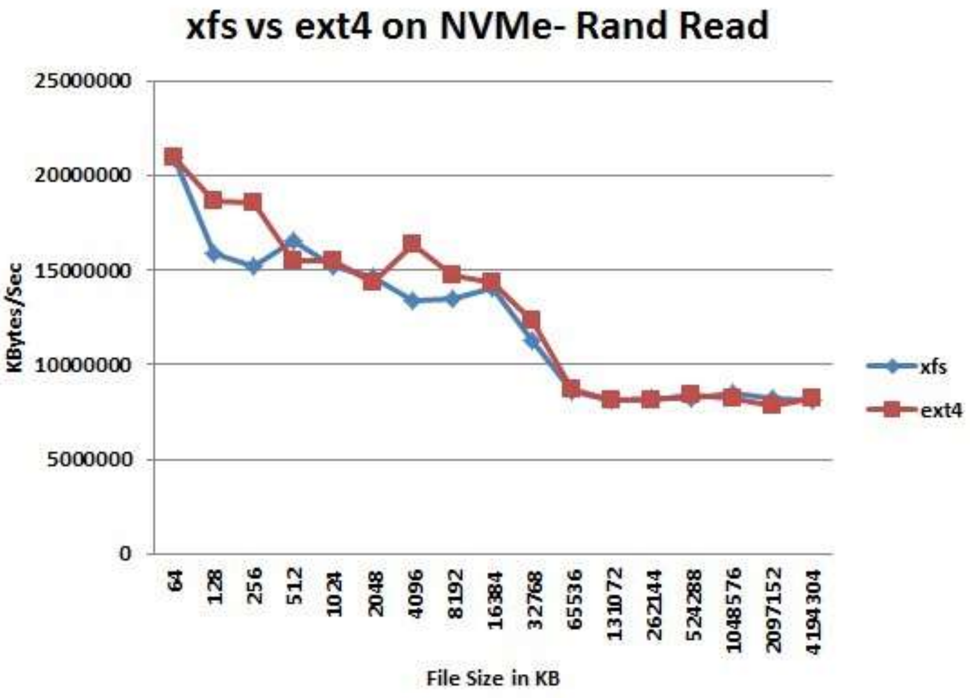
	Read	Write	Read-Write
🟦 NVMe SSD	627044	102965	159737
🟥 SAS SSD	115988	9121	13252

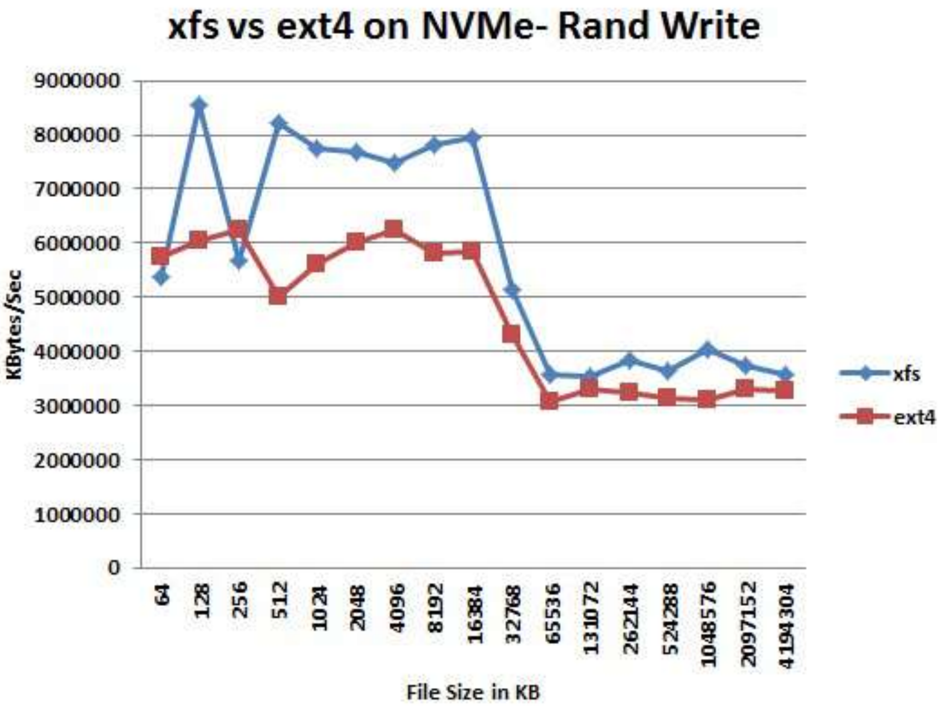


	Read	Write	Read-Write
🟦 NVMe SSD	12970.3	69964.43	25872.23
🟥 SAS SSD	70410	897650	306580



	Read	Write	Read-Write
🟦 NVMe SSD	634251	155864	159938
🟥 SAS SSD	115940	11519	13256





4.2 Cache Device

NVMe devices are great to use as cache or a storage tier. The performance of applications can be increased greatly as NVMe have read/write access times measured in tens of microseconds, compared with low milliseconds for spinning media HDD. As NVMe is flash components it significantly reduces I/O latency for applications and this translates directly into faster application response times or the ability to process more workload. Caching tends to be more responsive to application workload profile changes as active data changes over time. Caching algorithms simply evict in-active data in place of the application workload that has become busy.

During the experimentation flash cache and bcache are used for configuring the cache devices and performed the tests for the functionalities like, copy on write , Full data and metadata check summing, Data writes on multiple devices, replication, RAID features, Data Compression, Encryption, Snapshots etc.

4.3 Database

NVMe drives are expected to be deployed in datacenters. They will be used for I/O intensive applications. Since DBMSs are a prime example of I/O intensive applications for datacenters, this section verifies the performance of several NVMe-based DBMS applications. Both relational and non-relational databases (Cassandra and MongoDB), are experimented using NVMe in multiple configurations. Found that performance of NVMe SSDs gave significant advantage for real-world database applications.

5. CONCLUSION

NVMe memory and devices are getting popular, and cost effective. For latest applications there is a need for high performance storage solutions with high available bandwidth and with lesser access latencies. So there is a need to address this performance gain for the existing systems. To address this need, NVMe storage devices like SSDs would be right choice. In this paper, we present the first, detailed analysis and characterization of SSDs based on the Non Volatile Memory Express (NVMe) standard for storage subsystems, and make a number of important contributions.

Firstly, we show that there is tremendous benefit to be had from re-architecting the existing I/O stack of the operating system. We instrument the Linux storage driver and the popular blktrace tool to quantify the benefits of a I/O stack with NVMe components. Using this methodology, we show that the NVMe access stack allows the I/O requests to bypass most of the legacy I/O layers, resulting in a 4x decrease in system software overheads. Secondly, using benchmarks, we also verify the rated, raw performance of the NVMe SSDs. 4x. Lastly, using databases as an example class of applications. We show that NVMe's hardware and software redesign of the storage subsystem translates into real-world benefits for a number of scale-out file systems. In particular , we show that, as compared to a single SATA SSD,

NVMe based devices can provide performance benefits of up . Furthermore, we show that one NVMe based devices can outperform. Therefore, a better hardware interface, simpler hardware data-path, and a simpler stack, enable NVMe drives leads to outperform for real world filesystems, Databse, and Cache applications compared to SSDs and HDDs. The Filesystem and database application performance benefits would enabled the increased adoption of NVMe SSDs in enterprise and cloud data centers.

References:

1. Adi Fuchs , Ruby B. Lee (2015) Disruptive prefetching: impact on side-channel attacks and cache designs, SYSTOR '15- Proceedings of the 8th ACM International Systems and Storage Conference

2. Performance Analysis of NVMe SSDs and their Implication on Real World Databases Qiumin Xu1, Huzefa Siyamwala2, Mrinmoy Ghosh3, Tameesh Suri3, Manu Awasthi3, Zvika Guz3, Anahita Shayesteh3, Vijay Balakrishnan3
1 Univeristy of Southern California, 2San Jose State University, 3Samsung Semiconductor Inc., Milpitas, CA qiumin@usc.edu, 2huzefa.siyamwala@sjsu.edu,

[1] 3fmrinmoy.g, tameesh.suri, manu.awasthi, zvika.guz, anahita.sh, vijay.balag@ssi.samsung.com

[2] B.S Anami, Vishwanath Burkpalli, Sangamesh.S.H (2005), A Neural Network Based Model for Recognition of Singleton Food Objects, proceedings COGREG 2005 PES Collage Manday, pp419-426.

[3] B.S.Anami, Vishwanath Burkpalli (2009), Color Based identification and Classification of Bulk Food Objects, Proceedings ICSCI2009 Pentagram research center Hyderabad.

[4] Chan, T. F., & Vese, L. A. (2001), Active contours without edges, IEEE Transactions on Image Processing, Volume 10, Issue 2, 266-277.

[5] D. M. Hobson, R. M. Carter, Y Yan (2007), Characterization and Identification of Rice Grains through Digital Image Analysis, Technology conference, IEEE conference proceedings of Instrumentation and Measurement Technology, pp 1-5.

[6] Beveridge, J. R Griffith, J., Kohler, R. R., Hanson & Riseman, e.m. (1989), Segmenting images using localized histograms and region merging. International journal of Computer vision, Volume 2, pp311-347.

[7] Otsu N. (1979), A threshold selection method from gray-level histogram IEEE transactions on Systems, Man and Cybernetics, Volume 9, Issue 1, pp 62-66.

[8] Cheng-Jin Du and Da-wen Sun (2006.), Recent Developments in the applications of image processing for food quality evaluation. FRCFT Group, journal of food science and Technology, Volume 15, pp 230-249.

[9] Hongxu Ni a$nd Sundaram Guansekaran, (2004), “Image processing algorithm for cheese shred evaluation”, Journal of Food Engineering, Volume 61, Issue 1, Pages 37-45.