NVMe Solid State Storage Performance Testing

Vadim Ponomarev,Eugene Pitukhin
Petrozavodsk State University
Petrozavodsk, Russia
vadim@cs.petrsu.ru,eugene@petrsu.ru

***Abstract*—The paper presents the initial stage of constructing a performance model of a modern solid-state drive with a PCIe NVMe interface – a methodology for collecting performance data for existing devices, on the basis of which the model will be built. Approach, implementation and results of implementation of presented methodology are presented. An open source utility is proposed for creating a public NVMe SSD performance data set.**

I. Introduction

In previous works, models of performance indicators for solid state storage systems were presented [1]. However, the performance models were based on draft results, obtained in early experiments. The experiments used early samples of devices from a single manufacturer [2]. All tested devices had SATA interface, which is currently outdated and being superseded by the NVMe interface.

NVMe interface was designed from scratch to utilize low latency and massive parallelism of modern SSDs, processors, platforms and applications. For instance, while SATA has only one command queue per device, NVMe can provide up to 65535 queues per single device. New design allowed to achieve much higher performance.

Thus, constructing a mathematical model of the performance of a solid-state drive with a PCIe NVMe interface become relevant.

The first step in building a model is to get a data set that reflects the necessary characteristics of the object being modeled. A large number of works on NVMe SSD performance are available (for instance, [3], [4] and [5]), but, oddly enough, the authors were unable to find public data sets containing key performance metrics and parameters that affect performance of modern solid-state drives with an NVMe interface. Also, it was not possible to find a free tool which allows you to get a data set with the required characteristics youself.

Therefore, the task of obtaining the data necessary for modeling arises. And before starting the experiments with real devices we need methodology suitable for collecting performance data for solid-state drives with the NVMe interface.

This paper presents such a methodology and test results obtained using the implementation of this methodology developed by the authors.

The structure of paper is the following. Section II describes NVMe architecture and key factors affecting NVMe SSDs performance. These key factors should be saved when measuring performance for later use in modeling. The development of a software tool that implements performance measurement while preserving the key factors is described in section III. Section IV presents the results obtained. Finally, section V summarizes the final remarks and conclusions.

II. Specificity of NVMe SSD performance data and key affecting factors collection

A. High level architecture and background

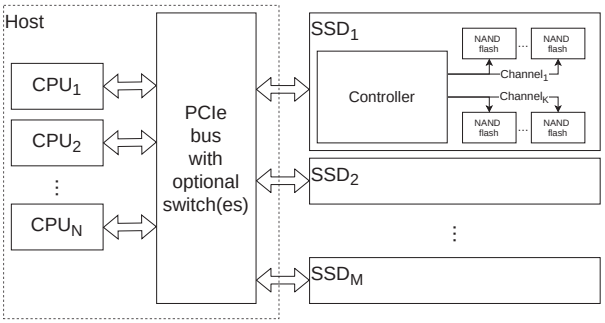

Fig. 1. High-level architecture of computer system with NVMe SSD

High-level architecture of computer system with NVMe solid state drives is presented on Fig. 1. Typical solid state drive has a number of flash memory chips supporting read, write and erase operations. Read and write operations can be done by a blocks with relatively small size (typically 4 to 16 Kb). Erase operation can be done only by a large block (typically a few megabytes) and flash block wearing out by erase operations. Each flash block can be erased only a finite number of times.

Group of flash memory chips is attached to flash memory channel, which is used for interaction and data transfer. Each channel can handle only one data transfer at a time. Howewer, different channels can work simultaneously and provide concurrent access to chip groups attached to them.

Flash memory channels are attached to flash drive controller which can be considered as multi-core embedded computer running complex program and implementing host interface protocol (NVMe/PCIe) and flash chips management. Another important task of flash drive controller is to provide flash translation layer (FTL) routines such as logical block mapping, wear leveling and garbage collection. Unlike legacy storage devices, there is no host bus adapter (HBA) between host and storage device, NVMe SSD controller is directly attached to host's PCIe subsystem.

ISSN 2305-7254

Peripheral Component Interconnect Express, abbreviated as PCIe, provides kind of communcation network for computer system components.

Finally, there are CPUs submitting commands and data to drives. Massive parallelism of NVMe allows to have multiple I/O submission and completion queues, one per each processor core on a single drive. This allows to avoid contention, almost entirely avoid locking and increase performance comparing to legacy SATA drives. A more complete overview of the subject area is available in [6], for instance.

B. Key factors affecting NVMe SSD performance

Taking into account all of the above, we can identify the key factors that affect performance of NVMe SSDs. First is the type (QLC, MLC, TLC, SLC) and other hardware characteristics of NAND flash chips used in drive. This is, for instance, I/O speed, program throughput and read latency.

NAND flash wearout also affects performance. The probability of bit errors increases during the use of the drive, therefore life time, total number of bytes written, and number of erases (if possible) must be recorded and reported with the collected performance data.

Next factor is the number of flash memory channels and their characteristics, such as bus frequency. Number of flash memory channels bounds internal parallelism of a drive. Also, knowing the total number of flash memory chips and number of flash memory channels we can easily determine the number of chips on a single channel.

Next and one of the most important factor is the model of flash drive controller. This factor is very complex and have a large amount of sub-factors. To name a few:

- firmware version
- DRAM memory capacity (if appropriate)
- supported number of flash memory channels
- performance characteristics (maximum throughput for sequential read and write, maximum IOPS for random read and write)

Next factor is the maximum available PCIe bandwidth and amount of NVMe drives sharing that bandwidth. We need to know PCIe topology of the system where drives are installed. Also, during the measurements, it is necessary to ensure that all drives are inactive, except for the tested one.

Another system-wide hardware factors that can affect performance are the model of CPU, model, amount and frequency of RAM, model of system board

Last, but not least hardware factors are the ambient temperature and temperature of the tested drive. SSDs have thermal throttling and even thermal shutdown to avoid damage of the drive, therefore temperature values must be saved and reported with collected performance data.

In addition to hardware, performance can be affected by software factors, such as NVMe driver settings, driver's version and I/O scheduling algorithm used, CPU affinity settings, CPU frequency scaling settings. Other software factors, such as data pattern, I/O pattern and testing algorithm must be uniform for all tests and will be described in the next subsection.

C. SNIA SSS PTS

The Storage Networking Industry Association (SNIA) is a non-profit global organization dedicated to developing standards and education programs to advance storage and information technology. The SNIA has developed methods which enable manufacturers to set, and customers to compare, the performance specifications of Solid State Storage devices, which are evolving with the state of the technology. These specifications define a set of device level tests and methodologies which enable comparative testing of SSS devices for Enterprise and Client systems [7].

Methodology published in SNIA Solid State Storage (SSS) Performance Test Specification (PTS) should be used. It takes into consideration that NAND Flash based solid state storage device performance tends to be highly dependent on the write history of the device. The test methodology defined in the specification attempt to create uniform conditions for write history and the state of the device prior to the test, I/O pattern (read/write mix, block size, access pattern) and data pattern.

The key idea in SNIA SSS PTS methodology is to run tests until so called "steady state" reached. Steady state condition is clearly defined in the document. Shortly, measured metric must not change more than 20% of the average value during measurement window (five test runs), ans the slope of the best linear approximation of measured metric must not be greater than 0.1.

Being a great document, SNIA SSS PTS is not intended for modeling purposes. The methodology proposed in the specification is intended only for the correct comparison of the performance characteristics of various solid-state drive models, and does not require collecting all factors listed in subsection II-B. This paper presents an attempt to extend SNIA SSS PTS methodology to collect all data necessary for later modeling of SSD with NVMe interface.

III. Implementation

A. Available tools

There are at least two public implementation of SNIA SSS PTS. One is [8]. It is implemented using Ansible to run tests and Python to check steady state condition. Unfortunately, this tool does not collect any information other than the `fio` utility results. It is also designed to work only with SATA drives. Using Ansible for such a task is also questionable.

Another available tool is [9]. It is implemented using PHP programming language, intended to use with SATA drives and implements outdated (2013) SNIA SSS PTS Enterprise v1.1. This tool also does not collect all information required for modeling purposes.

And there is a reference commercial SNIA SSS PTS implementation [10]. It is non-free, was not updated since 2015 year (latest news on the company web site), and doesn't look appropriate for scientific purposes.

B. Software requirements

Taking into account all the above, the authors decided to implement their own data collection tool that meets the following requirements:

- Open source code, publicly available to the research community;
- Follow current (version 2.0.2, October 1, 2020) SNIA SSS PTS methodology;
- Support drives with PCIe NVMe interface;
- Record as much data listed in subsection II-B as possible;
- Create both human and machine readable output suitable for modeling purposes;

Openness is obviously required for the ability to collect performance data and key factors listed in subsection II-B by the community. Authors have access only to a limited range of desktop and enterprise computer systems and solid state drives. Creating a data set that covers a large number of computer systems and drive models is possible only with community help.

SNIA SSS PTS methodology is mature, well-developed and need additions only for the modeling purposes (because it was originally intended only for comparing various devices, not for modeling them).

The requirement to support drives with the NVMe PCIe interface arose due to the fact that such drives are currently both modern, publicly available. and widely used in desktop and enterprise systems. Support of drives with legacy SATA interface is surely possible too, but from the modelling point of view SATA SSD drives a not very interesting, since the share of such disks is rapidly decreasing. There are also various forms of persistent solid state memory [11], but they are not yet currenly available for mass consumer.

The requirement to collect as much factors as possible is due to the fact that before the data actually collected and analyzed we don't currently know what impact will each of the listed factors have.

And the last requirement in the list is readyness: collected data must be easy to load by programs. Manual data entry is obviously inappropriate. Data must be saved in machine-readable format, for example, JSON or XML, to name a few. From the point of view of human readability, JSON looks more preferable.

C. Implementation

Only freely distributed software components were used in the implementation. The main utility used for management of NVMe devices is `nvme`. It has various sub-commands, of which the following are used for our purposes:

- `list`: list all NVMe devices and their parameters on machine;
- `id-ctrl`: retrieve information about SSD controller;
- `get-feature` and `set-feature`: get/set feature and show the resulting value;
- `format`: purge device and return it to "fresh-out-of-the-box" (FOB) state;
- `smart-log`: Retrieve various sensor and monitoring values (for example, device temperature, read and write counters).

Fortunately, `nvme` supports JSON output format. An example of `nvme list` command output follows:

```
{
  "Devices" : [
    {
      "Subsystem" : "nvme-subsys0",
      "SubsystemNQN" :
          "nqn.2014.08.org.nvmexpress:...",
      "Controllers" : [
        {
          "Controller" : "nvme0",
          "Transport" : "pcie",
          "Address" : "0000:01:00.0",
          "State" : "live",
          "Firmware" : "2B2QEXM7",
          "ModelNumber" :
              "Samsung SSD 970 EVO Plus 500GB",
          "SerialNumber" : "...",
          "Namespaces" : [
            {
              "NameSpace" : "nvme0n1",
              "NSID" : 1,
              "UsedBytes" : 0,
              "MaximumLBA" : 976773168,
              "PhysicalSize" : 500107862016,
              "SectorSize" : 512
            }
...
```

Listing 1. `nvme list` output example

We can see PCIe address `0000:01:00.0` allowing to link NVMe device with PCIe topology and other information obtained by `lspci` utility.

Low level information, such as number and type of flash memory chips, controller performance characteristics and amount of DRAM must currently be collected and provided manually. Hopefully there will be a database allowing to get low-level device parameters by device model. This task is impossible without community support.

The `lspci` is the next building block. This utility allows to get information about PCIe topology and devices connected with characteristics such as connection width (number of lanes), bandwidth and other parameters.

Generic information about computer system, such as CPU model and characteristics, number and model of installed DIMMs and other similar information can be obtained by `lshw` utility. It also supports JSON output format.

The main utility used for performance tests is `fio` [12]. It is de-facto standard for Linux block device testing and performance measurements. The utility provides a large number of features. Typical usage scenario is to write so called "job file", containing description of various global and "job" parameters, and run `fio` with job file as a parameter. An example of job file follows:

```
[global]
ioengine=libaio
iodepth=16
```

```
direct=1
gtod_cpu=1
thread
group_reporting

random_distribution=random
random_generator=tausworthe
allrandrepeat=1
randseed=3735928559

filename=/dev/nvme0n1
numjobs=2
size=183144969k
rw=randrw

[wdpc-rr65-4k]
stonewall
runtime=1m
time_based
rwmixread=65
bs=4k
```

Listing 2. `Fio` job file example

Among other parameters, the following are set:

- `iodepth` corresponds to "Outstanding IO (OIO)" in SSS PTS: the number of IO operations issued by a host awaiting completion;
- `randseed`: seed value for random number generator;
- `numjobs` corresponds to "Thread Count (TC)" in SSS PTS: the number of Threads (or Workers or Processes) specified by a test;
- `rw` and `rwmixread`: specifies I/O pattern, `rw` value `randwr` means "random mixed reads and writes", where `rwmixread` is the percentage of a mixed workload that should be reads;
- `runtime` limits the time of test execution, with the option `timebased` means "run for a specified amount of time";
- `bs` is the I/O block size

The `fio` utility supports JSON output format. Output file contains input parameters taken from job file and collected data for read, write and trim I/O operations. Data contains aggregated information, such as mean bandwidth and IOPS during job run, and latency histogram.

And finally we need some program to glue all mentioned building blocks together. It must be written in high-level programming language with ability to launch external programs, do some pattern matching, have statistics modules to calculate mean values and linear regression for steady state check. Nowadays the most appropriate programming language for such set of a tasks is Python. Program was written implementing adapted SNIA SSS PTS methodology with the Alg. 1.

The difference from the SNIA SSS PTS methodology is that after each WDPC run, parameters that change during testing, such as temperature and read/write counters, are collected and saved for future use in modeling. The proposed change will allow taking into account, for example, the heating of the drive during testing. The amount of stored information about the

Algorithm 1 IOPS performance test algorithm

procedure SAVE($data$)
▷ Various helper functions
end procedure
procedure WIPC
▷ Workload-independent pre-conditioning: Run `fio` to overwrite the device with random data two times
end procedure
procedure WDPC($rwmix, bs$)
▷ Workload-dependent pre-conditioning: Run `fio` with given I/O pattern $rwmix$ and block size bs one minute
end procedure
procedure STEADYSTATEREACHED(y, w)
▷ Steady state as defined in SNIA SSS PTS v2.0.2 2.1.24 (page 17): for the last w values of y:
a) $Range(y) < 0.2 * Ave(y)$
b) $Slope(y) < 0.1$ for the best linear curve fit of the y
end procedure
SAVE(Generic computer system information)
SAVE(PCIe information)
SAVE(NVMe information)
SAVE(System I/O scheduler settings)
SAVE(NVMe driver version and settings)
SAVE(Temperature sensor data and monitoring counters)
$Results^{4}_{0/100} \leftarrow \varnothing$
$Results^{64}_{65/35} \leftarrow \varnothing$
$Results^{1024}_{100/0} \leftarrow \varnothing$ ▷ Initialize with empty list
PURGEDEVICE ▷ Call `nvme format`
WIPC ▷ Run workload-independent pre-conditioning
repeat
  for all $rwmix \in 100/0, 95/5, 65/35, 50/50, 35/65, 5/95, 0/100$ **do** ▷ R/W mix %
    for all $bs \in 1024, 128, 64, 32, 16, 8, 4, 0.5$ **do** ▷ block size in KiB
      $Result \leftarrow$ WDPC($rwmix, bs$)
      SAVE($Result$)
      SAVE(temperature sensor data and monitoring counters)
      if $(rwmix = 0/100) \wedge (bs = 4)$ **then**
        APPEND($Results^{4}_{0/100}, Result$)
      else if $(rwmix = 65/35) \wedge (bs = 64)$ **then**
        APPEND($Results^{64}_{65/35}, Result$)
      else if $(rwmix = 100/0) \wedge (bs = 1024)$ **then**
        APPEND($Results^{1024}_{100/0}, Result$)
      end if
    end for
  end for
until (STEADYSTATEREACHED($Results^{4}_{0/100}, 5$)
  $\wedge$ STEADYSTATEREACHED($Results^{64}_{65/35}, 5$)
  $\wedge$ STEADYSTATEREACHED($Results^{1024}_{100/0}, 5$))
  $\vee$ (25 rounds passed)

system is also significantly expanded in comparison with the recommendations of SNIA SSS PSS v2.0.2 [7].

At the time of writing this text, program source code size was 27 Kb (807 lines of code).

The program is currently available for review, tests and contributions on GitHub [13]. The authors hope that the program will be further developed and it will be possible to obtain performance testing results for various models of drives (with the help of community).

IV. Measurements

A. Common metrics

This section presents quick view on the first results obtained. Test system was a desktop PC with AMD Ryzen 5 3600X processor, Asus TUF B450-PRO GAMING system board. 32 Gb RAM installed. Test drive was Samsung SSD 970 EVO Plus 500GB (NVMe in M.2 form factor) installed in socket with PCIe 3.0 x4 support. Operating system was openSUSE Leap 15.2 Linux distribution (kernel version 5.3.18 with openSUSE patches).

Each round takes $7 * 8 = 56$ minutes (one minute `fio` run for seven values of R/W mix % and eight values of block size, as specified by Alg. 1). During each `fio` run three files are generated:

- JSON file with `nvme smart-log` result (temperature sensor values, read/write counters);
- Text file with `fio` job;
- JSON file with `fio` results.

There must be at least five rounds to reach the steady state, but typically after the first round the performance metrics change significantly and the steady state is reached on the sixth round ($R = 6$ on figures). Therefore, typically, a minimum of $56 * 6 = 336$ minutes (5 hours 36 minutes) required to reach steady state and 1008 files produced.

SNIA SSS PTS methodology for IOPS test requires to track steady state for the following I/O patterns and I/O block sizes:

- R/W mix % = 0/100, I/O block size = 4KiB;
- R/W mix % = 65/35, I/O block size = 64KiB;
- R/W mix % = 100/0, I/O block size = 1024KiB.

Figures 2, 4 and 3 shows IOPS for three fixed values of block size used in steady state tracking (4Kb, 64Kb, 1024Kb) and all available I/O patterns (random R/W with mix % values 100/0, 95/5, 65/35, 50/50, 35/65, 5/95, 0/100).

Without any surprises, the highest mean IOPS value (more than 160k IOPS) was shown for 4Kb block size and read operation (R/W mix % 100/0 on Fig. 2). For all three block sizes, the lowest IOPS value was achieved when R/W mix % was 35/75 (surprisingly not 0/100).

Figures 5, 6 and 7 shows IOPS for three fixed values of R/W mix % used in steady state tracking (0/100, 65/35 and 100/0) and all available block sizes (1024 Kb, 128 Kb, 64 Kb, 32 Kb, 16 Kb, 8 Kb, 4 Kb and 512 bytes).

Steady state was reached on the sixth round. There is clear performance decreare after first workload-dependent pre-condition (WDPC) run.

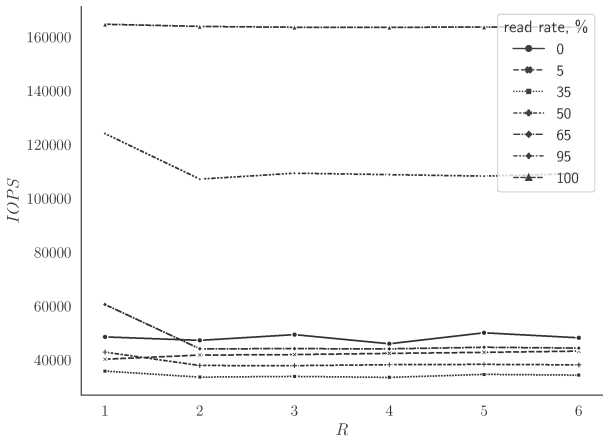

Fig. 2. Block size 4k IOPS

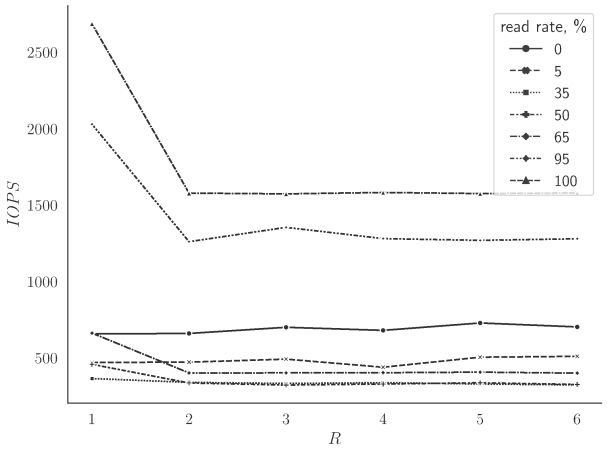

Fig. 3. Block size 1024k IOPS

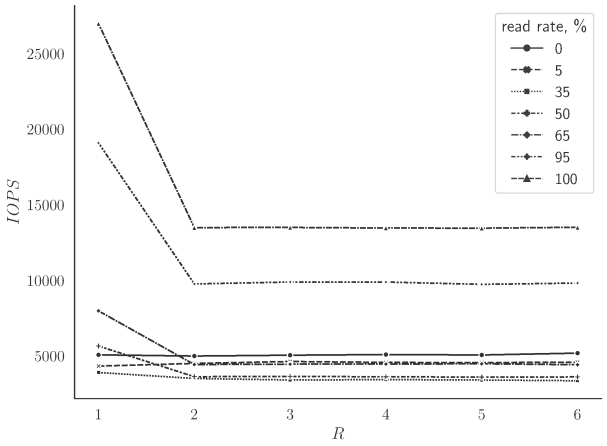

Fig. 4. Block size 64k IOPS

V. Conclusion

This paper continues work on performance models of solid state drives started in [1]. Drives with a modern PCIe NVMe

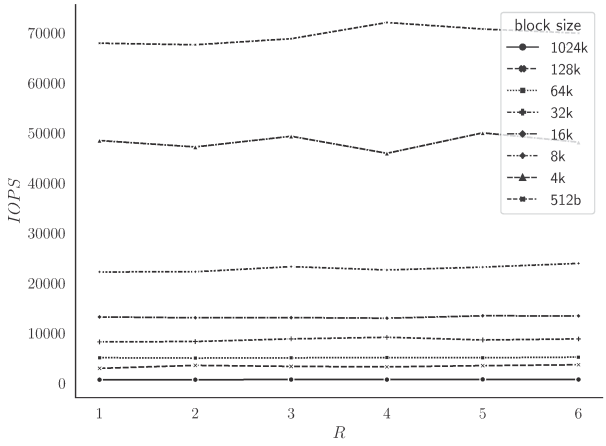

Fig. 5. R/W Mix 0/100 IOPS

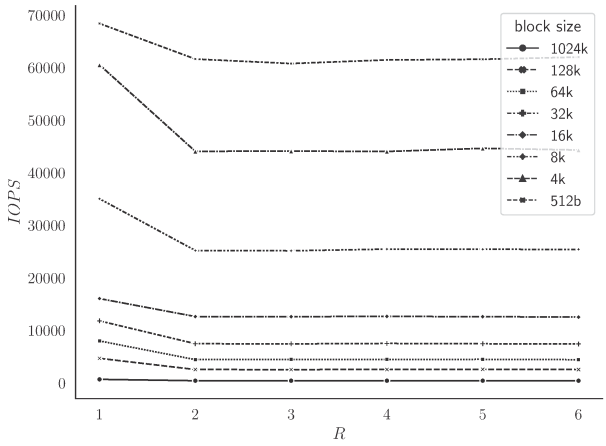

Fig. 6. R/W Mix 65/35 IOPS

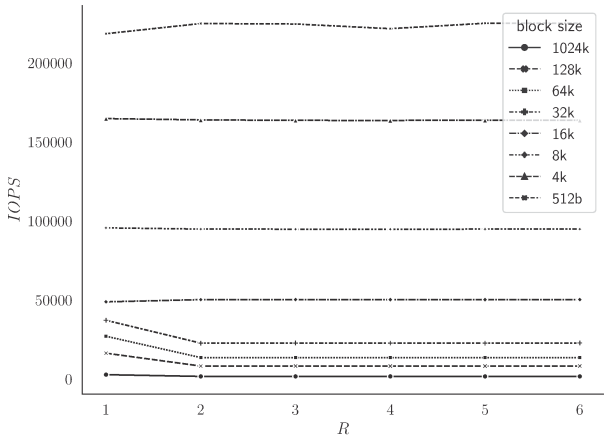

Fig. 7. R/W Mix 100/0 IOPS

interface are considered instead of legacy SATA interface drives. The first step to a more accurate modeling has been made: a methodology for obtaining performance data of devices has been developed (based on SNIA SSS PTS v2.0.2). A program was developed that implements this methodology. The program is open source and is available to the community [13].

Our further work is to continue the development of this software utility (implement more metrics) and make a public data set containing performance data for a bigger number of PCIe NVMe drives and various test computer systems (hopefully with the help of the community). The obtained data set is then planned to be used to create a model of an NVMe drive.

REFERENCES

[1] V. A. Ponomarev, "Simulation modeling performance indicators for solid state storage systems," *Programmnaya Ingeneria*, vol. 10, no. 9–10, pp. 367–376, Aug 2019, in Russian. [Online]. Available: http://doi.org/10.17587/prin.10.367-376

[2] K. A. Ekimov, S. F. Podryadchikov, V. V. Putrolaynen, M. A. Belyaev, and E. I. Maslennikov, "Testing experimental samples of solid state drives," *IOP Conference Series: Materials Science and Engineering*, vol. 537, p. 032042, jun 2019. [Online]. Available: https://doi.org/10.1088/1757-899x/537/3/032042

[3] Q. Xu, H. Siyamwala, M. Ghosh, T. Suri, M. Awasthi, Z. Guz, A. Shayesteh, and V. Balakrishnan, "Performance analysis of nvme ssds and their implication on real world databases," in *SYSTOR '15: Proceedings of the 8th ACM International Systems and Storage Conference*, 05 2015, pp. 1–11. [Online]. Available: https://doi.org/10.1145/2757667.2757684

[4] Y. T. Jin, S. Ahn, and S. Lee, "Performance analysis of nvme ssd-based all-flash array systems," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 04 2018, pp. 12–21. [Online]. Available: https://doi.org/10.1109/ISPASS.2018.00010

[5] Y. Son, H. Kang, H. Han, and H. Yeom, "An empirical evaluation and analysis of the performance of nvm express solid state drive," *Cluster Computing*, vol. 19, 09 2016. [Online]. Available: https://doi.org/10.1007/s10586-016-0591-8

[6] M. Torabzadehkashi, S. Rezaei, A. HeydariGorji, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Computational storage: an efficient and scalable platform for big data and hpc applications," *Journal of Big Data*, no. 6, 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0265-5

[7] E. Kim, A. Jones, M. Fausset, E. Ho, D. Landsman, and J. Thatcher, *Solid State Storage Performance Test Specification v2.0.2*. [Online]. Available: https://www.snia.org/sites/default/files/technical_work/PTS/SSS_PTS_2.0.2.pdf

[8] J. Liu, *Solid State Storage (SSS) Performance Test Specification in Ansible*. [Online]. Available: https://github.com/ljishen/SSSPT

[9] *Block storage test suite based on SNIA's Solid State Storage Performance Test Specification Enterprise v1.1*. [Online]. Available: https://github.com/cloudharmony/block-storage

[10] *Calypso Systems, Inc. - SNIA SSSI Certified Test Lab*. [Online]. Available: https://calypsotesters.com/about/

[11] *Persistent Memory Special Interest Group*. [Online]. Available: https://www.snia.org/forums/cmsi/NVDIMM

[12] J. Axboe, *Flexible I/O tester*. [Online]. Available: https://fio.readthedocs.io/en/latest/fio_doc.html

[13] V. A. Ponomarev, *Python implementation of SNIA Solid State Storage (SSS) Performance Testing Specification (PTS)*. [Online]. Available: https://github.com/ccrssaa/py-sss-pts