On the Impact of I/O Access Patterns on SSD Storage

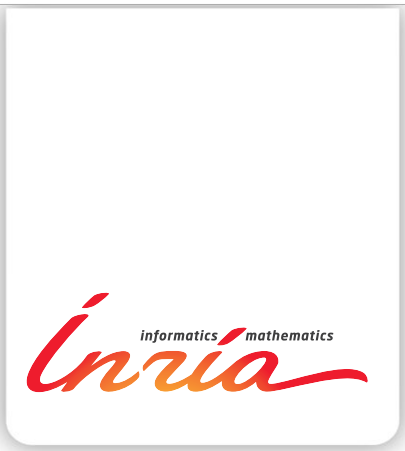
Abdulqawi Saif, Lucas Nussbaum, Ye-Qiong Song

On the Impact of I/O Access Patterns on SSD Storage

Abdulqawi Saif, Lucas Nussbaum, Ye-Qiong Song

RESEARCH REPORT
N° 9319
January 2020
Project-Team RESIST

ISSN 0249-6399 ISRN INRIA/RR--9319--FR+ENG



On the Impact of I/O Access Patterns on SSD Storage

Abdulqawi Saif*, Lucas Nussbaum*, Ye-Qiong Song*

Project-Team RESIST

Research Report n° 9319 — January 2020 — 12 pages

Abstract: Nowadays, solid-state drives (SSDs) are increasingly used in storage infrastructures thanks to their promising performance compared with hard disk drives (HDDs). Their internal architecture consists of parallel storage units that do not use mechanical movements, so considerably fortified for random data access. However, in this report, we show that some SSD storage is sensitive to I/O patterns, achieving low I/O performance in case of running random workloads. The performance degradation occurs with workloads that have specific characteristics such as having only one I/O process and I/O requests with small block sizes. After many investigations on the I/O stack and inside the SSD storage, our results show that the internal `readahead` of SSDs is behind the performance gap between the sequential and random workloads. Indeed, activating the internal `readahead` incurs a performance increase in case of sequential access thanks to pre-reading the next amount of data into the cache. In contrast, this makes no sense for random workloads as the upcoming requests may not depend on what already stored in the cache.

Key-words: IO patterns, solid-state drives, random workaloads, performance issues

* Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

RESEARCH CENTRE NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Une Étude sur l'impact des Patterns d'Accès d'Entrée/Sortie sur le Stockage SSD

Résumé : De nos jours, les disques SSD sont de plus en plus utilisés dans les infrastructures de stockage grâce à leur performance prometteuse par rapport aux disques durs. Ils ont des unités de stockage parallèles qui n'utilisent pas de mouvements mécaniques, donc fortement immunisés contre les workloads aléatoires. Cependant, dans ce rapport, nous montrons que certains modèles de disques SSD atteignent des performances moyennes avec des workloads aléatoires. Nous réalisons une série d'expériences utilisant une workload émise par un seul processus et ayant des petites requêtes d'entrée/sortie afin d'étudier la sensibilité des disques SSD aux patterns d'accès d'E/S. Le résultat montre que le `readahead` interne des disques SSD est le responsable principal de l'écart de performance de SSD entre l'utilisation de workloads séquentielles et aléatoires. Il augmente seulement la performance des workloads séquentielles et il n'affecte pas la performance des workloads aléatoires puisque les données lues en anticipation ne sont pas utiles et sont probablement effacées à chaque arrivée d'une nouvelle requête d'entrée/sortie.

Mots-clés : Pattern d'Entrée/Sortie, Disque SSD, charge de travail aléatoire, problème de performance

1 Introduction

I/O operations are centric for big data systems where an enormous quantity of data should be accessed regularly. Maintaining an acceptable I/O performance during data access means dealing with many challenges not only over the I/O stack but also including the storage devices themselves. Because hard disk drives (HDDs) still appear in today's infrastructure, many improvements are made in the I/O stack to strengthen their performance. Indeed, tuning some parameters on the I/O stack such as activating a different I/O scheduler can mitigate the performance impacts of HDDs to some extent, giving a small increase of IOPS. However, this is seen by the perspective of nowadays' I/O workloads as a minor achievement, which indirectly pushes towards having more robust storage devices. Certainly, having a new generation of storage devices that eliminate the mechanical movements of HDDs would be a sound solution.

Solid-state drives (SSDs) achieve higher performance than HDDs thanks to their complete change in design and architecture. They use performant storage units such as NAND flash memories for storing data, decreasing the performance gap between accessing data stored on RAM and secondary storage. However, the variety of I/O workloads makes it challenging to leverage the potential of SSDs for all use cases. On the one hand, many I/O workloads still have features that were useful to improve accessing data on HDDs, e.g., having I/O block sizes around 4 KB which is the size of memory pages in Linux. Doing so on SSDs does not make sense as SSDs have an internal page size between 4 KB and 4 MB. Hence, accessing data with small block sizes may nevertheless incur performance issues. On the other hand, SSD controllers are still black boxes, so extracting their internal configurations to adjust the workload characteristics accordingly is not feasible. Of course, methods such as extracting information from SSD behaviors [4] might be useful, but trying to do so every time having a different workload is exhaustive. Moreover, systems such as big data storage do not provide support about choosing suitable SSDs for their workloads. Given that, performance issues will occur sooner or later if workloads do not take into account the particularities of underlying storage devices.

Delivering a comparable performance for sequential and random workloads is one of the promising claims of SSDs. Since the majority of SSD devices rely on memory-like storage units, storing data sequentially and randomly should not provoke any issues. Indeed, the storage units take almost the same time to store the data in adjacent or far-off storage cells as there is no meaning of adjacency or contiguous allocation on SSDs. However, in a recent study that evaluates the performance of a NoSQL database [5], we observed that the SSD storage in use reports a delayed execution time in case of running random workloads. The experiments' workload has been initiated by only one I/O process that frequently accesses massive data using I/O requests with small size. Taking into account the fact that such a workload does not exploit the parallelism of the used SSD, reporting a degraded performance for random workloads is not logically expected. Hence, we are motivated to do further investigations to explain the leading causes behind this unusual behavior.

In this report, we perform a study over SSD storage to reveal the responses to I/O pattern changes. We perform several experiments that take into account the workload characteristics

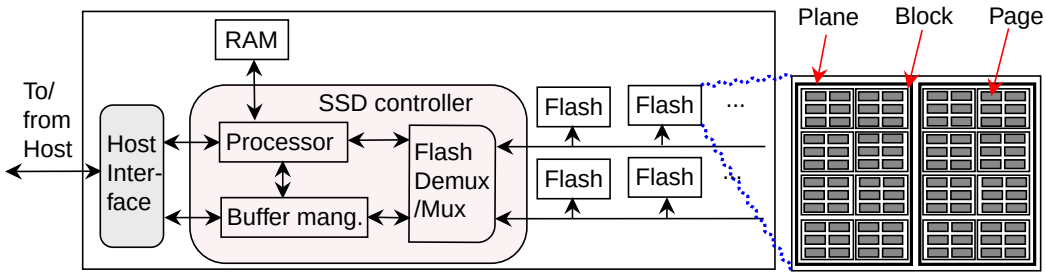


Figure 1: SSDs design [1] with minor adjustment to show the components of a NAND flash

of the study described in [5]. Several potential areas that may affect the I/O performance are investigated here, trying to understand what pushes the SSDs understudy to report moderate I/O performance for random workloads. These areas are either related to 1) the workload itself such as the size of I/O requests and the concurrency of I/O processes, 2) the I/O stack such as the impact of I/O schedulers, 3) or related to the internal parameters of the storage devices themselves such as the internal `readahead`.

The rest of this report is organized as follows. Section 2 briefly describes the internal structure of SSDs. Section 3 presents the initial benchmarks that reveal the impacts of I/O patterns on SSDs, while Section 4 describes the experiments performed to investigate that issue. Section 5 concludes.

2 SSD internal design

Figure 1 shows the essential components of a *NAND SSD*. It consists of an interface to communicate with hosts' I/O stack, a controller with a CPU, buffer manager and RAMs, and a set of *NAND* memories connected to the SSD controller via a multiplexer. When an I/O request arrives at the host interface, the controller processes that request, and then determine the target *flash NAND* for writing or reading data.

Three operations can be performed on an SSD (read, write and erase) with different granularities: erase operations can be done on blocks while reading or writing operations work on the page level. Updating the stored data results on using new memory pages and marking the previous copy of data as a stale to be erased in the next erase cycle. Generally, SSD controllers try to reduce the erase cycles by spreading the data into more *NAND* memories as possible.

Data is stored in memory cells. Three types of memory cells can be used in SSD design. Single level cell (*SLC*) where each cell only stores one bit of data. Multiple level cell (*MLC*) where each cell has four different states, so storing two bits, and triple level cells (*TLC*) for storing three bits per cell. Although having the possibility to store more data using *MLC* or *TLC NAND flashes*, their write/erase cycles are limited compared with *SLC* SSDs.

Table 1: Description of three SSD models used in the initial experimentation phase

SSD model	SSD Type	Size	Optimized for	manufacturer	Year
PX02SSF020	MLC	200 GB	High-endurance & write-intensive	Toshiba	2017
PX05SMB040	MLC	400 GB	High Endurance	Toshiba	2016
C400-MTFDDAA064M	MLC	64 GB	—	Micron	$<$ 2008

Table 2: Results of sequential and random reading workloads on three models of SSD

SSD model	Workload	4 KB block size	28 KB block size
PX02SSF020	Read	43.2 MB/s	172 MB/s
	Rand. Read	31 MB/s	148 MB/s
PX05SMB040	Read	136 MB/s	491 MB/s
	Rand. Read	40.1 MB/s	220 MB/s
C400-MTFDDAA064M	Read	61.9 MB/s	143 MB/s
	Rand. Read	25 MB/s	75 MB/s

Finally, SSDs controller sends one command per *NAND flash* [2] while planes in each *NAND* support parallel executions during data writing or data retrieval. Sending one command at a time indicates that if a file's data is allocated on the same *NAND flash*, the performance may be degraded since the data access will be done sequentially over that *NAND*.

3 Performance of Random Workload on SSDs

After analyzing the workload of the experiments presented in [5], we found that only one process performs the I/O activities of the executed workload. That process sends near 93% of I/O requests with 28 KB as a block size. The requests are sent directly to the storage device (synchronous I/Os) via `pread syscall`.

Although the internal architecture of SSDs can be different from an SSD to another one, we expect to have different performance results accordingly. Hence, we perform our initial experiments to reproduce the experiments of [5] on three SSD models located in the *Grid'5000* testbed [3]. These models are described in table 1.

We use *Fio* benchmark (v 2.16) to reproduce a similar workload. We create one process to mutually perform reading or writing operations over a file of 40 GB of data. Our experimental factors include two block sizes (4 KB & 28 KB) and two access modes (sequential & random). The experiments are performed on a machine running *debian9* with *Linux 4.9.0*

Table 3: Results of sequential and random writing workloads on three models of SSDs

SSD model	Workload	4 KB block size	28 KB block size
PX02SSF020	Write	88.2 MB/s	312 MB/s
	Rand. Write	64 MB/s	292 MB/s
PX05SMB040	Write	123 MB/s	496 MB/s
	Rand. Write	99.2 MB/s	470 MB/s
C400-MTFDDAA064M	Write	58 MB/s	97 MB/s
	Rand. Write	38 MB/s	86 MB/s

and *ext4* as a filesystem. The experimental factors are added to the *Fio* command or script. For instance, a *Fio* command to specify one I/O process for performing a sequential reading over a file of 35 GB and an I/O block size of 4 KB is as follows:

```
fio --name TEST1 --filename=test.img --rw=read --size=35G  --ioengine=sync
--iodepth=64 --blocksize=4k --direct=1  --numjobs=1 --group_reporting
```

Tables 2 & 3 show the results of reading and writing experiments on the SSDs described in table 1. For both experiments, increasing the block size of I/Os from 4 KB to 28 KB leads to an increase in the performance according to the SSD model. However, changing the access mode from sequential to random has a significant effect on the reading experiments; the throughput is degraded by half for some studied SSD models (*PX055MB040* and *C400*) for both block sizes. Similarly, the writing performance is affected by switching the access mode between sequential and random. However, the performance difference does not exceed 30 MB/s which is not comparable with reading experiments where the access mode influences more the performance.

The next section describes the experiments performed to understand why the studied SSDs show degraded performance for random workloads.

4 Experimentation

Determining the primary cause behind the sensibility of certain SSDs to the I/O patterns is not easy since it can be related to several hypotheses. The reason could be either related to 1) the characteristics of the used workload (e.g., small I/O block size), 2) the configuration of I/O stack since some layers such as I/O scheduler can merge, sort, and/or delay the I/O requests for optimizing the performance, 3) the fact of having only one I/O process, or 4) the internal behaviors of the SSDs in use as they are not all impacted by random workloads (e.g., *PX02SSF020* model). That I/O pattern sensibility could also be related to all these points together. We narrow our investigation scope to experimenting only on the *PX05SMB040* SSD model since it is the most affected SSD by random reading workloads (see table 2).

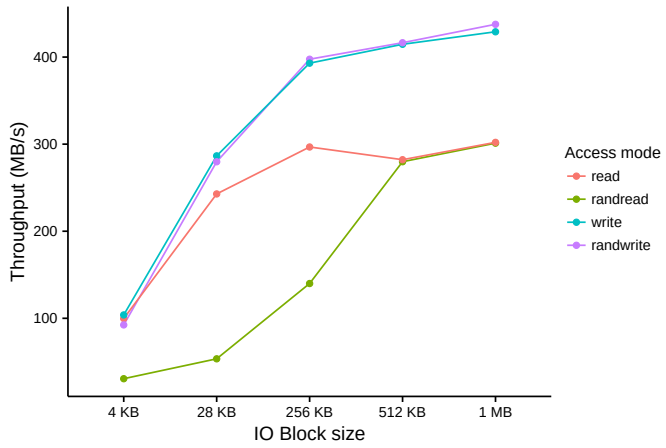


Figure 2: Varying I/O block size over the *PX05SMB040* SSD

All results of the experiments described in the following subsections are an average of ten executions.

4.1 I/O block size experiments

Experiments that vary the I/O block size between 4 KB and 1 MB are performed. Figure 2 shows the results of these experiments. One can see that the throughput of the writing workloads is not affected if the data is accessed sequentially or randomly. It just increases by increasing the block size because larger block size means a larger quantity of data written by each I/O request. However, the impact of random workloads is present during the reading experiments. The gap between the throughput of sequential read and random read is shown on the block size points from 4 KB to 512 KB included. Using larger block sizes than 256 KB eliminates the gap of reading data sequentially or randomly. However, these results do not bring answers to understand why that gap is still present with smaller block sizes.

4.2 I/O scheduler experiments

We perform experiments using several I/O schedulers to identify if some of them provoke the performance degradation for random workloads. Linux proposes several I/O schedulers that are basically invented to minimize the HDD's internal seeks as the schedulers are nothing but algorithms to organize and reorder I/O requests. These schedulers are *noop*, *deadline*, and *complete fair queuing (CFQ)* ordered by their level of complexity from the simple to the most complex scheduler that creates several I/O queues per process. Although these schedulers are still usable with SSDs despite their HDD-related goals, I/O frameworks such as block multi-queue (aka *blk-mq*) are developed to optimize the I/O for SSDs. *blk-mq* completely

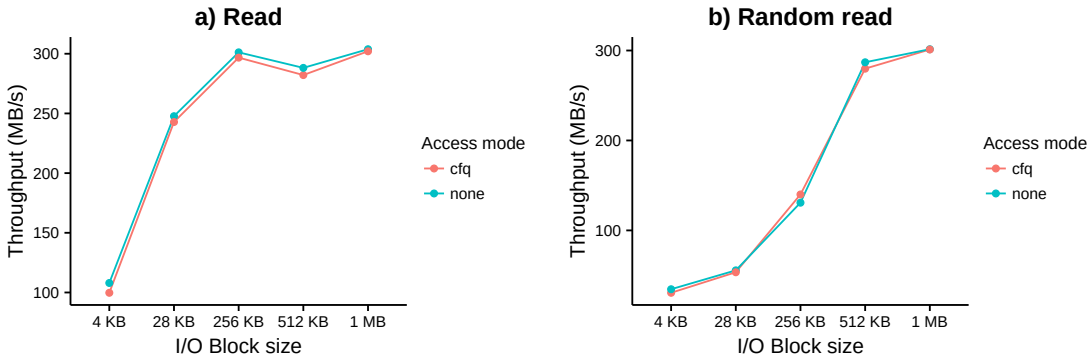


Figure 3: Results of using CFQ scheduler and *blk-mq* with no scheduler over the *PX05SMB040* SSD

bypasses the old I/O block layer and do not have any I/O scheduler so far, relying on the internal schedulers of underlying SSDs. In our experiments, we use the notation of Linux (*none*) to describe the case when *blk-mq* is used. The experiments are performed over the traditional I/O schedulers of I/O block layer (*noop*, *deadline*, and *CFQ*) as well as over *none* of *blk-mq*.

Figure 3 only shows the result of using CFQ scheduler and *blk-mq none*. Indeed, the results of *noop* and *deadline* are completely comparable with *CFQ* results due to the usage of only one I/O process in workload execution. In that figure, one can see that varying the I/O schedulers do not have any impacts on the performance as the curves of *CFQ* and *blk-mq none* go hand in hand on both sub-figures. Additionally, these results do not explain the gap between the sequential and random workloads for small block I/O sizes. For instance, the performance of reading data using a block size of 28 KB is different for sequential and random access. It is near 250 MB/s in case of sequential reading while its less than 50 MB/s in the case of random reading.

4.3 Concurrent I/Os

We narrow our investigation scope to include block sizes between 4 KB & 64 KB since previous experiments show that the issue of random workloads occurs while dealing with small data sizes. We perform experiments with a different number of concurrent jobs, trying to understand if the issue is persistent regardless of the simultaneous access to disks' data.

Figure 4 shows the results of varying the number of jobs that access the SSD at the same time. Every job consists of an I/O process that tries to read sequentially or randomly its own 4 GB file. From a, b, and c sub-figures, it is clearly shown that the performance gap between the sequential and random reading workloads occurs when having only one I/O process. Increasing the number of concurrent jobs means sending many I/O requests at a time,

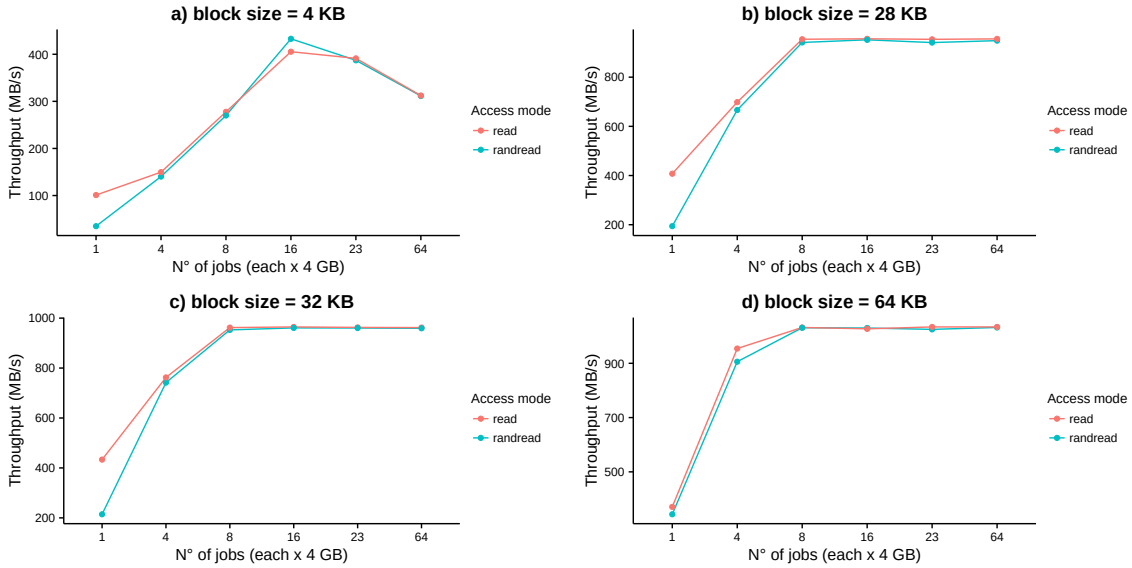


Figure 4: Results of using concurrent Fio's jobs over the *PX05SMB040* SSD

exploiting the parallelism of the SSD and eliminating the impact of random access. Indeed, having a large number of jobs indicates that the SSD controller fetches more data each time it accesses the NAND flashes as several files are read simultaneously. Additionally, the larger the block size, the larger the throughput. For instance, the corresponding throughout of 16 jobs in the sub-figures a, b, and c are 432 MB/s, 950 MB/s, and 964 MB/s respectively.

Figure 4-d shows that the block size is also behind the issue of random performance. Increasing the block size to 64 KB leads to eliminate the performance gap between the sequential and random access even in case of using only one I/O process.

The results recommend having larger blocks size in order to leverage the potential of the storage device. However, it indicates that the SSD behaves abnormally using smaller block size. We expect that the internal usage of buffers and internal page size of SSD is behind that behavior. To emphasize, let us suppose that the internal page size of the used SSD is 64 KB and that a file is allocated on multiple NAND flashes. Randomly accessing that file using a block size of 4 KB means that each read will return 64 KB into the internal buffer, but only the target 4 KB will be sent back to the host. Having millions of I/O requests means that the NAND flashes are always accessed to retrieve the data. In contrast, this always robust the performance of sequential workloads even in case of a single I/O process as the I/O requests often have chances to retrieve data from the buffer instead of reaching the NAND flashes.

In the next subsection, we tune internal parameters of the used SSD in order to understand the performance while having one I/O process and small I/O requests.

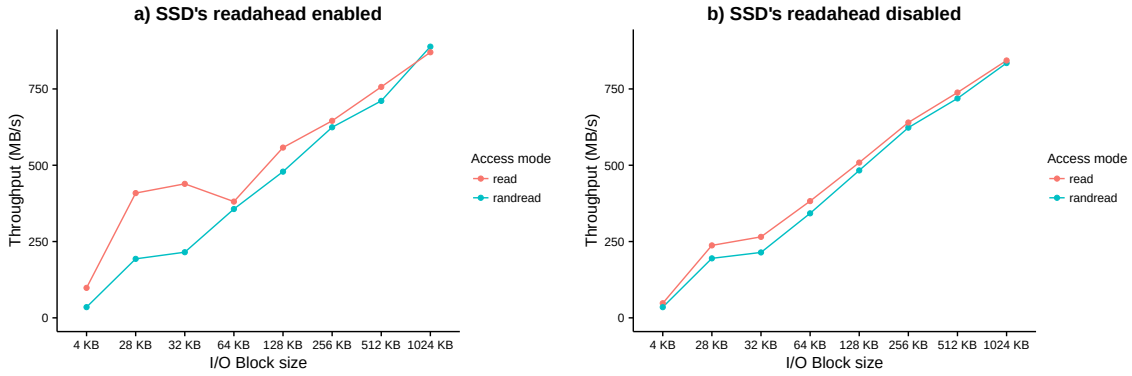


Figure 5: The effects of SSD's internal *readahead* on the experiments

4.4 Tuning the SSD parameters to understand the performance

Despite having hints about how to eliminate the performance gap between the sequential and random workloads, we try to tune some internal parameters of the used SSD to understand why that SSD is sensible to random workload in case of having small block sizes and one individual I/O process. We expect that buffering behaviors are behind the distinction of performance between sequential and random workloads.

The `sdparm`[1] tool is used to alter the state of some internal parameters of the SSD. The study includes several parameters among them (*RCD: read cache disable, WCE: write cache enable, FSW: force sequential write*, & *DRA: disable read ahead*). Our experiments show that all parameters except the *DRA* do not affect our experiments. Figure 5 shows the results of enabling and disabling the SSD's internal *readahead* over several block sizes. Figure 5-a shows that the pre-read of consequent segments of data increases the performance of sequential workloads, especially for the small I/O block sizes as most of the requests will end up reading from the buffer. In contrast, submitting random IOs does not benefit from such a feature since the SSD could not predict what data will be read next. Disabling SSD's *readahead* prevents pre-fetching data for sequential reads, resulting by having comparable performance between sequential and random reads.

4.5 Discussion

Although the experiments of the previous section confirm that the internal *readahead* is behind the performance gap between sequential and random reading workloads, this finding only concerns the studied SSD model (*PX055MB040*). Hence, we need to confirm that via experimenting on other SSDs, in order to neutralize the hardware effects on results.

[1] A tool that can be used to change several control parameters for most SCSI devices

Table 4: A serie of SSD models for investigating the impacts of internal *readahead* on random workload performance

SSD model	SSD Type	Size	Optimized for	manufacturer	Year
MZ7KM240HMHQ0D3	MLC	240 GB	Reliability & heavy demands	Samsung	2017
SSDSC2BB300G4T	MLC	300 GB	Read intensive	Intel	2013
SSDSC2KG480G7R	MLC	480 GB	Read intensive	Intel	2017

We perform further experiments on three different models of SSDs which show different performance for sequential and random workloads. The characteristics of these SSDs are described in table 4.

Although the SSDs showed in table 4 report a performance gap between sequential and random workloads, we are not able to confirm that the internal *readahead* is behind that performance gap as we do with the *PX055MB040* SSD. Unfortunately, manipulating internal parameters of SSDs like the *readahead* is not always possible. It depends on the ability of the target SSD to integrate changes. On the one hand, disabling the internal *readahead* using `sdparm` for Samsung *MZ7KM240HMHQ0D3* is not feasible since the *readahead* parameter on that SSD does not have a changeable state, i.e., we can only read its value. On the other hand, in the case of both *Intel* SSDs (see table 4), the `sdparm` indicates that the parameter is changeable, but the desired value cannot be saved into the SSD. It seems like there is an interface mismatch where the `sdparm` considers our parameter as changeable but unfortunately when we try to alter its value, the SSD refuses to integrate the new value. Given that, finding the main reason for the issue of I/O pattern sensibility on such SSDs is not feasible without the ability to tune their internal parameters.

5 Conclusions

SSDs are capable of serving a large number of I/O requests simultaneously thanks to having their performant storage units arranged in parallel. However, many systems and applications still use only one I/O process to frequently accessing small chunks of data. In this report, we show that such a use case reports different performance between sequential and random workloads. On one side, our experiments confirm that the internal *readahead* of certain SSDs is behind that issue as it does not improve the performance of random workloads as it does with sequential ones. Disabling the *readahead* feature leads to having a similar, but also a moderate performance as there is no more anticipation in data access for sequential workloads. On the other side, eliminating the performance gap without performance degradation can be done via either increasing the I/O block sizes or increasing the number of concurrent I/O processes. The higher the number of concurrent I/O processes, the higher

the obtained performance of SSDs no matter if workloads are accessing data sequentially or randomly.

Future work should consider reproducing that study on *open-channel* SSDs such as *LightNVM*-compatible SSDs. Their accessible and programmable controllers permit to study the internal parameters in details, eliminating the existing restriction of running experiments on black box SSDs.

References

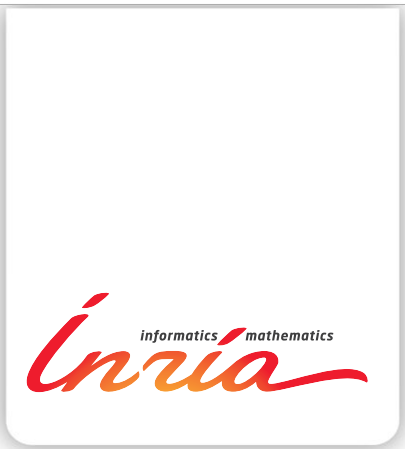
[1] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. pages 57–70, 01 2008.

[2] Matias Bjørling, Javier González, and Philippe Bonnet. Lightnvm: The linux open-channel ssd subsystem. In *FAST*, pages 359–374, 2017.

[3] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Noredine Melab, et al. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *The International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[4] Jaehong Kim, Sangwon Seo, Dawoon Jung, Jin-Soo Kim, and Jaehyuk Huh. Parameter-aware i/o management for solid state disks (ssds). *IEEE Transactions on Computers*, 61(5):636–649, 2012.

[5] Abdulqawi Saif, Lucas Nussbaum, and Ye-Qiong Song. Ioscope: A flexible i/o tracer for workloads' i/o pattern characterization. In *International Conference on High Performance Computing*, pages 103–116. Springer, 2018.



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399