Parallel Storage Devices Profiling with SeRRa

Vinicius Rodrigues Machado[1], Francieli Zanon Boito[1,2], Rodrigo Virote Kassick[1,2], Jean Luca Bez[1], Philippe O. A. Navaux[1], Yves Denneulin[2]

[1]Institute of Informatics – Federal University of Rio Grande do Sul
Porto Alegre, Brazil
{vrmachado, fzboito, jlbez, rvkassick, navaux}@inf.ufrgs.br
[2]LIG Laboratory – INRIA – University of Grenoble Alpes
Grenoble, France
{yves.denneulin}@imag.fr

Abstract. *This work presents the parallel storage device profiling tool SeRRa. Our tool obtains the sequential to random throughput ratio for reads and writes of different sizes on storage devices. In order to provide this information efficiently, SeRRa employs benchmarks to obtain the values for only a subset of the parameter space and estimates the remaining values through linear models. The MPI parallelization of SeRRa presented in this paper allows for faster profiling. Our results show that our parallel SeRRa provides profiles up to* 8.7 *times faster than the sequential implementation, up to* 895 *times faster than the originally required time (without SeRRa).*

1. Introduction

Hard Disk Drives (HDDs) have been the main non-volatile storage devices on personal computers and supercomputers for High Performance Computing (HPC). For this reason, most systems were developed or adapted in order to maximize performance when accessing these devices. For instance, disk schedulers often employ elevator algorithms to reduce head movements [Zhang and Bhargava 2008], improving data access performance. Several optimizations target on generating sequential accesses instead of random accesses, since HDDs benefit from this access pattern.

Solid State Drives (SSDs) are a recent alternative to hard disks. These devices use flash memory to store data. Because of that, they are more resistant to falls and vibrations. Moreover, their advantages over HDDs include size, noise generation, heat dissipation and energy consumption [Chen et al. 2009]. Nevertheless, despite the growing adoption of SSDs, their larger cost per byte still hampers their use on large-scale systems for HPC. Therefore, several parallel file system deployments on clusters still store data on hard disks. Furthermore, hybrid solutions where both devices are used have been gaining popularity [Xu et al. 2014].

Another popular solution for storage on HPC systems is the use of RAID arrays that combine multiple hard disks onto a virtual unit for performance and reliability purposes. Data is striped among the disks and can be retrieved in parallel, which improves performance. Nonetheless, since both SSDs and RAID solutions are inherently different from HDDs, they should not be treated simply as "faster disks". Several assumptions about performance from HDDs do not hold when using SSDs and RAID arrays, and different requirements arise [Rajimwale et al. 2009].

Regarding spatial locality, HDDs are known for having better performance when accesses are done sequentially. Although RAID arrays are composed by hard disks and their performance is usually better with sequential accesses, their organization complicates this behavior. The alignment of accesses to the stripe size has a great impact and must be considered. On the other hand, works that aim at characterizing SSDs' performance behavior achieve different conclusions. Despite most SSDs presenting better performance for sequential writes than for random writes, read operations' performance follows no general rule. On some SSDs, there is no difference between sequential and random accesses, but on others this difference achieves orders of magnitude [Chen et al. 2009]. The sequential to random throughput ratio on some SSDs surpasses what is observed on some HDDs [Rajimwale et al. 2009].

Therefore, we cannot simply classify optimizations - such as I/O schedulers [Boito et al. 2013] - by saying they are only suitable for HDDs or SSDs. Approaches that aim at generating contiguous accesses (originally designed for HDDs) can greatly improve performance when used on SSDs that are also sensitive to access sequentiality. Furthermore, on any device, the performance improvement caused by the use of a specific optimization may not compensate its overhead. Hence, these optimizations could be classified according to the *sequential to random throughput ratio* that devices must present in order to benefit from them.

However, obtaining this metric to a storage device can be a time-consuming task. In order to provide accurate results as fast as possible, SeRRa was developed in our previous work [Boito et al. 2015]. This tool reports, for a storage device, the sequential to random throughput ratio for read and write operations with different request sizes. Benchmarks are executed only over a subset of all profiled request sizes, and the remaining values are estimated through linear regressions. It has been shown that this approach is capable of obtaining a profile in a fraction of the originally required time with small errors.

Although SeRRa offers a faster option for storage devices profiling, obtaining these profiles even faster would facilitate the tool's use for situations such as making decisions about optimizations in execution time. Additionally, it is possible to increase SeRRa's accuracy by allowing for more repetitions of its benchmarks. Thus, accelerating its execution would make it possible to achieve better results from the same profiling time.

In this scenario, this paper proposes a parallel implementation of SeRRa. We evaluate our approach over clusters with HDDs, SSDs, and RAID arrays, and show performance improvements of up to 8.7 times over the sequential version of the tool.

This paper is organized as follows: the next section discusses related work. Section 3 presents the design of SeRRa. Section 4 details the parallel implementation. Section 5 discusses results obtained with the new SeRRa. Finally, Section 6 brings final remarks.

2. Related Work

With the growing adoption of solid state drives, several works focused at characterizing these devices by evaluating their performance over several access patterns [Chen et al. 2009, Rajimwale et al. 2009]. These works point at SSDs' project options,

their impact on performance, and illustrate common phenomena as *write amplification* and *stripe alignment*. Differently, our work aims at measuring storage performance in a generic way, and not only on modeling or explaining SSDs' performance.

El Maghraoui et al. [El Maghraoui et al. 2010] propose a detailed model of SSDs' performance. Nonetheless, the model needs low-level information that must be profiled through micro-benchmarks. Moreover, the proposed model is a low-level model, focusing on the device only and not including higher levels of the I/O subsystem. A similar approach is used by Desnoyers [Desnoyers 2012]. Such models are suitable for evaluating project options for SSDs, for instance, contrary to what our tool does. We aim at providing a high level profiling of the storage system in order to make decisions about optimizations.

For similar reasons, device simulators like *DiskSim* [Bucy et al. 2008] and others [Agrawal et al. 2008, Kim et al. 2009, Yoo et al. 2013] have no use on our context. They allow the evaluation of devices parameters, but not the profiling of existing systems.

Although several benchmarking tools report access time and throughput on the access to files over different access patterns, we could not find any tool that reports the sequential to random throughput ratio. Other tools also do not estimate results for a large set of parameters from a few measurements, as SeRRa does through linear regressions. These reasons motivated the development of the tool, described in the next section.

3. SeRRa: A Storage Device Profiling Tool

This section describes SeRRa, a storage device profiling tool. Its development was motivated by the need for a fast way to obtain the sequential to random throughput ratio from devices as HDDs, SSDs and RAID arrays. The main goals of SeRRa's project are:

- Performance: the information must be provided as quickly as possible;
- Accuracy: the provided information must fairly reflect the real behavior of the profiled storage device;
- Generality: the tool must be easy to use and do not require user-provided information about the device.

Keeping both performance and accuracy goals at the same time is a challenging problem because profiling a storage device adequately can take a long time. Table 1 presents the time spent to profile the devices used in this study. It includes time required for one execution of the tests and for the complete profiling (that includes several executions in order to provide statistical guarantees). Details about these devices and the executed tests will be presented in Section 5. From the times reported in Table 1, it is clear that these tests are time consuming, as the fastest profile took over 1.5 days, while the slowest one took almost 14 days.

Table 1. Original time in hours to profile the storage devices used in this study.

	One test execution	Total profiling time
Pastel (HDD)	15.22	329.49
Graphene (HDD)	12.26	120.38
Suno (RAID-0)	4.21	61.32
Edel (SSD)	3.09	40.44

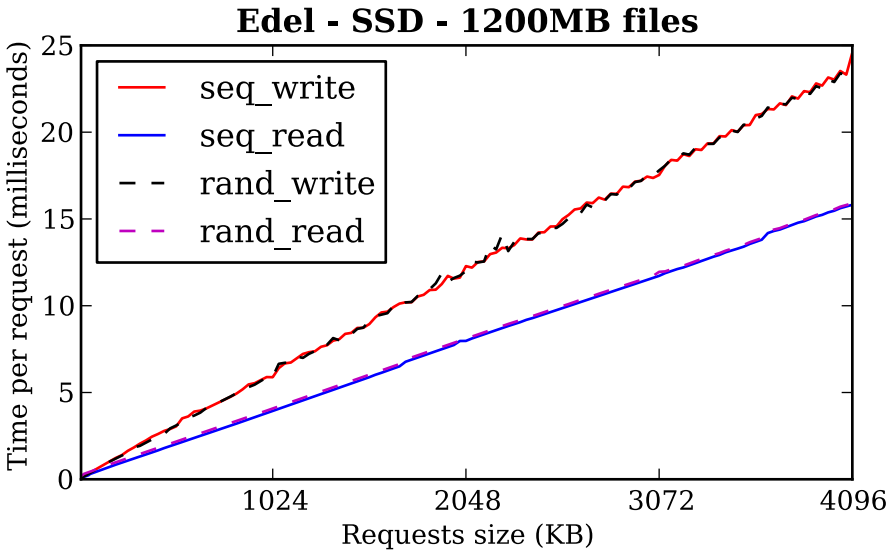


Figure 1. Profiling data from a SSD - Access time per request for several request sizes.

Ideally, such a complete profiling would be needed only once for each storage device and its stored results could be used for a long time. However, even considering this fact, to dedicate the environment for days can be prohibitive. Moreover, since aspects other than the storage device like local file system, disk scheduler and operating system also affect performance, the reported results are dependent on a system configuration. Changing the operating system version, for instance, could require a new profiling.

The slow profiling whose times are presented in Table 1 consists of executing an I/O benchmark several times varying file sizes, request sizes and operation (sequential read, random read, sequential write, and random write). Figure 1 shows an example of access times graph - obtained for 1200MB files with different request sizes on a SSD. Similarly to what can be observed in this example, most of the access times graphs present a linear function appearance. Because of this observation, we have decided to use linear regressions on the design of our profiling tool. Therefore, the following steps compose SeRRa's execution:

1. **Monte Carlo:** request sizes inside a given interval are randomly picked. This interval is provided as a parameter, as well as the gap between every two consecutive sizes. For instance: if asked to approximate the results for the interval [8KB, 40KB] using 8KB gaps, the tool would pick points from the set {8KB, 16KB, 24KB, 32KB, 40KB}. The number of points to be selected for each interval is also defined by the user. Multiple intervals, with different gaps, could be provided. The points at the extremes of the interval (in this example, 8 and 40KB) are always included on this step, since our previous analysis [Boito et al. 2015] has shown that doing so improves approximations.
2. **Benchmark:** the test is executed for the request sizes picked on the previous step. For this task, SeRRa uses the $IOzone$ benchmark [Norcott and Capps 2006], which generates requests to the local file system with configurable parameters such as file size, request sizes, and access pattern (sequential, random, etc). We have decided to use $IOzone$ because it is a widely adopted I/O benchmark, easy

to install and use, and because it fits our needs. Other benchmarks that allow sequential and random accesses, like IOR [Shan et al. 2008], could be used with few modifications on the tool.

3. **Linear Regression:** from the access times measured on the previous step, the complete set of access time results for each given interval is estimated through linear regression. Each test (read or write, sequential or random) for each interval is estimated separately.
4. **Report:** The sequential to random throughput ratios for the read and write tests are reported. The tool provides such values for all request sizes, as well as averages, maximum and minimum values. The estimated access times curves are also provided.

The tool, implemented on *Python*, is open source and available at `http://serratool.bitbucket.org`.

4. Parallel SeRRa

Although SeRRa profiles a machine's storage device through its local file system, one important use of the tool is to provide information to I/O optimization techniques that work to improve performance in the access to a Parallel File System (PFS). In these file systems for cluster architectures, files are distributed among multiple data servers, and these servers use their local storage systems to store data. Therefore, optimization techniques can benefit from knowing how the PFS data servers' storage devices behave regarding access sequentiality. This information can be used, for instance, to decide between different I/O scheduling algorithms, favoring the ones which generate sequential access patterns when this characteristic is important for performance, and avoiding them when the performance improvement caused by sequential access would not compensate the scheduling overhead.

It is usual for HPC architectures to dedicate a set of nodes for the parallel file system deployment. This shared storage infrastructure is often homogeneous, with identical storage devices on all involved machines. These devices are expected to present the same performance behavior, and thus the same sequential to random throughput ratios.

For this reason, we have decided to create a parallel implementation of SeRRa which benefits from this characteristic to provide information faster. A faster profiling facilitates the tool's use for dynamic decision making. Moreover, it would also be possible to dedicate the same amount of time to obtain a profile but achieve more accurate information.

The system administrator could, when deploying the storage infrastructure, use the parallel SeRRa among the nodes with homogeneous storage devices in order to generate information to be later used by optimization techniques. At this moment, the administrator could dedicate as much time as possible to obtain a profile with the best accuracy possible. On the other hand, allowing for faster profiling facilitates the tool's adoption when dedicating the infrastructure for a long time is prohibitive. Even if the storage infrastructure is not completely homogeneous, the homogeneous sets of nodes can be profiled separately (possibly with different parameters).

Moreover, parallel file systems could use SeRRa to profile storage devices that were not profiled beforehand. These devices might, for instance, have been dynamically

added to the storage infrastructure. The PFS could dedicate idle time for this task, or run SeRRa over a subset of its storage nodes. The latter approach would be possible by redirecting requests to other servers which store replicated data. Many file systems keep such redundant servers for fault tolerance purposes.

Another possibility we could envisage is using separated small periods of idle time to storage devices profiling. This would mean obtaining results with low accuracy at first and then incrementally refining by more benchmark repetitions. Every profile refining would require access times curves approximations and sequential to random ratios to be recalculated. Adapting SeRRa's implementation to allow this approach is subject of future work.

SeRRa's parallel implementation uses the master-slave paradigm for communication between processors. We have parallelized the step of benchmarking (step 2 described in the previous section), since it is the most time-consuming one. The master is responsible for all other steps, such as picking measuring points, linear regressions and reporting results. Additionally, the master sends tasks to slaves and receives their results until there are no tasks left. Each task corresponds to a request size to be tested with four access patterns: sequential write, random write, sequential read and random read. This organization is illustrated in Figure 2.

Therefore, it makes no sense to use more processes than available machines, since this approach would lead to benchmarks being executed concurrently in the same node, compromising the results. Furthermore, in this implementation the parallelism is limited by the number of intervals, measuring points per interval and repetitions of the benchmark.

The parallel implementation of SeRRa was developed using MPI4PY[1]. The next section discusses the results obtained with this new version of the tool.

[1]http://packages.python.org/mpi4py

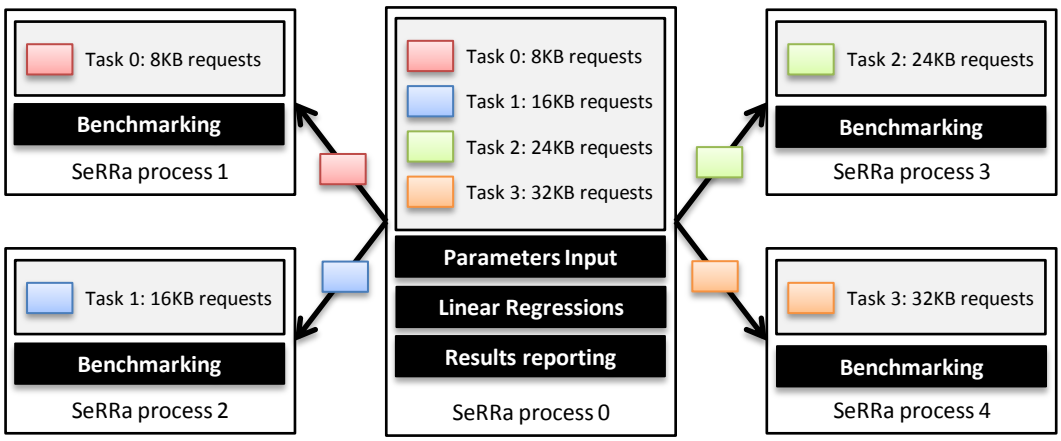


Figure 2. Parallel SeRRa organization.

5. Performance Evaluation

This section presents a performance evaluation of SeRRa's parallel implementation. First, we describe the test environments and methodology on Section 5.1. Then Section 5.2 discusses the obtained results.

5.1. Tests Environments and Methodology

This section describes the four systems used as our test environments, listed in Table 2. These systems were selected by their variety, aiming at representing most available devices. All of them are homogeneous clusters from *Grid'5000* [Bolze et al. 2006].

Table 2. Configuration of the evaluated storage devices

Cluster	RAM	Storage Device			
		Type	Model	Capacity	Bus
Pastel	$8GB$	HDD	Hitachi HDS7225SC	$250GB$	SATA $1.5Gb/s$
Graphene	$16GB$	HDD	Hitachi HDS72103	$320GB$	SATA $3Gb/s$
Suno	$32GB$	RAID-0	2×Seagate ST9300653SS	$2 \times 300GB$	SAS $3Gb/s$
Edel	$24GB$	SSD	Micron C400	$64GB$	SATA $1.5Gb/s$

The tests were executed on the *Linux* operating system, using *IOzone* version *3.397*. All tested devices were accessed through the *Ext3* local file system, and the default *cfq* disk scheduler was kept. Both virtual memory's page size and file system's block size are 4KB. Caching was explicitly disabled through the *O_DIRECT* POSIX flag (*"-I"* parameter for IOzone).

For the tests described in this paper, five different file sizes were used - 40MB, 200MB, 400MB, 800MB and 1200MB. On all tested devices, we observed that the access time curves usually stabilize before 1200MB, not showing significant differences as we increase the file size further.

The request sizes lied within two ranges: from 8KB to 64KB with gaps of 8KB; and from 64KB to 4MB with gaps of 32KB. These values have been chosen aiming at generating interesting information to be used by I/O optimization techniques. Most parallel file systems employ stripe sizes of at least 32KB for their data servers, therefore the requests they receive (and are object to optimizations) are usually multiples of this value. This happens because each large request issued by an application becomes a set of requests, that are multiples of the stripe size, to servers. We defined a smaller gap on the first interval in order to better represent small requests.

We have executed tests using two measuring points per interval, as this configuration was the one with the best results in our previous analysis [Boito et al. 2015]. We use up to four benchmark repetitions. Statistically, executing more repetitions of the benchmarks is expected to provide more realistic results.

5.2. Results with Parallel SeRRa

We compare the parallel implementation's results with SeRRa's sequential implementation and with the profile obtained without the tool. To obtain the latter, we have executed the complete profiling - without estimations, executing the benchmarks for all the specified request sizes - on all tests environments with the parameters discussed in Section 5.1. All tests were repeated until a 90% confidence could be achieved with a *t-student* distribution - with at least six executions. The maximum accepted error was of 10%. The necessary time to obtain this complete profile per machine, up to 14 days, was previously presented in Table 1.

Table 3 presents the total time to perform all experiments described in the previous section by the different implementations. The numbers between parenthesis represent fractions of the originally required time (without SeRRa). The results with parallel SeRRa from the table were obtained using 17 processes (a master and 16 slaves). We show the times for sequential SeRRa with 1 and 4 benchmark repetitions. Repeating the benchmarks multiple times and taking the results' average allows for more accurate results. On the other hand, the time required to obtain a profile increases linearly with the number of repetitions.

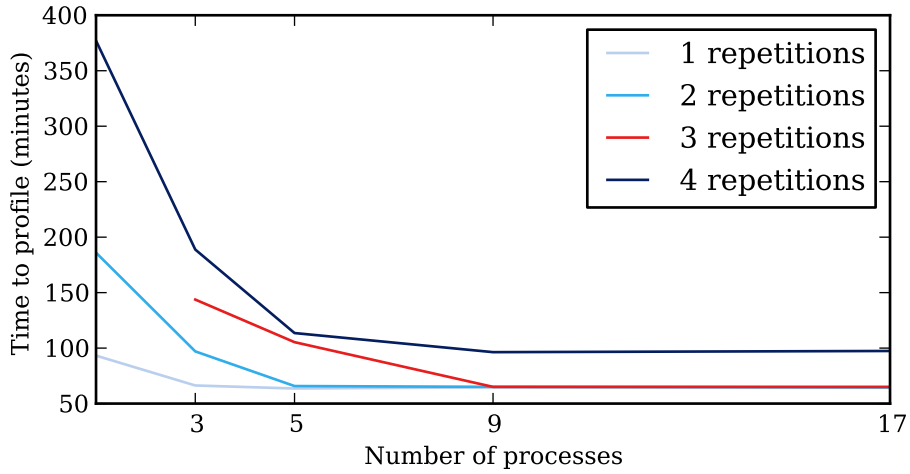
Table 3. Time to profile (in minutes). The fractions between parenthesis compare SeRRa's results with the originally required time (without SeRRa).

	No SeRRa	Sequential SeRRa		Parallel SeRRa
		1 repetition	4 repetitions	4 repetitions
Pastel (HDD)	19769.4	93.21 (1/212)	376.99 (1/52)	71.57 (1/276)
Graphene (HDD)	7222.8	85.69 (1/84)	341.50 (1/21)	70.75 (1/102)
Suno (RAID-0)	3679.2	28.12 (1/130)	109.42 (1/33)	21.56 (1/171)
Edel (SSD)	2426.4	5.92 (1/409)	23.58 (1/102)	2.72 (1/892)
Sum	33097.8	212.94 (1/155)	851.49 (1/39)	166.6 (1/199)

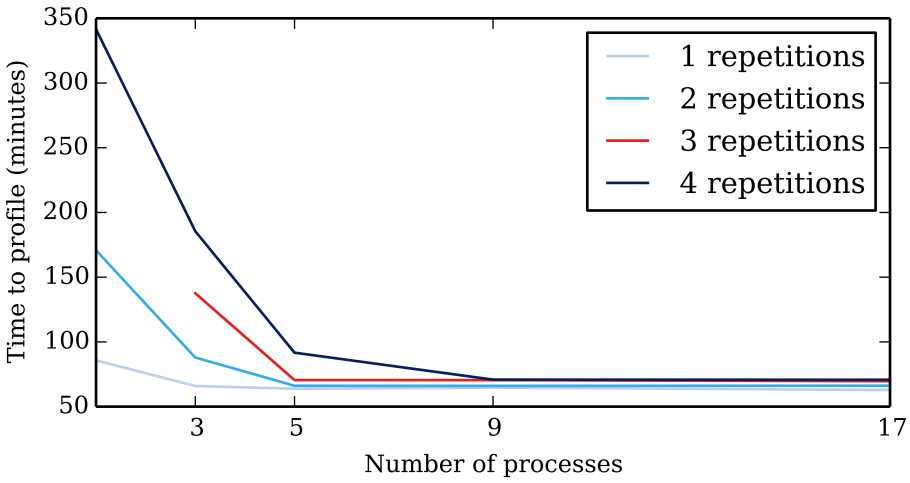
We can see that the parallel version provides a faster alternative. Parallel SeRRa's times for 4 benchmark repetitions were faster than (but close to) sequential SeRRa with 1 benchmark repetition. In other words, it is possible to obtain more accurate results using roughly the same profiling time.

Figures 3 and 4 present the profiling time with SeRRa on the different clusters varying the number of benchmark repetitions. The x axis represents the number of processes. The single-process case (the first point of each line) is the time obtained with the sequential implementation. Onward, the number of processes depicted includes the master plus the slaves – i.e. the number of processes that actually run benchmarks is the number of processes minus 1.

We can see that performance increases (the profiling time decreases) as we increase the number of processes. With few repetitions, performance reaches its best with few processes and does not profit from more nodes to run the profiling. This happens due to the parallelism limit of our experiment: using two intervals with two measuring points per interval, the total number of tasks is $2\ intervals\ \times\ 2\ points\ per\ interval\ \times\ N\ benchmark\ repetitions$. Therefore, using 2 benchmark repetitions we were expected to decrease the profiling time until 8 slaves



(a) Pastel cluster (HDDs)



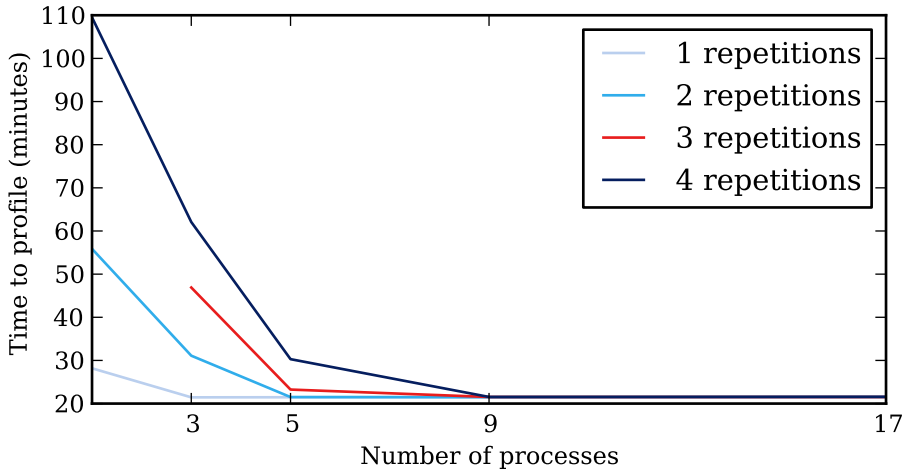
(b) Graphene cluster (HDDs)

Figure 3. Time to profile disks with Parallel SeRRa - clusters with HDDs

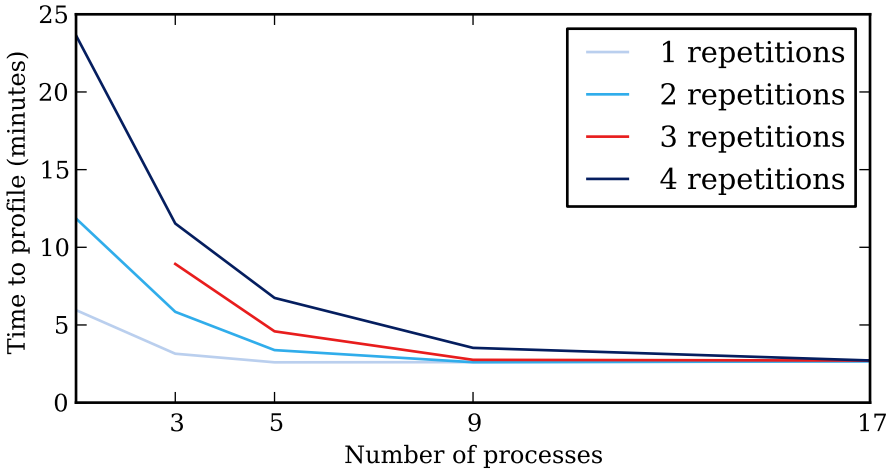
(9 processes), and with 4 repetitions until 17 processes. In order to benefit from more processes, it would be needed to use more benchmark repetitions. However, we can see that in most cases (in Pastel, Graphene, and Suno) there was no difference between the results with 4 benchmark repetitions using 9 or 17 processes. The same happened for results with 2 repetitions considering 5 or 9 processes. This indicates that the maximum speedup can be reached using a number of tasks which is twice the number of slave processes. This configuration leads to a better load balance, as it will be discussed later in this section.

In the Pastel cluster (Figure 3a) with 4 benchmark repetitions, the profiling time is higher than what is observed for other values. One possible reason for this is the higher overhead for the master to manage a higher number of tasks. Since this cluster is the one with the least amount of memory and processing power on the nodes, it was more affected by this overhead than the others.

Another possible explanation for this result is if one of Pastel's nodes is taking longer to run the benchmarks (its storage device is presenting a different performance



(a) Suno cluster (RAID arrays)



(b) Edel cluster (SSDs)

Figure 4. Time to profile disks with Parallel SeRRa - clusters with RAID arrays and SSDs

behavior). This is not expected, since it is a homogeneous cluster, but could happen due to problems with the device. Since the same node is used to test the four access patterns, this difference should not impact the measured sequential to random throughput ratios, only the absolute estimated times. Further analysis is required to quantify this impact and try to avoid it.

Table 4 presents the speedups observed in the results from Figures 3 and 4, comparing the best result from the parallel implementation with the sequential one. As previously stated, the best result for the parallel implementation is obtained with a different number of processes to each number of benchmark repetitions. The speedup increases with the number of benchmark executions because of more parallelism available, as previously discussed.

The speedup is not expected to be linear with the number of processes, since the tasks distributed to the nodes are of different sizes: considering a fixed file size (as SeRRa does), it takes longer to execute the tests with smaller request sizes. In other words, the

Table 4. Speedup provided by SeRRa's parallel implementation (with the best number of processes to each case).

Benchmark repetitions	Pastel (HDD)	Graphene (HDD)	Suno (RAID-0)	Edel (SSD)
1	1.47	1.37	1.31	2.29
2	2.76	2.59	2.6	4.51
3	2.15	1.98	2.18	3.31
4	5.34	4.83	5.08	8.71

parallel implementation profiling time will be limited by the slowest test.

We can observe from Table 4 that speedups observed for the Edel cluster are the highest. This happens because of the sequential to random throughput ratios of the tested devices, presented in Table 5. Since Edel has the lowest ratios from the four clusters, the difference between its tasks is smaller.

Table 5. Sequential do random ratio with 1200MB files for 8KB requests - measured vs. estimated with SeRRa (4 repetitions).

		Pastel (HDD)	Graphene (HDD)	Suno (RAID-0)	Edel (SSD)
Write	Measured	21.29	15.12	8.17	0.66
	SeRRa	22.62	15.21	8.42	0.67
Read	Measured	38.91	40.68	25.46	2.37
	SeRRa	39.08	40.65	25.51	2.38

When storage devices present a high sequential to random throughput ratio, a task which has to perform many small, random accesses to access a fixed-size file will take much longer than a task which accesses a file of the same size with larger, random or sequential requests. This will create a situation of task imbalance which can impair speedup due to longer execution times.

The load imbalance explains why the best performance was achieved having a number of processes which is half the number of tasks for Pastel, Graphene, and Suno; and why this did not happen for Edel, where the difference between the tasks is smaller due to lower sequential to random throughput ratios. Having more available tasks than processes allows for better load balancing. Therefore, one possible solution to avoid the imbalance problem would be to break the tests in smaller units.

6. Final Remarks

This paper presented a parallel implementation of a tool for storage devices profiling regarding access sequentiality named SeRRa. It quantifies the difference between accessing files sequentially and randomly for a given device. In order to obtain this information quickly, SeRRa executes the benchmarks on a small subset of the reported values and the remaining ones are estimated through a linear model.

Decreasing SeRRa execution time is important because it facilitates its use by I/O optimizations to dynamically adapt to storage devices' characteristics. For this reason, we

have developed a parallel implementation with MPI following a master-slave paradigm. This approach is adequate for homogeneous storage nodes.

We have evaluated our approach with four different clusters, using HDDs, RAID arrays and SSDs. Our results show performance improvements (decreases in the profiling time) of up to 8.71 times with the parallel implementation of SeRRa over the sequential one, up to 895 times over not using SeRRa. We show that with the parallel implementation it is possible to achieve results with better accuracy dedicating the same profiling time.

7. Acknowledgments

This research has been partially supported by CNPq and CAPES-BRAZIL under the grants 5847/11-7 and Stic-Amsud 6132-13-8. The experiments presented in this paper were carried out on the Grid'5000 experimental test bed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see `https://www.grid5000.fr`). This research was accomplished in the context of the International Joint Laboratory LICIA and of the HPC-GA project.

References

[Agrawal et al. 2008] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J. D., Manasse, M., and Panigrahy, R. (2008). Design tradeoffs for ssd performance. In *USENIX Annual Technical Conference*, ATC'08, pages 57–70, Berkeley, CA, USA. USENIX Association.

[Boito et al. 2013] Boito, F. Z., Kassick, R., Navaux, P. O. A., and Denneulin, Y. (2013). Agios: Application-guided i/o scheduling for parallel file systems. In *International Conference on Parallel and Distributed Systems (ICPADS)*, pages 43–50, Seoul, South Korea. IEEE.

[Boito et al. 2015] Boito, F. Z., Kassick, R. V., Navaux, P. O., and Denneulin, Y. (2015). Towards fast profiling of storage devices regarding access sequentiality. In *Symposium on Applied Computing (SAC)*, Salamanca, Spain. ACM.

[Bolze et al. 2006] Bolze, R., Cappello, F., Caron, E., Dayde, M., Desprez, F., Jeannot, E., Jegou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.-G., and Touche, I. (2006). Grid5000: A large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494.

[Bucy et al. 2008] Bucy, J. S., Schindler, J., Schlosser, S. W., and Ganger, G. R. (2008). *The disksim simulation environment reference manual*. Parallel Data Laboratory, 4 edition. Available at http://www.pdl.cmu.edu/PDL-FTP/DriveChar/CMU-PDL-08-101.pdf. Accessed in January 2015.

[Chen et al. 2009] Chen, F., Koufaty, D. A., and Zhang, X. (2009). Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *11th International Joint Conference on Measurement and Modeling of Computer Systems*, pages 181–192, New York, NY, USA. ACM.

[Desnoyers 2012] Desnoyers, P. (2012). Analytic modeling of ssd write performance. In *Proceedings of the 5th Annual International Systems and Storage Conference*, SYSTOR '12, pages 12–1, New York, NY, USA. ACM.

[El Maghraoui et al. 2010] El Maghraoui, K., Kandiraju, G., Jann, J., and Pattnaik, P. (2010). Modeling and simulating flash based solid-state disks for operating systems. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 15–26, San Jose, California, USA. ACM.

[Kim et al. 2009] Kim, Y., Tauras, B., Gupta, A., and Urgaonkar, B. (2009). Flashsim: A simulator for nand flash-based solid-state drives. In *First International Conference on Advances in System Simulation (SIMUL)*, SIMUL '09, pages 125–131, Washington, DC, USA. IEEE Computer Society.

[Norcott and Capps 2006] Norcott, W. D. and Capps, D. (2006). *Iozone filesystem benchmark*. Available at www.iozone.org. Accessed in March 2014.

[Rajimwale et al. 2009] Rajimwale, A., Prabhakaran, V., and Davis, J. D. (2009). Block management in solid-state devices. In *Proceedings of the USENIX Annual Technical Conference*, pages 279–284, San Jose, CA, USA. USENIX Association.

[Shan et al. 2008] Shan, H., Antypas, K., and Shalf, J. (2008). Characterizing and predicting the i/o performance of hpc applications using a parameterized synthetic benchmark. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, SC '08, pages 42–53, Piscataway, NJ, USA. IEEE Press.

[Xu et al. 2014] Xu, W., Lu, Y., Li, Q., Zhou, E., Song, Z., Dong, Y., Zhang, W., Wei, D., Zhang, X., Chen, H., et al. (2014). Hybrid hierarchy storage system in milkyway-2 supercomputer. *Frontiers of Computer Science*, 8(3):367–377.

[Yoo et al. 2013] Yoo, J., Won, Y., Hwang, J., Kang, S., Choi, J., Yoon, S., and Cha, J. (2013). Vssim: Virtual machine based ssd simulator. In *IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–14, Long Beach, CA, USA. IEEE.

[Zhang and Bhargava 2008] Zhang, Y. and Bhargava, B. (2008). Self-learning disk scheduling. *IEEE Transactions on Knowledge and Data Engineering*, 21(1):50–65.