SAND2021-12186C

SCTuner: An Auto-tuner Addressing Dynamic I/O Needs on Supercomputer I/O Sub-systems

Houjun Tang*[‡], Bing Xie*[†], Suren Byna[‡], Philip Carns[§], Quincey Koziol[‡]
Sudarsun Kannan[¶] , Jay Lofstead[‖] , Sarp Oral[†]
[†] Oak Ridge National Laboratory, [‡] Lawrence Berkeley National Laboratory, [§] Argonne National Laboratory
[¶] Rutgers University, [‖] Sandia National Laboratories

Abstract—**In HPC, scientific applications often manage a massive amount of data using I/O libraries. These libraries provide convenient data model abstractions, help ensure data portability, and most importantly empower end users to improve I/O performance by tuning configurations across multiple layers of the HPC I/O stack. This work proposes SCTuner, an auto-tuner integrated within I/O library itself to dynamically tune both the I/O library and the underlying I/O stack at application runtime. To this end, we introduce a statistical benchmarking method to profile the behaviors of individual supercomputer I/O sub-systems with varied configurations across I/O layers. We use the benchmarking results as the built-in knowledge in SCTuner, implement an I/O pattern extractor and plan to implement an online performance tuner as the SCTuner runtime. We conducted a benchmarking analysis on the Summit supercomputer and its GPFS file system Alpine. The preliminary results show that our method can effectively extract the consistent I/O behaviors of the target system under production load, building the base for I/O auto-tuning at application runtime.**

I. Introduction

On high-performance computing (HPC) platforms, I/O sub-systems are built around scientific applications. These applications may execute on hundreds of thousands of CPU/GPU cores and analyze/generate tens of TBs to several PBs of data periodically. Ranging from climate modeling to drug discovery, the computation of scientific applications is often forced to stall due to the increasing performance gap between the computational speed in supercomputers and the bandwidth capacity of their I/O sub-systems. It is crucial for software to make efficient use of I/O bandwidth and accelerate application executions accordingly.

At HPC facilities, scientists usually manage data via I/O libraries, such as HDF5 [1], ADIOS [2], and PnetCDF [3]. These libraries support a rich variety of data structures and maneuver toward high throughput by tuning the parameters across multiple I/O layers. For instance, HDF5 provides parallel I/O services via MPI-IO to leverage optimized interfaces for parallel file systems (e.g., Lustre [4] and GPFS [5]). Accordingly, HDF5 users can specify configurations for HDF5 internals (e.g., HDF5 metadata management), MPI-IO (e.g., through MPI_Info object) and file systems (e.g., data layout on Lustre) as well. Ideally, end users can obtain high I/O throughput by tuning the multi-layer configurations on their own. In practice, due to a lack of expert knowledge, most users simply rely on the I/O library defaults, which typically select configurations based on heuristics rather than perspectives from applications.

There is a lack of a unified I/O tuning framework at HPC scale that can handle all critical requirements toward performance optimization, such as online tuning (i.e., during application runtime), tune across layers of the complex HPC I/O stack, and dynamically adapt to changing I/O needs and performance variations of the target applications and file systems. As is shown in Table I, prior HPC-scale I/O tuning efforts are mainly dependent on static heuristics and techniques [6], [7] or offline information characterized from benchmarking [8] and modeling [9], [10]. Other proposals [11] collected the interactions between applications and the underlying file systems at application runtime, and proposed an auto-tuner to optimize I/O across layers, but the proposal was limited to optimized memory use of file systems [12]. A recent study [13] auto-tuned the performance of HPC storage systems with neural networks, but their use is still limited to small-scale systems. For I/O tuning in enterprise datacenters, recent studies [14], [15] auto-tuned the parameters in storage systems by sampling and reducing tuning parameter greedily. Furthermore, Bhimani et al. [16] employed reinforcement learning to auto-tune the parameters of SSDs. These efforts invented I/O tuning frameworks to fine-tune the performance of specific storage systems or devices, but lack the capability to address the dynamic I/O needs of applications at runtime.

To overcome the limitations of prior solutions, we design SCientific I/O Tuner (SCTuner), a generic solution to auto-tune parameters across I/O layers transparently and dynamically. At the heart of our solution, SCTuner realizes dynamic tuning within I/O libraries in two steps. First, SCTuner introduces a generic benchmarking method to profile the behaviors of supercomputer I/O sub-systems with varied configurations across I/O layers scaling up to thousands of cores, and builds performance models in SCTuner based on the profiling knowledge. Second, as part of SCTuner's runtime, we extracts I/O patterns (e.g., number of nodes and cores in use, data size per core, etc) and their performance, adapt the models to the observed performance variations, and determines the corresponding configurations for the entire I/O stack.

SCTuner serves as the first step toward dynamic I/O auto-tuning for HPC applications without requiring application

*Equal contribution.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

TABLE I: SCTuner vs. existing I/O auto-tuners.

I/O Auto-tuner	HPC Scale	Online Tuning	Cross-layer. Tuning	Dynamic Optimization
SCTuner	✓	✓	✓	✓
[6], [7]	✓		✓	
[8], [11], [9], H5Tuner [10]	✓	✓	✓	
[12], CAPES [13]		✓		✓
[14], Carver [15],PatIO [16]				✓

changes. We implement SCTuner within HDF5 and apply our benchmarking method on Summit and its GPFS file system Alpine [17]. We present the results and analysis of our benchmarking approach. We observed that the performance of reads in the target environment is highly variable, suggesting the necessities on a dynamic I/O tuner such as SCTuner. For writes, some configurations obtain consistently high performance, suggesting an effective I/O tuner can bring significant performance improvement to applications.

In summary, we implemented an I/O pattern extractor in HDF5 and plan to realize an online performance tuner in HDF5 asynchronous I/O VOL connector [18] (discussed in III-C). While our initial SCTuner prototype is implemented and integrated with HDF5 for benchmarking and auto-tuning, we believe that SCTuner can be easily applied and integrated into other HPC I/O libraries and file systems.

II. HPC I/O Stack

A. *Scientific I/O in Production Codes*

HPC platforms support a wide range of applications, including traditional numerical simulations and Deep Neural Networks (DNNs) applications [19], [20], [21]. Scientists submit their computational work as a *batch job* to utilize the compute power of a supercomputer. These jobs often execute iterative computations on CPUs/GPUs and process/produce data periodically for resilience (checkpointing) and future data analysis about the physics evolution models.

In general, applications perform I/O for different purposes and with various I/O patterns. For a given application, end-users may execute it in different jobs each with different computation and I/O scales. For example, numerical simulations that solve a fixed-space problem, such as (XGC [22]) execute the numerical *solver* iteratively, and process/produce equal-sized synchronous bursts of I/O. It has four I/O patterns with three for intermediate results (1KB — 10MB of data per core) and one for checkpointing (100MB — 1GB of data per core) [23], [24]. Moreover, machine learning (ML) and deep learning (DL) applications are iterative and exhibit multiple I/O patterns for staging, shuffling, and checkpointing, respectively. For example, a typical DL training code runs on a group of GPU accelerators that read equal-sized data, shuffle and read them periodically, and produce writes for checkpointing and/or post-processing based on a pre-configured frequency. Across DNN models and I/O scales, the typical I/O burst size per GPU could vary from several KBs to several GBs.

Observation ①. There exist different applications and I/O patterns on supercomputers, exhibiting different I/O performance needs at application runtime.

B. *Supercomputer I/O Subsystems*

Summit, the 2nd fastest supercomputer in the world, is housed at the Oak Ridge Leadership Computing Facility (OLCF). It consists of 4,608 compute nodes, each node containing 42 CPUs and 6 GPUs. Summit is connected to Alpine, the center-wide GPFS file system, comprised of 154 Network Shared Disk (NSD) servers. Users have no control over data layout on GPFS deployments, where the block size (GPFS block size) per NSD is configured at file system creation time. For Alpine, the block size is 16MB.

Beyond these systems, other parallel file systems (PFS) such as Mira-FS [25] uses a smaller block size of 8MB, whereas other PFSes (e.g., Lustre) allow users to configure the number of storage targets (*stripe count*) and the data size per target (*stripe size*). In general, as concluded by prior studies [26], [27], [23], [28], [29], [24], all PFS show high performance variability and the variability could change over time [30].

At many HPC facilities, burst buffer [31] is deployed as another storage tier. Using node-local NVMe technology, burst buffers create an independent file system for each job to store its temporary data with high I/O bandwidth during job execution time. It is expected that, burst buffers at different facilities are deployed with different storage technologies and are managed by different file system software.

Observation ②. Different file systems have different hardware, are deployed with different file system software, and in most cases, are configured differently.

C. *I/O Libraries in HPC*

A key attribute of HPC I/O runtimes, in contrast to traditional datacenter I/O file systems and object stores, is the flexibility in allowing applications to customize the data structures that support data and metadata management, I/O work and data sharing policies across thousands of processes. Consequently, flexible runtimes increase the complexity of auto-tuning.

For instance, on HPC platforms, many scientists choose HDF5, a self-describing file format and I/O library [1], to manage their data as it provides flexibility, extendibility, and portability. HDF5 realizes parallel I/O operations via MPI-IO APIs. MPI-IO, in turn, uses POSIX-IO APIs to communicate with the underlying supercomputer I/O storage system. Within HDF5, end users can use either independent MPI-IO calls or collective MPI-IO calls to process read/write operations on their HDF5 files. Moreover, once they choose to use collective I/O, they can further determine the relative MPI-IO configurations, such as the number of read/write aggregators and the buffer size per aggregator. Moreover, HDF5 also provides APIs to let users configure their data layout on the PFSes such like Lustre and BeeGFS [32].

In general, a typical HPC I/O library serves as a higher-level abstraction and provides uniform I/O interfaces for various data structures. It allows end users to set their own configurations on the I/O layers via the library-specific interfaces.

Unfortunately, as a lack of I/O knowledge, end users of these libraries usually choose the default configurations that are determined by simple heuristics and usually fail to address dynamic I/O patterns and performance needs across times.

Observation ③. HPC systems empower end users to configure and customize different layers of I/O stack. However, most users would stick to the default configurations which could result in far-from-ideal I/O performance.

D. Related Work

In HPC, deeper software and hardware I/O stack combined with a rich set of I/O libraries such as HDF5 [1], ADIOS [2], and PnetCDF [3] that rely on MPI-IO and POSIX-IO complicates I/O tuning. Prior work [7] evaluated the setting of several HDF5 tuning parameters such as chunked dataset alignment and metadata flush configurations with the Lustre file system. Other studies explored tuning I/O parameters using genetic algorithms, and performance modeling [8], [9], [10]. On the MPI-IO level, collective I/O [33], [34] allows users to provide a set of *hints* that inform the library regarding the access patterns for runtime optimization. Additionally, Lustre allows users to customize their file layout on storage devices with the striping parameters setting. [35], [27], [36], [28] found that properly setting striping parameters result in multi-fold performance improvements.

While these approaches focus on optimizing specific I/O workloads, they demand I/O expertise from users, scientists, and developers. In contrast, our work, SCTuner, uses a benchmarking method to systematically profile the performance for various access patterns and use the gained knowledge to tune parameters without users' input automatically. Further, unlike prior work for data-centric storage [15], [14], our work is focused on an extreme scale.

III. Autotuning I/O in HPC

Shown in Figure 1, SCTuner is an auto-tuner for scientific I/O on production supercomputers. Motivated by the diverse I/O patterns in the target applications (**Obs.** ①) and the great disparity in the target systems (**Obs.** ②), we conduct I/O benchmarks for target I/O sub-systems. Leveraging the benchmarking results, we integrate SCTuner into HPC I/O libraries, as they generally support parameter configurations across layers in the HPC I/O stack (**Obs.** ③). We use the benchmarking results to build performance models. We extract I/O patterns and performance variations at application runtime, and use this information to adapt the models and determine the best values of the tuning parameters.

In this work, we integrate SCTuner into HDF5 as an example to demonstrate its functionalities. We realized our benchmarking method and devloped the I/O pattern extractor in HDF5 and applied it on Summit's I/O sub-system. The preliminary results show that our proposed benchmarking method can effectively identify the consistently good configurations in a highly variable production environment. Although SCTuner is currently integrated in HDF5 and tested on Summit and Alpine, we believe our benchmarking method and the SCTuner runtime can be easily applied to other HPC I/O libraries and other supercomputer I/O sub-systems.

A. A Statistical I/O Benchmarking Method

Design Principles: We follow three design principles. (1) To address the dissimilarity in the target systems, we design benchmarking experiments to profile the I/O behaviors of individual systems. (2) To address I/O patterns presented in scientific codes, we use IOR [37]) as an I/O pattern generator that covers a wide range of burst sizes and I/O scales. (3) To capture consistent behaviors from noise and randomness, we repeat the experiments and characterize the results by five-number summary [38] and clustering.

1) I/O benchmarking

For an I/O library on a given I/O sub-system, we design a group of controlled experiments, each exercising IOR with an I/O pattern and a set of tuning parameters across I/O layers.

To achieve a low core-hour consumption and low impact on production systems, we perform the experiments on small to medium scales (2—1344 cores) as they can effectively reflect the large-scale behaviors [23], [24]. To attain good coverage on burst sizes, we strategically choose burst-size ranges and randomly produce bursts in each chosen range. We consider bursts in a wide range: 1KB—4GB aggregate data per node. To guarantee balanced burst-size coverage, we break it into 6 ranges (Column 3 in Table II), and generate 20 random burst sizes for each range. Similarly, for each I/O scale, we randomly choose the number of cores per node (Column 2 in Table II). For HDF5 on Summit/Alpine, the configurable layer is MPI-IO. We alternate the configurations between independent I/O and collective I/O, and further vary the values of the collective I/O parameters. Columns 4 and 5 in Table II address the collective I/O configurations in detail. In general, we choose the burst-size range and the values for collective I/O parameters in consideration of production use [23], [29], [24] and Alpine's GPFS block size setting (16MB).

We submit the experiments as regular supercomputer jobs. After a job starts, it reads a job description file, which specifies the IOR executions for specific burst size and a specific number of nodes/cores in use with a multi-level for-loop. Each loop varies the values of an IOR parameter on an I/O layer. We submit each such job many times and execute one at a time to avoid self-interference.

Each job includes a number of IOR executions for one I/O pattern across the same set of varying configurations. Each execution simulates a typical I/O pattern: in an execution, the synchronous processes read/write a single shared file. In particular, a number of benchmark processes each issue a *file_open*, a read or write system call, and a *file_close* in a sequence. The processes are synchronized with MPI barriers before *file_open* and after *file_close*. To avoid read/write-cache effects, each job only executes one read or one write for an I/O pattern across the entire I/O configuration settings. We collected the end-to-end performance from the minimum of *file_open* to the maximum of *file_close* among the bursts.

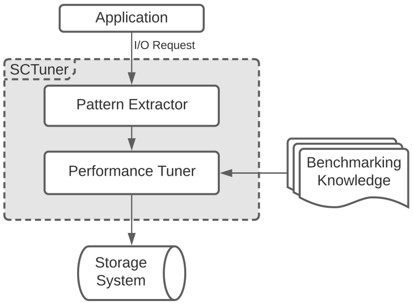


Fig. 1: Key components of SCTuner integrated into HPC I/O libraries

2) *Statistical analysis*

For each IOR execution, we normalize its performance to the best observed from the I/O pattern and accordingly get a relative performance measure in 0—1 for the associated configuration. Clearly, a higher measure suggests a better performance delivered by the configuration. We execute each configuration on each experiment repeatedly, normalize the performance of the repeated IOR executions, and characterize the normalized repetitions using five-number summary (the minimum, lower quartile, median value, upper quartile, maximum). We use *hierarchical clustering* to group the five-number summaries across scales, I/O patterns and configurations.

B. SCTuner Runtime

The SCTuner runtime includes three major components: performance models, an I/O pattern extractor and a performance tuner. Here, each model is built on the benchmarking results of a supercomputer I/O sub-system.

We implement the I/O pattern extractor in HDF5. In particular, when a file is opened for the first time, we extract the information of I/O patterns (e.g., the numbers of compute nodes in use, the number of MPI ranks, the configurations of the underlying file system, etc.). When a parallel I/O call is issued, we extract the per-rank burst size, the starting offset of each burst, as well as the aggregate data size. When an I/O call completes, the extractor also collects the performance related information, and passes the information to the performance tuner. In particular, for an I/O read call, the completion means the data is fetched to the client's memory (in compute node) of the file system; for an I/O write call, the completion indicates the data is committed to the storage system disks.

To address the performance variability in the target environment, the tuner adapts the model to the observed online performance data. Online performance modeling (e.g., online gradient descent [39]) has been used in dynamic resource management in cloud [40]. In this work, we plan to use similar techniques to address I/O performance variations at application runtime. Moreover, when receiving an I/O call, the tuner executes the updated model with the collected I/O pattern and sets parameters across I/O layers accordingly.

C. Dynamic Parameter Setting

Similar to other HPC I/O libraries, HDF5 provides APIs to let end users tune the parameters across the I/O software stack. For example, `H5Pset_mdc_config` allows deferring the HDF5 metadata cache flush until file close time, `H5Pset_coll_metadata_write` enables collective I/O for HDF5 metadata read and write, `H5Pset_fapl_mpio` and `MPI_Info_set` supports the settings on MPI-IO.

TABLE II: Varying parameters in benchmarking experiments

#nodes (m)	Cores per Node (n)	Burst Size (K)	Aggregators (na)	Buffer Size (BS)
2, 4, 8, 16, 32	1—42	1KB—4MB, 4MB—16MB, 16MB—64MB, 64MB—256MB, 256MB—1GB, 1GB—4GB	$\frac{1}{4}n, \frac{1}{2}n, n, 2n, 4n$	1M, 4M, 16M, 64M, 256M

Dynamic parameter setting is challenging as many of the tuning parameters in MPI-IO must be set before the file open. Without the user's input, we have no knowledge of the *future* I/O patterns at the parameter setting time. To address this issue, we build the tuning function on HDF5 asynchronous I/O VOL connector (async VOL) [18], in which I/O operations are queued and executed asynchronously. In other words, with async VOL, we delay the execution on file open until we know the exact I/O operations on the exact I/O patterns. Another challenge comes from the dynamic nature of scientific I/O. As is discussed in §II-A, a typical scientific code performs I/O by following several I/O patterns, and each may benefit from different configurations. To address this dynamism, we may take a close-reopen approach to change the values of tuning parameters if we see the reset obtains performance gain.

IV. Preliminary Results

We present the results collected from Summit (§II-B). We plan to integrate such results from each specific I/O sub-system into SCTuner to build performance models and dynamically tune the parameters across I/O layers at application runtime.

A. Experiment Setup

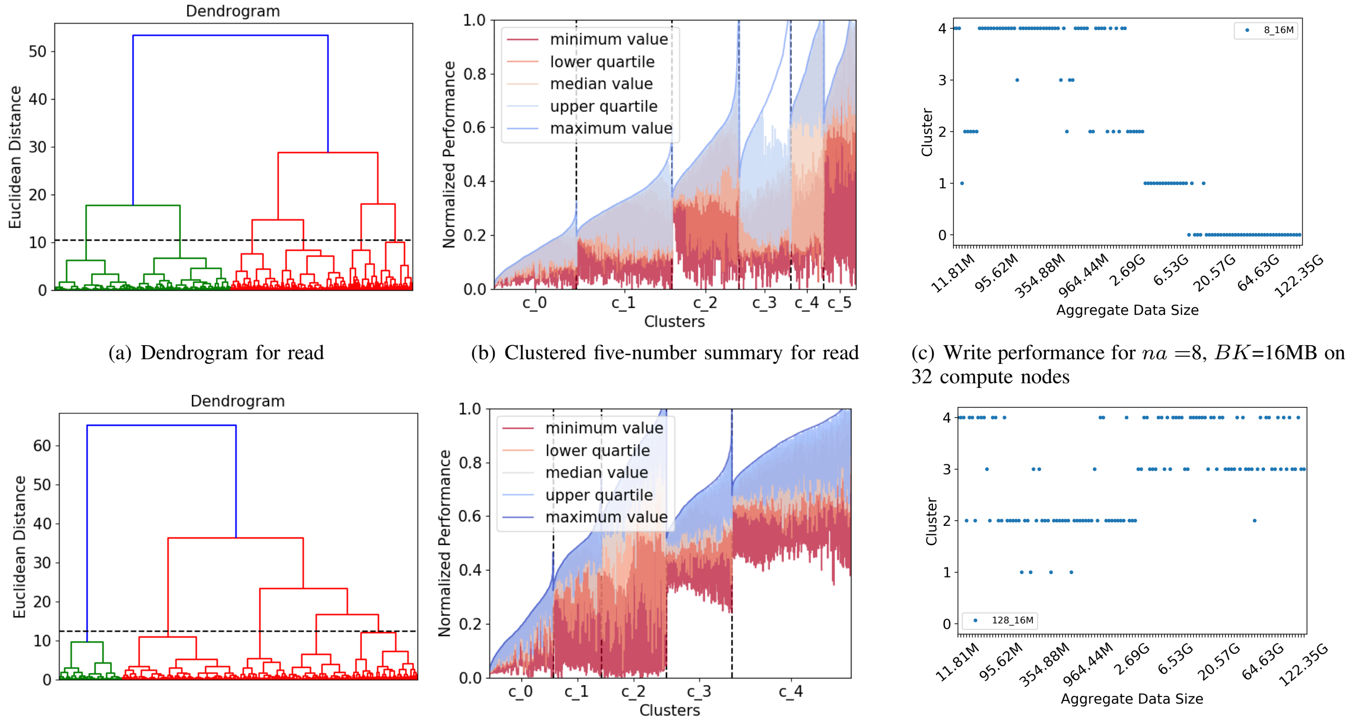
We generate I/O patterns and the values of configurable parameters by following the benchmarking method in (§III-A1). Table II reports the varying parameters and values in detail.

For each I/O scale, we generate 120 I/O patterns across compute cores and burst sizes, and use each pattern to benchmark both read and write operations. For each I/O pattern, we perform IOR with 27 MPI-IO configurations, including independent I/O, the default collective I/O configuration on Summit, and the 25 specified configurations given in Table II. Across scales, we collected overall 12,960 (4x120x27) benchmarking measures. Each measure is a five-number summary characterized from 9-12 repeated IOR executions for a specific MPI-IO configuration on a specific I/O pattern.

B. Clustering Results

Figures 2(a) and 2(d) report the dendrograms of hierarchical clustering with agglomerative (bottom-up clustering) for the five-point summary values. We determine the number of clusters using two commonly used metrics: Ward's linkage and Euclidean distance. In summary, we identified 6 clusters for read and 5 clusters for write.

Figures 2(b) and 2(e) report the read and write performance of individual five-number summaries, each associated with a specific parameter set, pattern, and scale. In the two figures,



(a) Dendrogram for read

(b) Clustered five-number summary for read

(c) Write performance for na =8, BK=16MB on 32 compute nodes

(d) Dendrogram for write

(e) Clustered five-number summary for write

(f) Write performance for na =128, BK=16MB on 32 compute nodes

Fig. 2: Result Summary for the Statistical Benchmarking Method. In Figures 2(c) and 2(f), the x-axis is sorted based on the aggregate data size of an I/O pattern; the y-axis represents the cluster of a pattern.

a line represents the minimum values, or lower quartile, or median value, or upper quartile, or maxi-mum values. We sort the clusters first based on the mean value and then further sort the measures within a cluster based on their maximum values.

Based on Figures 2(b) and 2(e), we summarize the observations into three points: (1) For both read and write, certain MPI-IO configurations consistently deliver poor performance. In particular, in clusters 0 and 1, the maximum values are <57% (for read) and <61% (for write) of the peak bandwidths, respectively. (2) For read, the performance of the target environment is highly variable. For any configuration on any I/O pattern, the read performance varies in a wide range of 7% — 100% of the peak, suggesting the necessities on the dynamic I/O tuner such as SCTuner. (3) For write, some configurations obtain consistently high performance. In clusters 3 and 4, the minimum values are >36% and >42% of the peak performance (observed maximum I/O throughput for all experiments). This also suggests that our method can identify the behaviors of consistently good I/O configurations for an HPC system that is useful for runtime configuration.

C. Configuration Results

Due to the space limitation, we show the effectiveness of our method using the results of two configurations. Figures 2(c) and 2(f) report the clustering results for write on 32 compute nodes, configured with 16MB buffer size per aggregator, and using 8 aggregators and 128 aggregators respectively. We choose these two configurations as they show clearly that different I/O patterns benefit from different configurations.

In particular, for small writes in the range of 11.81MB—2.23GB, when using 8 aggregators, 92.6% of the I/O patterns are categorized into clusters 3 and 4 (high performance clusters), and for larger writes (>2.23GB), the same configuration performs poorly with no measure in clusters 3 or 4. On the contrary, when using 128 aggregators on the large writes (>2.23GB), only one measure falls in cluster 2 (low performance cluster). It is clear that, small writes benefit from aggressive I/O aggregation as it results in larger writes and is more efficient. On the other hand, large writes benefit from independent I/O as they can obtain the efficient use of I/O bandwidth. Undoubtedly, this information is useful to tune the I/O configurations at runtime.

V. Conclusion

This work proposes SCTuner, an auto-tuner built in I/O libaries to tune I/O parameters at runtime. To this end, we introduce a statistical benchmarking method to profile the behaviors of individual supercomputer I/O sub-systems. We implemented the runtime I/O pattern extractor and plan to realize performance models and the online performance tuner in the near future. We conducted benchmarking experiments on Summit supercomputer and its GPFS file system Alpine. The results show that our benchmarking method can effectively extract the consistent I/O behaviors of the target systems.

Acknowledgment

This research is supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-

05CH11231. This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This work used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. Sudarsun Kannan was partially supported by NSF CNS 1850297 award.

References

[1] The HDF Group, "HDF5," https://www.hdfgroup.org/solutions/hdf5/.

[2] ADIOS team at ORNL, "The Adaptable I/O System," https://csmd.ornl.gov/adios.

[3] J. Li, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netcdf: A high-performance scientific i/o interface," in *SC*, 2003, pp. 39–39.

[4] P. Braam, "The Lustre storage architecture," *arXiv preprint arXiv:1903.01955*, 2019.

[5] F. B. Schmuck and R. L. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *FAST*, vol. 2, no. 19, 2002.

[6] S. Byna, M. Chaarawi, Q. Koziol, J. Mainzer, and F. Willmore, "Tuning HDF5 subfiling performance on parallel file systems," *CUG*, 2017.

[7] M. Howison, "Tuning HDF5 for Lustre File Systems," *IASDS*, 2010.

[8] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, Prabhat, R. Aydt, Q. Koziol, and M. Snir, "Taming parallel i/o complexity with autotuning," in *SC*, 2013.

[9] B. Behzad, S. Byna, S. M. Wild, M. Prabhat, and M. Snir, "Improving parallel i/o autotuning with performance modeling," in *HPDC*, 2014, p. 253–256.

[10] B. Behzad, S. Byna, and M. Snir, "Optimizing I/O performance of HPC applications with autotuning," *ACM Transactions on Parallel Computing*, vol. 5, no. 4, pp. 1–27, 2019.

[11] S. J. Kim, S. W. Son, W.-k. Liao, M. Kandemir, R. Thakur, and A. Choudhary, "IOPin: Runtime profiling of parallel I/O in HPC systems," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis PDSW'12*. IEEE, 2012, pp. 18–23.

[12] N. Watkins, Z. Jia, G. Shipman, C. Maltzahn, A. Aiken, and P. McCormick, "Automatic and transparent I/O optimization with storage integrated application runtime support," in *Proceedings of the 10th Parallel Data Storage Workshop (PDSW'15)*, 2015, pp. 49–54.

[13] L. Yan, K. Chang, O. Bel, E. L. Miller, and D. D. Long, "CAPES: Unsupervised storage performance tuning using neural network-based deep reinforcement learning," in *International conference for High Performance Computing, Networking, Storage and Analysis (SC'17)*, 2017.

[14] C. Zhen, V. Tarasov, S. Tiwari, and E. Zadok, "Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems," in *USENIX Annual Technical Conference (ATC'18)*, 2018.

[15] C. Zhen, G. Kuenning, and E. Zadok, "Carver: Finding important parameters for storage system tuning," in *USENIX Conference on File and Storage Technologies (FAST'20)*, 2020.

[16] J. Bhimani, A. Maruf, N. Mi, R. Pandurangan, and V. Balakrishnan, "Auto-tuning parameters for emerging multi-stream flash-based storage drives through new I/O pattern generations," *IEEE Transactions on Computers*, 2020.

[17] B. Xie, H. Tang, S. Byna, J. Hanley, Q. Koziol, T. Li, and S. Oral, "Battle of the defaults: Extracting performance characteristics of HDF5 under production load," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 51–60.

[18] H. Tang, Q. Koziol, S. Byna, J. Mainzer, and T. Li, "Enabling Transparent Asynchronous I/O using Background Threads," in *PDSW'2019*.

[19] J. Kates-Harbeck, A. Svyatkovskiy, and W. Tang, "Predicting disruptive instabilities in controlled fusion plasmas through deep learning," *Nature*, vol. 568, no. 7753, pp. 526–531, 2019.

[20] B. Maldonado Puente, B. Kaul, C. Schuman, S. Young, and P. Mitchell, "Dilute combustion control using spiking neural networks," *SAE Technical Paper Series*, vol. 2021, no. 01, 2021.

[21] R. M. Patton, J. T. Johnston, S. R. Young, C. D. Schuman, T. E. Potok, D. C. Rose, S.-H. Lim, J. Chae, L. Hou, S. Abousamra *et al.*, "Exascale deep learning to accelerate cancer research," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 1488–1496.

[22] G. Bateman, S.-H. Ku, J. Cummings, C.-S. Chang, and A. Kritz, "Xgc documentation," http://w3.physics.lehigh.edu/xgc/, 2016.

[23] B. Xie, Y. Huang, J. Chase, J. Y. Choi, S. Klasky, J. Lofstead, and S. Oral, "Predicting output performance of a petascale supercomputer," in *HPDC'17*, 2017.

[24] B. Xie, Z. Tan, P. Carns, J. Chase, K. Harms, J. Lofstead, S. Oral, S. S. Vazhkudai, and F. Wang, "Interpreting write performance of supercomputer i/o systems with regression models," in *IPDPS'21*, 2021.

[25] K. Kumaran, "Introduction to Mira," in *Code for Q Workshop*, 2016.

[26] J. F. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. A. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing variability in the IO performance of petascale storage systems," in *Conference on High Performance Computing Networking, Storage and Analysis, SC 2010, New Orleans, LA, USA, November 13-19, 2010*. IEEE, 2010, pp. 1–12. [Online]. Available: https://doi.org/10.1109/SC.2010.32

[27] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," in *SC*, 2012.

[28] B. Xie, S. Oral, C. Zimmer, J. Y. Choi, D. Dillow, S. Klasky, J. Lofstead, N. Podhorszki, and J. S. Chase, "Characterizing output bottlenecks of a production supercomputer: Analysis and implications," *ACM Transactions on Storage*, 2020.

[29] B. Xie, Z. Tan, P. Carns, J. Chase, K. Harms, J. Lofstead, S. Oral, S. Vazhkudai, and F. Wang, "Applying machine learning to understand write performance of large-scale parallel filesystems," in *PDSW*, 2019.

[30] G. Lockwood, S. Snyder, T. Wang, S. Byna, P. Carns, and N. Wright, "A year in the life of a parallel file system," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'18)*, 2018.

[31] J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, and J. Woodring, "Jitter-free co-processing on a prototype exascale storage stack," in *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, 2012, pp. 1–5.

[32] B. team, "BeeGFS:the leading parallel file system," https://www.beegfs.io/c/, 2021.

[33] W.-k. Liao and A. Choudhary, "Dynamically adapting file domain partitioning methods for collective I/O based on underlying parallel file system locking protocols," in *SC*, 2008, pp. 1–12.

[34] Y. Chen, X.-H. Sun, R. Thakur, P. C. Roth, and W. D. Gropp, "LACIO: A new collective I/O strategy for parallel I/O systems," in *2011 IEEE International Parallel & Distributed Processing Symposium*, 2011, pp. 794–804.

[35] W. Yu, J. S. Vetter, and H. S. Oral, "Performance characterization and optimization of parallel I/O on the Cray XT," in *IPDPS*, 2008, pp. 1–11.

[36] S. Byna, R. Sisneros, K. Chadalavada, and Q. Koziol, "Tuning parallel I/O on Blue Waters for writing 10 trillion particles," *CUG*, 2015.

[37] "HPC IO Benchmark Repository," https://github.com/hpc/ior.

[38] D. Hoaglin, F. Mosteller, and J. Tukey, *Understanding robust and exploratory data analysis*, 2000, no. Sirsi) i9780471384915.

[39] Y. Ying and M. Pontil, "Online gradient descent learning algorithms," *Foundations of Computational Mathematics*, vol. 8, no. 5, pp. 561–596, 2008.

[40] B. Xie, Q. Cao, M. Kunjir, L. Wan, J. Chase, A. Mandal, and M. Rynge, "WIRE: Resource-efficient scaling with online prediction for DAG-based workflows," in *IEEE Cluster 2021*. IEEE, 2021, pp. 1–10.