ANL/ALCF-17/9

Reproducibility and Variability of I/O Performance on BG/Q

Lessons Learned from a Data Aggregation Algorithm

Leadership Computing Facility

ANL/ALCF-17/9

Reproducibility and Variability of I/O Performance on BG/Q

Lessons Learned from a Data Aggregation Algorithm

prepared by
Francois Tessier, Venkatram Vishwanath
LCF, Argonne National Laboratory

11/28/2017

***Abstract*—Reading and writing data efficiently from different tiers of storage is necessary for most scientific simulations to achieve good performance at scale. Many software solutions have been developed to decrease the I/O bottleneck. One well-known strategy, in the context of collective I/O operations, is the two-phase I/O scheme. This strategy consists of selecting a subset of processes to aggregate contiguous pieces of data before performing reads/writes. In our previous work, we implemented the two-phase I/O scheme with a MPI-based topology-aware algorithm. Our algorithm showed very good performance at scale compared to the standard I/O libraries such as POSIX I/O and MPI I/O. However, the algorithm had several limitations hindering a satisfying reproducibility of our experiments. In this paper, we extend our work by 1) identifying the obstacles we face to reproduce our experiments and 2) discovering solutions that reduce the unpredictability of our results.**

***Keywords*-reproducibility; performance variability; parallel I/O; data aggregation; interference;**

I. Introduction

The problem of reproducibility in scientific research is of paramount importance. It gives the scientists an essential level of intellectual and scientific honesty and the published results engage the legitimacy of the scientific community. Ensuring reproducibility allows one to independently confirm a set of experiments and can offer the community the confidence to fully appreciate the outcomes of a research. Reproducible results are also useful to provide researchers with confidence in their own results as well.

Reproducible results and performance are two areas of interest in scientific computing. In this paper, we address the issues involved in the reproducibility of I/O performance for scientific computing. This is a challenging problem as there are several layers of software and hardware involved in the I/O stack, including the runtimes systems, the interconnect and storage topology, and the filesystem. All these make reproducible and portable I/O a challenging problem.

Reproducibility in I/O performance is critical given the importance of I/O in computational science. Simulations in scientific domains such as climate science, neuroscience, and computational cosmology have increasingly large I/O requirements.

Table I depicts three examples of the data volumes generated by applications. Typically, we estimate the wall time spent in I/O around 10% to 20%. Thus, converging to a stable I/O performance is essential, particularly at scale and regarding the upcoming architectures.

Table I: I/O requirements of diverse large-scale applications

Scientific domain	Simulation	Data size
Cosmology	Q Continuum	**2 PB / simulation**
High-Energy Physics	Higgs Boson	**10 PB / year**
Climate / Weather	Hurricane	**240 TB / simulation**

In this paper, we present our work in optimizing I/O on large-scale supercomputers and share our experience in facing reproducibility challenges we have faced. We elucidate these issues and draw lessons about the necessity of a strong knowledge of the architecture or about the impact of input parameters in results' variability. We also propose some guidelines, and in some cases we augment this with the help of a micro-benchmark to emphasize specific characteristics.

II. Related Works

The growing amout of data generated by large-scale simulations makes parallel I/O a critical topic to address. Research has been conducted on this topic for several years, whether it be at the file system level with solutions designed for HPC such as GPFS [1] or Lustre [2] or at a runtime level with parallel I/O library like MPI I/O included in the MPI-2 [3] standard. In MPI I/O, collective I/O is a significant research area as it allows to improve performance. A number of algorithms have been developed to optimize collective I/O. Venkatesan et al. [4] have proposed an approch based on process placement while Isaila et al. [5] have chosen to combine autotuning with machine learning techniques. From a more general point of view, studies have been made to evaluate various collective I/O algorithms [6]. As for us, we have focused our work [7] on a topology-aware optimized implementation of the major two-phase I/O scheme [8].

Reproducibility in computer science has been studied for years [9]. Several aspects have been addressed such as the free licences of scientific innovation [10], the reproducibility of results in numerical libraries like BLAS [11] or the reproducibility of performance in HPC [12], [13].

This paper is focusing on this last topic and specifically on reproducibility of I/O performance. Several problems have already been studied for I/O, but often without the reproducibility perspective. For instance, Yildiz et al. have proposed to study the causes of I/O interference [14]. Another work has focused on a scheduling model reducing I/O interferences between HPC applications [15] leading to more stable performance. Some work has also been done to re-think the software stack managing I/O in a unified way [16].

In this paper, we propose to study these challenges from a reproducibility point of view through our experience with the topology-aware I/O library we previously developed.

III. Context and Motivation

MPI [3] is commonly used to implement large-scale distributed memory applications on high-performance clusters. MPI I/O is a critical component to perform I/O. The collective I/O mechanism in MPI I/O helps applications effectively read and write data at scale. In collective I/O, all MPI tasks involved in the communicator call the I/O routine, typically in a bulk synchronous manner. This type of operation allows the MPI runtime system to optimize data movement based on knowledge from the application

including parameters such as data size, and the layout in both memory and on storage.

The Two-phase I/O algorithm is a well-known and efficient optimization available in MPI I/O implementations such as ROMIO [17]. It consists in selecting a subset of processes to aggregate contiguous pieces of data (aggregation phase) before writing them to the storage system (I/O phase). Figure 1 depicts this technique using an illustrative example involving four processes, two of them being chosen as aggregators. Thus, the network contention decreases around the storage system while the I/O bandwidth is substantially increased by the write of large chunks of contiguous data. However, this approach as it has been implemented suffers several limitations. First, even if the I/O performance is better compared to an unoptimized operation, it usually remains far from the peak I/O bandwidth achievable. Second, we noticed an inefficient aggregator placement policy even though a smart aggregators mapping may have a real impact on performance. Finally, the common implementations fail to take advantage of the data model, the data layout, and the memory hierarchy.

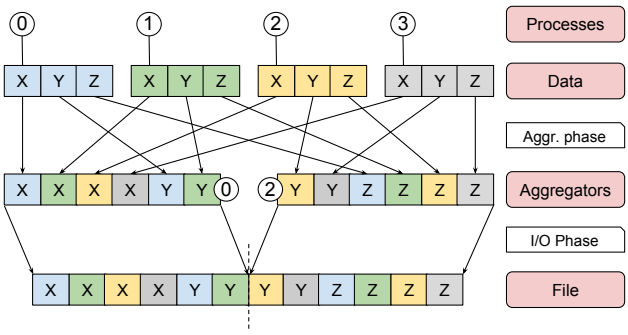

Figure 1: Two-phase I/O mechanism

Our previous work [7] focused on the two-phase I/O scheme and proposed to address several of these limitations. We developed an I/O library on top of MPI I/O implementing a topology-aware two-phase I/O scheme optimized for large-scale supercomputers. This library has been created following two key directions: an efficient implementation of the two-phase I/O algorithm and an improved aggregator placement taking into account system characteristics. We next describe these two points and present results obtained from our experiments on the Mira Blue Gene/Q supercomputer at the Argonne National Laboratory. Finally, we will introduce the challenges we encountered to ensure a satisfying reproducibility and a good performance portability.

A. Implementation of the Two-phase I/O scheme

In comparison to the MPI standard, our approach requires the description of the upcoming I/O operations before performing read or write calls. We extract from this information the data model (multidimensional arrays, meshes, ...) and the data layout (array of structures, structure of arrays). These data patterns allow us to determine better I/O scheduling and to reduce the idle time for all the MPI tasks. Additionally, we improved both the aggregation and I/O phases by overlapping them. With this aim in mind, we allocate two buffers per aggregator and supply them as a pipeline: while aggregating data in the first buffer, an aggregator can flush the second one to the storage system concurrently. This can be done with one-sided MPI communication (Remote Memory Access) to aggregate data in the aggregators' buffers and, thanks to non-blocking MPI I/O functions, to effectively read or write buffers in file.

B. Topology-aware aggregators placement

The second main contribution of this work on data aggregation concerns the aggregators placement policy. The various implementations of the MPI-2 standard propose a couple of aggregators mapping strategies for two-phase I/O. For example, in MPICH [18] a strategy consists in selecting the bridge node (i.e. the node directly linked to the I/O node) as a first aggregator and the other aggregators following a rank order. This strategy takes into account neither the distance between the compute nodes and the storage system nor the amount of data exchanged. Moreover, the process mapping may severely impact the performance by selecting aggregators on neighboring nodes inevitably creating contention. Our strategy involves considering the topology of the architecture and the data access pattern in an objective function in order to determine a near-optimal aggregator placement optimizing data movements. For the rest of this paper, we call "partition" a subset of nodes hosting processes sharing a contiguous piece of data in file. The number of aggregators defines the partition size, each partition electing one aggregator among the processes.

Given, for each partition:

- V_C: The set of compute nodes performing aggregation in the partition;
- $A \in V_C$: An aggregator chosen among compute nodes;
- IO: The storage system (I/O node) of the partition;
- $\omega(u, v)$: The amount of data exchanged between nodes u and v;
- $d(u, v)$: The number of hops between nodes u and v;
- l: The interconnect latency;
- $B_{i \to j}$: The bandwidth from node i to node j.

Figure 2 shows the two costs computed in our objective function for one partition. These two costs model the two phases of the algorithm.

The cost C_1 corresponds to the cost of aggregating data into aggregator buffers. Every process involved in the partition computes this cost in a distributed way as if it was chosen as the aggregator. From a candidate point of view, for each rank producing data the aggregation cost is computed and the maximum is kept as all the others will be bounded by this one. Formally, each process A computes the cost C_1:

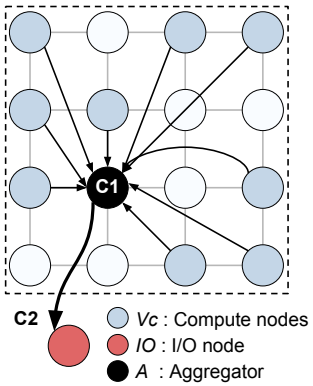

Figure 2: Objective function minimizing the communication costs to and from an aggregator.

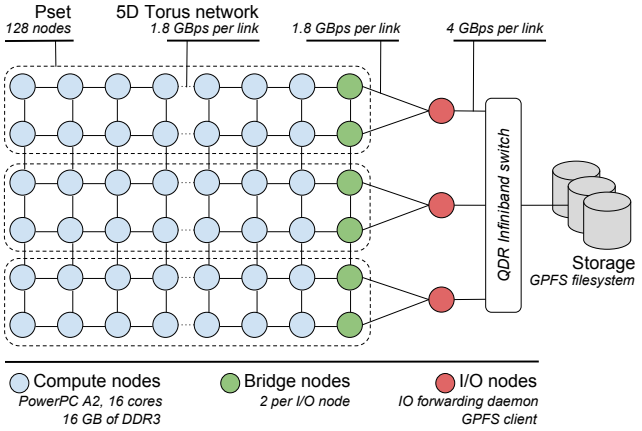

Figure 3: IBM BG/Q Architecture

$$C_1 = \max\left(l \times d(i,A) + \frac{\omega(i,A)}{B_{i\to A}}\right), i \in V_C, i \neq A$$

For the next step, the candidate compute the cost of sending the sum of aggregating data to the storage system (I/O node). For each process A, we define the cost C_2 as:

$$C_2 = l \times d(A,IO) + \frac{\omega(A,IO)}{|V_C| \times B_{A\to IO}}$$

The objective function modelling our topology-aware aggregator placement strategy offers to minimize the sum of the costs C_1 and C_2. Formally, each process in a partition computes this objective function:

$$TopoAware(A) = min\,(C_1 + C_2)$$

A call to `MPI_Allreduce` with the `MPI_MINLOC` enables our algorithm to choose as an aggregator the process with the minimal cost. Hence, for each partition an aggregator is elected.

C. Results

We evaluated our topology-aware I/O library on HACC-IO, the I/O kernel of HACC [19], a large-scale cosmological application simulating the mass evolution of the universe with particle-mesh technique.

The experiments were carried out on Mira, a 10 PetaFLOPS IBM BG/Q supercomputer at Argonne National Laboratory (see Figure 3) with 48K nodes interconnected with a 5D-torus high-speed network providing a theoretical bandwidth of 1.8 GBps per link. Each node hosts 16 hyper-threaded PowerPC A2 cores (1600 MHz) and 16 GB of main memory. Following the BG/Q architecture rules, Mira splits the nodes into *Psets*. A *Pset* is a subset of 128 nodes sharing the same I/O node. Two compute nodes of a *Pset* offer a 1.8 GBps link to the I/O node. These nodes are called the bridge nodes. GPFS manages 27 PB of storage.

In this paper, we will present our results with HACC-IO in terms of write I/O bandwidth on 1024 nodes on Mira with a single shared file and one file per *Pset*. Let's note that on 1024 nodes, the peak I/O bandwidth is estimated to 22.4 GBps (theoretical: 28.8 GBps). For these experiments, we set the number of aggregators to 128, in other words 8 aggregators per *Pset* similar to the default MPICH configuration on BG/Q. We also allocate 16 MB per aggregator's buffer. Our topology-aware strategy is compared to ROMIO, the MPI I/O implementation in use on Mira and to POSIX I/O. Two data layouts are shown, highlighting their impact on performance.

Figure 4 shows the results on 1024 Mira-nodes, with 16 ranks per node and a single shared file as output. By varying the number of particles managed by each rank, we increase the data size on the x-axis. The first remark we can make is that the best performance we can achieve does not exceed 5 GBps, being approximately 20% of the peak performance on the best case. Nonetheless, our approach still outperforms the standard strategies on both data layouts. In particular, with the smallest number of particles tested (5K particles or 190KB per process) and an array of structures data layout, our implementation performs 15 times faster than MPI I/O.

When increasing the number of processes, reading or writing data in a single shared file proves to be extremely limited to achieve good I/O performance. A solution consists in partitioning the data in several files (subfiling). Figure 5 presents our experiments in the same configuration as the previous ones except that one file is created per *Pset*. We can notice that subfiling is an efficient technique to improve I/O performance since up to 90% of the peak I/O bandwidth is achieved by our topology-aware strategy. Again, we outperform the default implementations even on large messages.

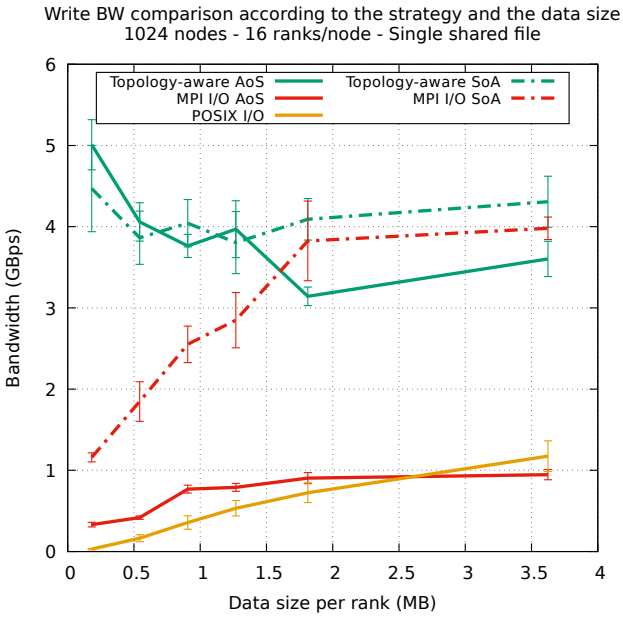

Figure 4: I/O bandwidth achieved on Mira by writing one single shared file from 1024 nodes (16 ranks/node)

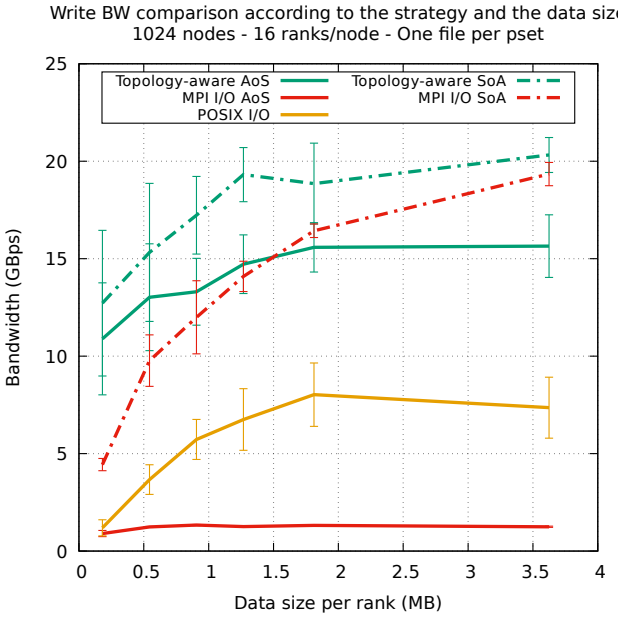

Figure 5: I/O bandwidth achieved on Mira by writing one file per *Pset* from 1024 nodes (16 ranks/node)

D. Reproducibility issues

While implementing this optimized version of the two-phase I/O scheme and during our experiments, we have noticed a couple of reproducibility issues. The first one is related to the supercomputer architecture. A minimal base of knowledge is required to interpret the results and to reduce I/O interference. The second issue concerns the software stack's stability. From one version of the runtime to another, the performance can be heavily impacted. Finally, the choice of input parameters may cause a large variation in results as can be seen in Figure 5. In the next section of the paper, we will give details about these three issues and propose some guidelines to restrict as much as possible their impact on reproducibility and performance portability.

IV. Reproducibility

A. Topology-awareness and I/O interference

When carrying out experiments with our I/O library, we observed a certain variability in the I/O bandwidth measured. This instability was due to I/O interference with other jobs. On Mira, a set of I/O nodes is isolated only as part of a 512-nodes allocation.

To emphasize this behavior, we ran controlled benchmark tests using one *Pset* (128 nodes compute nodes, two bridge nodes and one I/O node). Our tests were run to highlight the impact of I/O interference. In one case, we ran a single I/O intensive HACC-IO job on 64 of the 128 nodes, while leaving the other 64 nodes idle. This case eliminated interference on the bridge and I/O nodes. In the other case, we ran the same I/O intense job on 64 of the 128 nodes, while simultaneously running jobs of varying I/O intensity on the other 64 nodes. Node allocation was distributed such that each 64 node job used 32 nodes per bridge node. This configuration corresponded to the default distribution on BG/Q. Figure 6 depicts a 5D Torus flattened on 2 dimensions and the aforementioned jobs partitioning.

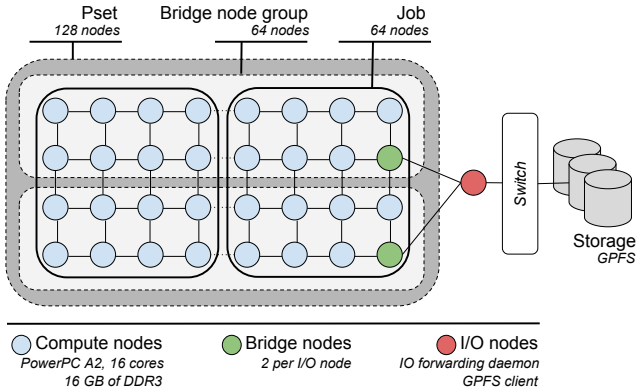

Figure 6: Job partitioning on a *Pset* on Mira to demonstrate the impact of I/O interference on performance.

Table II shows the mean I/O bandwidth achieved with HACC-IO with and without interference. A single I/O intensive HACC-IO job running on 64 nodes sharing two bridge nodes can reach more than 60% of the peak I/O bandwidth. However, the performance is decreased by 13% when a concurrent job is running on the same *Pset*. We can also notice a rise in variability (standard deviation) of 37.5%.

This result demonstrates the need to have a good knowledge of the underlying topology. On BG/Q for instance, we have learnt that the minimal unit to consider for a node allocation is a block of four *Psets* (512 nodes) to reduce as much as possible the impact of I/O interference and ensure a good reproducibility.

Table II: Mean I/O bandwidth achieved with HACC-IO (2 MB per rank) through our I/O library with and without interference. Concurrent jobs have variable I/O intensity (0.2 MB to 4 MB per rank).

	HACC-IO		Other	
	Average	**Std_dev**	**Average**	**Std_dev**
no-interference	2.20 GBps	0.10 GBps	N/A	N/A
interference	1.92 GBps	0.16 GBps	1.15 GBps	0.35 GBps

B. Software Stack

As discussed in Section III-B, our strategy creates partitions based on contiguous chunks of data in file and elects an aggregator per partition. The chunk size is always a multiple of the file system block size[1]. This constraint mainly avoids lock contention in the file system. However, it also implies that a process may read or write data from two partitions given that its data can belong to two different chunks. Because of that, for each file as output, the aggregators are chosen sequentially. This algorithm step is performed with a call to `MPI_Comm_split` to create a sub-communicator including the processes involved in a partition. In the case of subfiling per *Pset*, the sets of aggregators per file (usually 8) are chosen in parallel. On the other hand, in the case of a single shared file, this step becomes expensive as all the aggregators (usually 64 for 1024 nodes) are elected sequentially. In the big picture, the aggregators selection is usually done once during the execution and even a number of seconds is insignificant compared to a large-scale simulation execution time and its I/O phases. However, we still would like to limit the computation of aggregator placement to a reasonable time. This is primarily to deal with upcoming supercomputing architectures. That force us to consider several tiers of storage hierarchy with various performance during the same application lifetime. Thus, we may have to compute a new placement at various stages of the execution in relation to one or the other storage system.

We profiled our implementation and identified the performance of `MPI_Comm_split` as a key bottleneck. On BG/Q, the default MPI implementation is based on MPI-2. We decided to evaluate the performance of an alternate MPI implementation based on the MPI-3 standard on the BG/Q. We noticed significant performance differences. To better understand this, we developed a benchmark reproducing

[1]The block size of a file system is the smallest indivisible piece of memory involved when performing I/O.

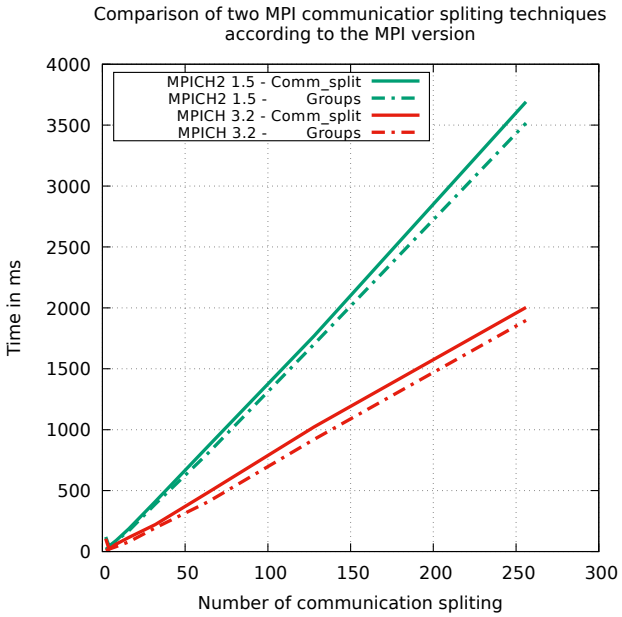

Figure 7: Time of two spliting techniques with MPICH-2 1.5 and MPICH 3.2 applied on a 1024 ranks communicator.

the behavior of our library for splitting communicators. This benchmark applies successive `MPI_Comm_split` on a 16K ranks communicator. We also developed a `MPI_Comm_split` benchmark based on `MPI_Groups` wherein we add processes to a group in order to achieve the same functionality of splitting a communicator. We compared these two techniques using MPICH-2 1.5 and MPICH 3.2 on Mira. The results are presented in Figure 7. These results show a factor of two between the two MPI implementations regardless of the splitting technique. They bring out the effect of the MPI runtime in the performance. In general, the reproducibility of our experiments can be heavily affected by any change in the software stack, and, in this case the MPI implementation.

C. Problem Size

The choice of input parameters when carrying out experiments is critical. It's often a trade-off to find between what one wants to highlight and the scientific relevance of the experiments. On HACC-IO, the main input parameter is the number of particles managed by rank. Each particle is composed of 9 variables bringing the size of a particle to 38 bytes. To evaluate our approach, we ran our tests while varying the number of particles per rank from 5,000 to 100,000 (190 KB to 3.7 MB). This range of particles is quite common in HACC, validating the scientific accuracy of the experiments. However, particularly in the subfiling case (see Figure 5 in Section III-C), we noticed a significant variation of the I/O bandwidth achieved according to the data size (number of particles). Figure 8 depicts the absolute standard

deviation calculated from 10 runs of this experiment. The percentage of the mean execution time represented by this standard deviation is also plotted.

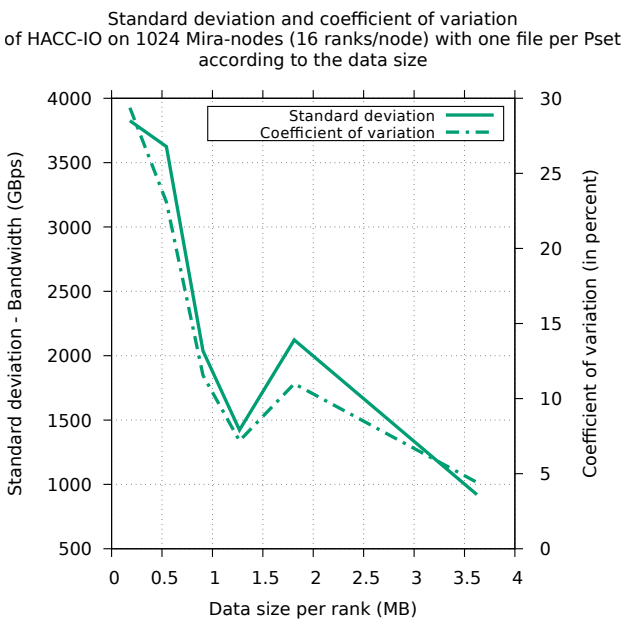

Figure 8: Variability of I/O performance obtained with HACC-IO and our I/O library on Mira.

In the smaller case with 5,000 particles per process, the standard deviation is extremely high, reaching a coefficient of variation[2] of almost 30%. Apart from a peak observed with 50,000 particles (1.8 MB), the coefficient of variation tends to strongly decrease with the data size. In the larger case, this value is close to 5% showing a low dispersion of the experimental results. This analysis points out the effect of the problem size, and more generally of input parameters in performance variability. It leads to try out a large range of parameters to get close to a compromise between reproducibility and scientific precision.

V. Conclusion

In this paper, we presented our experiences with the development of a topology-aware I/O library and discussed some of the challenges we addressed to ensure reproducible I/O performance. Particularly, we have demonstrated the necessity to have a good understanding of the system topology to reduce I/O interference. We discussed the impact of a change in the software stack on the time taken by a placement algorithm. Finally, we have shown that the variability of the I/O bandwidth on a large-scale system can be correlated with the choice of the problem size and, in a more general way the input parameters. With the reproducibility results in mind, we provided useful lessons while implementing and evaluating our model: 1) the need to carefully select allocation size isolating I/O nodes, 2) the necessity to stabilise the I/O software stack and 3) the significance of input data size for reducing variability.

A next step for us is to ensure the performance portability on other platforms. This promises to be challenging considering tremendous diversity of the different layers of the I/O stack. An imminent availability of our work under a free licence is also a requirement to ensure a new level of reproducibility.

[2]The coefficient of variation is the ratio of the standard deviation and the mean expressing the data dispersion.

Acknowledgment

This research has been funded in part from and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract no. DE-AC02-06CH11357.

References

[1] F. Schmuck and R. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, ser. FAST '02. Berkeley, CA, USA: USENIX Association, 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083323.1083349

[2] "Lustre filesystem website," http://lustre.org/.

[3] M. P. I. Forum, "Mpi-2: Extensions to the message-passing interface," July 1997, http://www.mpi-forum.org/docs/docs.html.

[4] V. Venkatesan, R. Anand, J. Subhlok, and E. Gabriel, "Optimized process placement for collective i/o operations," in *Proceedings of the 20th European MPI Users' Group Meeting*, ser. EuroMPI '13. New York, NY, USA: ACM, 2013, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/2488551.2488567

[5] F. Isaila, P. Balaprakash, S. M. Wild, D. Kimpe, R. Latham, R. Ross, and P. D. Hovland, "Collective i/o tuning using analytical and machine-learning models," in *IEEE Cluster 2015*, IEEE. Chicago, IL: IEEE, 09/2015 2015.

[6] M. Chaarawi, S. Chandok, and E. Gabriel, *Performance Evaluation of Collective Write Algorithms in MPI I/O*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 185–194.

[7] F. Tessier, P. Malakar, V. Vishwanath, E. Jeannot, and F. Isaila, "Topology-aware data aggregation for intensive i/o on large-scale supercomputers," in *Proceedings of the First Workshop on Optimization of Communication in HPC*, ser. COM-HPC '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 73–81. [Online]. Available: https://doi.org/10.1109/COM-HPC.2016.13

[8] J. M. del Rosario, R. Bordawekar, and A. Choudhary, "Improved parallel i/o via a two-phase run-time access strategy," *SIGARCH Comput. Archit. News*, vol. 21, no. 5, pp. 31–38, Dec. 1993. [Online]. Available: http://doi.acm.org/10.1145/165660.165667

[9] V. Stodden, "The legal framework for reproducible scientific research: Licensing and copyright," *Computing in Science Engineering*, vol. 11, no. 1, pp. 35–40, Jan 2009.

[10] ——, "Enabling reproducible research: Licensing scientific innovation," 2015.

[11] R. Iakymchuk, S. Collange, D. Defour, and S. Graillat, "ExBLAS: Reproducible and Accurate BLAS Library," in *NRE: Numerical Reproducibility at Exascale*, Austin, TX, United States, Nov. 2015, numerical Reproducibility at Exascale (NRE2015) workshop held as part of the Supercomputing Conference (SC15). [Online]. Available: https://hal.archives-ouvertes.fr/hal-01202396

[12] D. J. et al., "Standing together for reproducibility in large-scale computing: Report on reproducibility@xsede," *CoRR*, vol. abs/1412.5557, 2014. [Online]. Available: http://arxiv.org/abs/1412.5557

[13] T. Hoefler and R. Belli, "Scientific Benchmarking of Parallel Computing Systems." ACM, Nov. 2015, pp. 73:1–73:12, proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC15).

[14] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu, "On the root causes of cross-application i/o interference in hpc storage systems," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 750–759.

[15] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir, "Scheduling the I/O of HPC applications under congestion," LIP ; INRIA, Research Report RR-8519, Oct. 2014. [Online]. Available: https://hal.inria.fr/hal-00983789

[16] F. Isaila, J. Garcia, J. Carretero, R. B. Ross, and D. Kimpe, "Making the Case for Reforming the I/O Software Stack of Extreme-Scale Systems," Edinburgh, Scotland, 2013.

[17] R. Thakur, W. Gropp, and E. Lusk, "A case for using MPIs derived datatypes to improve I/O performance," in *Proceedings of SC98: High Performance Networking and Computing*. ACM Press, November 1998. [Online]. Available: http://www.mcs.anl.gov/ thakur/dtype/

[18] W. Gropp, "Mpich2: A new start for mpi implementations," in *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. London, UK, UK: Springer-Verlag, 2002, pp. 7–. [Online]. Available: http://dl.acm.org/citation.cfm?id=648139.749473

[19] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Luki, S. Sehrish, and W. keng Liao, "Hacc: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49 – 65, 2016. [Online]. Available: //www.sciencedirect.com/science/article/pii/S138410761500069X

Leadership Computing Facility
Argonne National Laboratory
9700 South Cass Avenue, Bldg. #240
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC