Understanding Parallel I/O Performance Trends Under Various HPC Configurations

Hanul Sung
Department of Computer Science and Engineering, Seoul National University

Jiwoo Bang
Department of Computer Science and Engineering, Seoul National University

Alexander Sim
Lawrence Berkeley National Laboratory

Kesheng Wu
Lawrence Berkeley National Laboratory

Hyeonsang Eom
Department of Computer Science and Engineering, Seoul National University

ABSTRACT

In high-performance computing (HPC) environments, an appropriate amount of hardware resources must be used for the best parallel I/O performance. For this reason, HPC users are provided with tunable parameters to change the HPC configurations, which control the amounts of resources. However, some users are not well aware of a relationship between the parallel I/O performance and the HPC configuration, and they thus fail to utilize these parameters. Even if users who know the relationship, they have to run an application under every parameter combination to find the setting for the best performance, because each application shows different performance trends under different configurations. The paper shows the result of analyzing the I/O performance trends for HPC users to find the best configurations with minimal efforts. We divide the parallel I/O characteristic into independent and collective I/Os and measure the I/O throughput under various configurations by using synthetic workload, IOR benchmark. Through the analysis, we have figured out that the parallel I/O performance is determined by the trade-off between the gain from the parallelism of increased OSTs and the loss from the contention for shared resources. Also, this performance trend differs depending on the I/O characteristic. Our evaluation shows that HPC applications also have similar performance trends as our analysis.

CCS CONCEPTS

• **Computer systems organization** → *Real-time operating systems*; • **Software and its engineering** → *Software development methods*;

KEYWORDS

High performance computing; Parallel I/O performance; Supercomputer;Performance tuning;

SNTA'19, June 25, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6761-5/19/06…$15.00
https://doi.org/10.1145/3322798.3329258

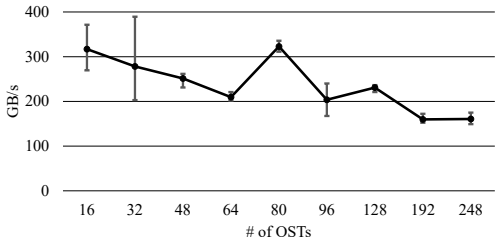


Figure 1: Performance fluctuation on Cori with IOR benchmark

ACM Reference Format:
Hanul Sung, Jiwoo Bang, Alexander Sim, Kesheng Wu, and Hyeonsang Eom. 2019. Understanding Parallel I/O Performance Trends Under Various HPC Configurations. In *Systems and Network Telemetry and Analytics (SNTA'19), June 25, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3322798.3329258

1 INTRODUCTION

As the petascale era approaches, HPC applications have begun to utilize hundreds of cores simultaneously using huge amounts of data [22]. The role of the parallel shared file system has become important in order to manage such amounts of data quickly and accurately. Many supercomputers (i.e., Jaguar system at Oak Ridge National Laboratory [17] [13] and Cori system at National Energy Research Scientific Computing Center [5]) provide users with various tunable parameters such as the number of compute nodes, the number of cores, the number of OSTs and stripe size, for efficient use of this file system. By adjusting the parameter settings, the users can find the best configurations showing high I/O performance.

However, there are several limitations. First, according to the Cori log, most users are using the default configuration provided by Cori system. In other words, since the users are unfamiliar with their HPC environments, they do not use these parameter settings properly and get unexpected I/O performance. Second, there are too many combinations of the parameters to be considered for experiments in order to achieve the best performance [7] [10] [12]. The I/O characteristic for each application is different and the I/O performance differs depending on the HPC system. Therefore, in order to obtain the configuration for the best performance, HPC

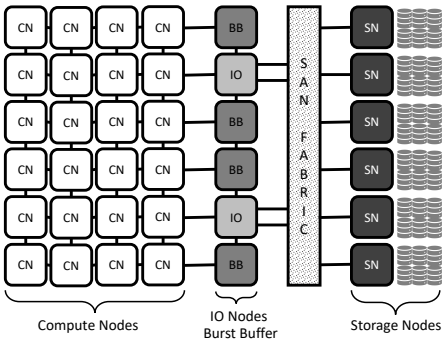


Figure 2: Cori Architecture

applications have to be executed in every combination of the parameter settings. However, this brute-force method is practically impossible because there are a huge number of combinations. In addition, it may take several hours until a job finishes executed after being requested in a supercomputer such as Cori system, especially when a large number of compute nodes or a large number of OSTs are required simultaneously. Therefore it may take several days to measure the I/O performance in every configuration. Third, HPC users sometimes experience performance fluctuations, because they share many hardware resources with others. Thus it is difficult to get expected or correct I/O performance in a single experiment per configuration. Figure 1 shows the performance fluctuations of IOR benchmark. IOR runs three times under the same configuration. It shows similar results in some configurations, but frequently the performance difference is quite large up to twice. Forth, some HPC users expect to get high I/O performance when they have a lot of hardware resources allocated. However, against expectations, they may get lower I/O performance despite the use of more resources.

As a result, it is necessary to find an easy way to get the best configuration with minimal efforts for highest I/O performance. To do this, we analyze performance trends by adjusting the tunable parameter settings with the synthetic benchmark, IOR, in Cori system. The HPC I/O characteristic is divided into independent and collective I/O, and the number of compute nodes, the number of cores, and the number of OSTs are used as the tunable parameters. Based on the IOR results, each I/O characteristics shows specific performance trend and other HPC applications such as Chombo [1] and VPIC-IO [8] have the trends similar to our analysis.

2 BACKGROUND

2.1 Cori Supercomputer

Cori system, NERSC's supercomputer, is Cray XC40 as shown in Figure 2 [3]. Cori consists of 2,388 Intel Xeon "Haswell" processor nodes and 9,688 Intel Xeon Phi "Xnights Landing" nodes (KNL). Each Haswell nodes has two 16-core Intel Xeon Processor E5-2698 v3 at 2.3GHz and each KNL nodes has a 68-core Intel Xeon Phi processor 7250. Cori also has a 1.8TB Cray Data Warp Burst Buffer with the performance of 1.7TB/s, but this feature is not used in this paper.

All these nodes are connected to the Cray "Aries" high-speed inter-node network using Dragonfly topology, giving the global

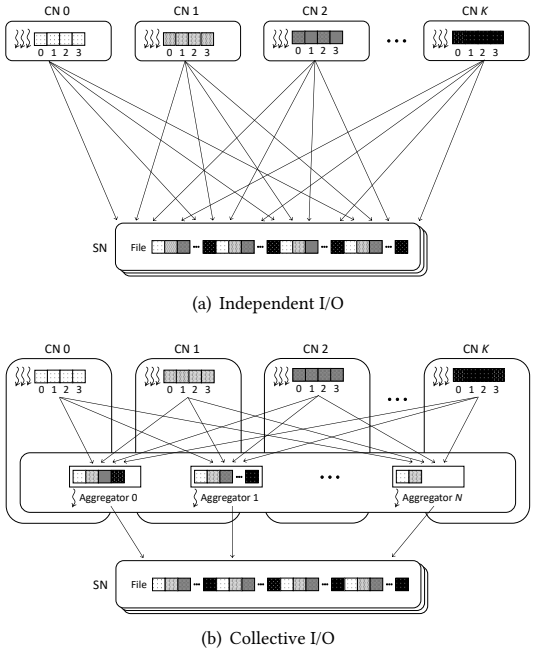


Figure 3: Parallel I/O operation

bandwidth of 5.625TB/s in Haswell nodes and the bandwidth of 45TB/s in the KNL nodes. Cori uses Lustre scratch file system for disk storage and efficient I/O performance [4]. It consists of 248 IO servers (OSS) including 41 hard disks and 248 OSTs, providing total 27TB of storage. It supports peak performance of 744GB/s.

2.2 Parallel I/O Operation

The MPI I/O operation is divided into the independent and the collective I/Os [11]. In the independent I/O, each MPI processes handles the I/O operations independently on its own data. In Figure 3(a), all the processes in the K compute nodes(CN) issue the I/O operations simultaneously. If there are many small I/O operations to handle, each process has to access non-contiguous locations in disks many times and results in low I/O performance. To solve this problem, the collective buffering I/O is provided.

The collective I/O is divided into two I/O phases [9]. In the first phase, several MPI processes called aggregators merge other processes' data into the temporary buffer making contiguous large chunks. In the second phase, the corresponding chunk is written to one-to-one mapped OSTs by the aggregators when their buffer is full. In Figure 3(b), the processes send their own data to aggregators' buffer. Then, N aggregators issue the file write operations to their mapped OST when the buffer is full. In the collective I/O, since only the aggregators participate in the file I/O, there is less contention and results in high performance.

3 PARALLEL I/O PERFORMANCE TREND

We analyze the performance trends of two different I/O characteristics, the independent and the collective I/O, under the various configurations. IOR benchmark is executed with sequential write

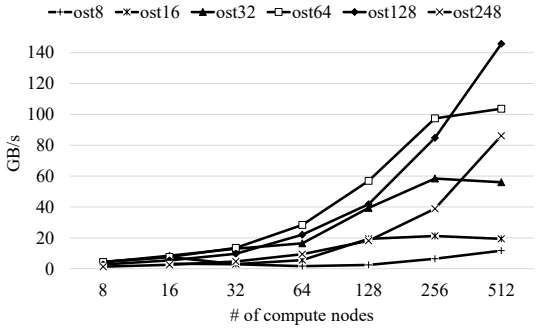


Figure 4: Independent I/O performance trend depending on the number of compute nodes

operations in the single shared file. The various configurations are generated by adjusting the number of compute nodes, the number of cores per compute node, and the number of OSTs, excluding the stripe size. The parallel I/O performance is calculated by dividing the output size by the average write time of MPI processes. Even though actual MPI performance should be determined by the write time of the slowest MPI process, since the hardware resources in Cori are shared by many users, the performance fluctuation is shown frequently. So multiple times of experiments are needed under the same configuration to get stable performance results. For this reason, we run IOR three times under same configuration and use the average write time of MPI processes.

3.1 Independent I/O

In order to analyze the performance trends of the independent I/O, we use 8 to 512 KNL compute nodes, 64 cores of each compute nodes, and 8 to 248 OSTs. The output size of all experiments is 512GB and the stripe size is set to 1MB. We set the block size, which is contiguous bytes for each thread to write, to be larger than the stripe size for having enough I/O requests to issue.

$$B = \frac{total\ output\ size}{total\ number\ of\ threads}$$

$$O = \frac{block\ size}{stripe\ size}$$

The formula above shows *the block size*(B) and *the number of accessed OSTs per thread*(O). Since the file I/O requests for the fixed size file are issued across every MPI process, the block size depends on the number of threads. And the number of OSTs that the single thread accesses is determined by dividing the block size by the stripe size. Therefore, if the number of compute nodes is the same, the number of OSTs accessed by a thread is the same even when the number of OSTs changes. However, when the number of OSTs is the same and the number of compute nodes increases, the number of OSTs to be accessed by the single thread decreases due to the decreased block size.

3.1.1 Number of Compute Nodes. Figure 4 shows the MPI throughput (y-axis) on the number of compute nodes used (x-axis) for different number of OSTs. In all experiments, the MPI throughput increases as the number of compute nodes increases because the number of issued I/O requests at the same time increases, resulting

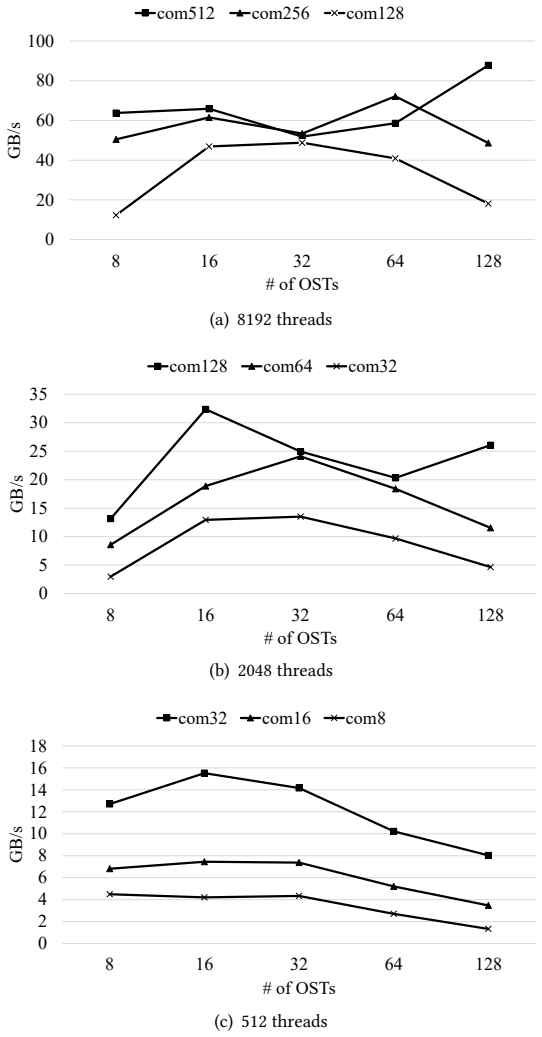


Figure 5: Indenpendent I/O performance trend depending of the number of cores per compute node

in better parallelism. In addition, the larger the number of OSTs, the more requests can be handled at the same time. So, the MPI throughput variation gets larger as the number of OSTs increases.

In summary, if the number of available OSTs is fixed, it is better to use as many compute nodes as possible for high performance.

3.1.2 Number of Cores per Compute Node. We analyze the performance trends depending on the number of cores used in each compute nodes. To do this, we adjust the number of cores used per compute node by changing the number of compute nodes and fixing the total number of threads.

Figure 5 shows the performance trends under three configurations, 16 cores per compute node, 32 cores per compute node, and 64 cores per compute node, each when the total number of threads is 8192, 2048 and 512. Because all MPI threads issue I/O requests simultaneously, there is contention for shared resources such as a cache, a memory, and a network within the single compute node.

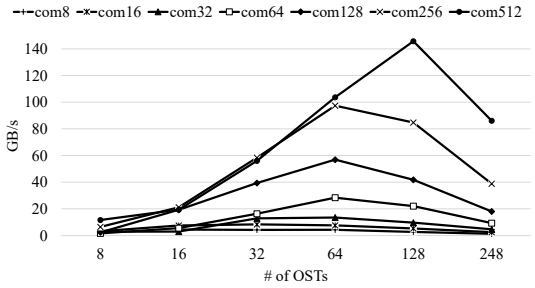


Figure 6: Independent I/O performance trend depending on the number of OSTs

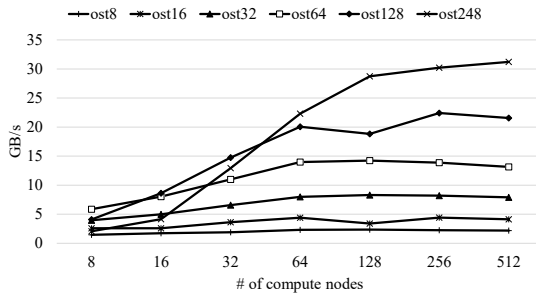


Figure 7: Collective I/O performance trend depending on the number of compute nodes

As the number of compute nodes increases, the contention per compute node is reduced because the number of threads in the single compute node decreases. Thus, in all the figures, the I/O throughput gets higher when using more compute nodes even if the same number of MPI threads are executed.

In summary, if there is enough number of compute nodes, it is better to use the fewer cores per compute node for high performance.

3.1.3 Number of OSTs. In order to analyze the parallel I/O performance trends depending on the number of OSTs, we measure the MPI throughputs by varying the number of OSTs for the different number of compute nodes. As shown in Figure 6, there are two findings.

First, when the number of compute nodes is fixed, the MPI throughput increases as the number of OSTs increases until a certain point. After that, the MPI throughput becomes worse as the number of OSTs increases. If the number of compute nodes is fixed, increasing the number of OSTs increases the number of I/O requests processed concurrently, thereby the parallelism increases. However, the increase in the number of OSTs causes an increase in the waiting time than the I/O service time [21]. One of reasons for this is the contention in the OSTs that are shared with other HPC users. As the number of OSTs shared with other users increases, the interference from them increases. Until the certain point, the MPI throughput is improved because the gain from parallelism is larger than the loss from the contention. But, after the certain point, the gain becomes smaller than the loss and the throughput decreases.

Second, the number of OSTs giving the best performance differs depending on the number of compute nodes. Also, the certain point showing the best performance increases as the number of compute nodes gets larger. As the number of compute nodes increases, the number of MPI threads increases, so that the number of issued I/O requests increases proportionally. Therefore, more OSTs are needed to handle the increased I/O requests for the best performance. As shown in the figure 6, with 8 compute nodes, the performance increases until there are 16 OSTs. In the same aspect, 32 OSTs for 16 and 32 compute nodes, 64 OSTs for 64, 128 and 256 compute nodes, 128 OSTs for 512 compute nodes give the best performance. The tendency shows that the number of OSTs for the best performance gradually increases in proportion to the number of compute nodes. However, in some cases, despite the number of compute nodes differs, the best performance is shown in the same number of OSTs. This is because only the 2^n number of OSTs is used in the experiments, we cannot find the exact number of OSTs for the best performance. So we run IOR benchmark with 48 OSTs and 64 compute nodes for an additinal experiment and get higher throughput, 30.81GB/s than one with 64 OSTs, 28.14GB/s. This verifies that the number of OSTs showing the best performance increases as the number of compute nodes increases.

In summary, when the number of compute nodes used is fixed, the parallel I/O performance increases only until there is certain number of OSTs. And the number of OSTs showing the best performance gets larger as the number of compute nodes increases.

3.2 Collective I/O

In this section, we analyze the collective I/O performance trends under the various configurations. For all the experiments, we use 8 to 512 compute nodes, 32 cores per compute node, 8 to 248 OSTs, 16MB stripe size and 512 GB output size.

$$A = \frac{number\ of\ OSTs}{number\ of\ compute\ nodes}$$

The formula above shows *the number of aggregators per compute node*(A). Since the number of aggregators is equal to the number of OSTs, the number of aggregators per compute node is calculated by dividing the total number of OSTs by the number of compute nodes. So when the number of compute nodes is not changed, as the number of OSTs increases, the number of aggregators of each compute nodes increases. On the contrary, when the number of OSTs is fixed and the number of compute nodes increases, the number of aggregators per computer node is reduced.

3.2.1 Number of Compute nodes. Figure 7 shows the performance trends of the collective I/O, when the number of OSTs fixed and the number of compute nodes is changed. Regardless of the number of OSTs, the I/O throughput is improved as the number of compute nodes increases and saturates from a certain point. If the number of OSTs is fixed, the total number of aggregators is also fixed. In this situation, if the number of compute nodes increases, the number of OSTs per compute node decreases. In other words, the aggregators clustered on a small number of the compute nodes are increasingly spread to the multiple compute nodes. Since the aggregators share the hardware resources such as the cache, the memory and the network, they experience the contention for the shared resources. Therefore, the number of aggregators within a

compute node decreases, the contention is reduced and the I/O requests are issued faster. However, if the number of aggregators per compute node becomes small enough, the contention is negligible, so even if the number of compute nodes increases, the performance does not improve.

Also, the larger the number of OSTs, the greater the performance improvement with the increasing number of compute nodes. As the number of OSTs gets larger, the number of aggregators in the single compute node increases which results in the greater contention. In this case, as the number of compute nodes increases, the performance increases greatly because the contention decreases by a lot. For example, if there are 8 compute nodes, there is one aggregator per compute node when there are 8 OSTs. And when there are 248 OSTs, the number of aggregators per compute node is 31. Since it indicates that half of total threads in a compute node are the aggregators, the contention is severe. Therefore, as the number of compute nodes increases, the aggregators become more dispersed and the performance increases rapidly.

In summary, if larger the number of available OSTs, the greater the performance improvement can be achieved with more compute nodes.

3.2.2 Number of Cores Per Compute Node. In the same way as the section 3.1.2, we analyze the performance trends depending on the number of cores used in the single compute node. Figure 8 shows the effect of the contention of the aggregators per compute node on the performance. In the figure 8(a), regardless of the number of compute nodes, all the I/O throughputs are similar. If there are 248 OSTs, the number of aggregators per compute node is 0.5, 1 and 2 when 64, 32 and 16 cores are used. It indicates that there are too few aggregators to make the severe contention in the single compute node. In the figure 8(b), if there are 248 OSTs, the number of aggregators per compute node is 2, 4 and 8 which is larger than the figure 8(a)'s one. So more shared resource contention occurs. Therefore, the performance increases greatly as the number of compute nodes increases. From the figure 8(c), there are more aggregators per compute node than the previous figures, which results in the more serious contention. So the performance increases most greatly as the number of compute nodes increases.

In summary, the larger the number of aggregators in the single compute node, the better performance can be acheived by spreading the aggregators to more compute nodes.

3.2.3 Number of OSTs. Figure 9 shows the I/O throughputs depending on the number of OSTs. In the collective I/O, the performance trends have similar features shown in the independent I/O.

First, the performance increases until a certain point and decreases when there are more OSTs. Second, the number of OSTs showing the best performance is different depending on the number of compute nodes. Since the number of aggregators is determined by the number of OSTs, if the number of aggregators is small, the parallelism is also small, and if the number of aggregators is large, the parallelism is also large. However, if there are a large number of aggregators per compute node, the contention for the shared resources becomes severe. Therefore, the performance changes depending on the trade-off between the gain from the parallelism and the loss from the contention.

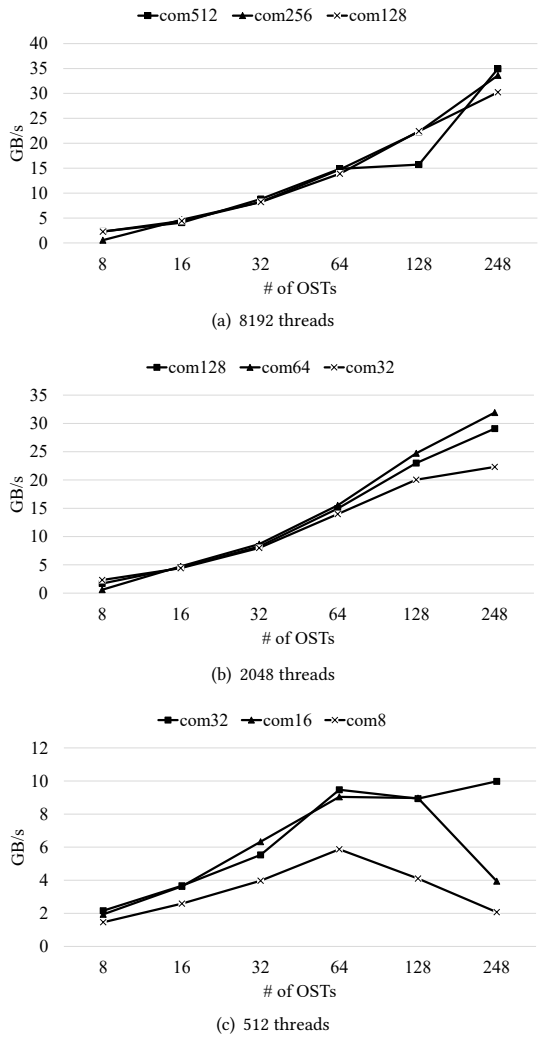


Figure 8: Collective I/O performance trend depending of the number of cores per compute node

In the figure, when there are 8 compute nodes, the best performance is shown when the number of OSTs is 64. As the number of OSTs increases from 8 to 64, the number of aggregators per compute node increases from 1 to 8. When the number of OSTs is larger than 64, the number of aggregators increases from 16 to 32, which results in the reduced performance due to the increased shared resource contention. When there are 32 compute nodes, the number of OSTs showing the best performance is larger than the previous case. The reason is that as the number of compute nodes increases, the number of I/O requests issued at the same time gets larger, requiring more OSTs for the best performance. When there are more than 128 compute nodes, the number of aggregators per compute node is at most 2, causing the small contention. As a result, the I/O throughput continuously increases as the number of OSTs increases.

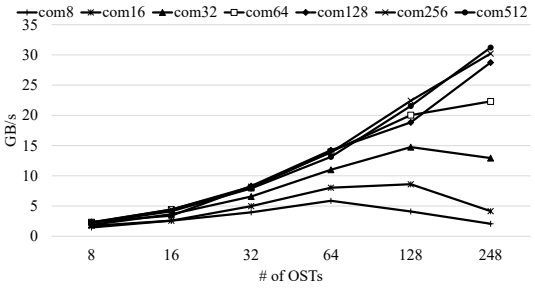


Figure 9: Collective I/O performance trend depending on the number of OSTs

In summary, when the number of aggregators in each compute nodes is small (less than 4 OSTs in these experiments), it is better to use more OSTs for better performance.

4 EVALUATION

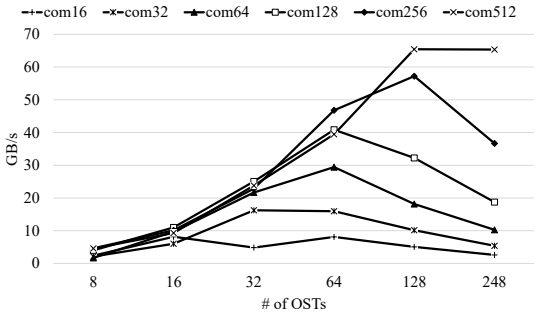
In this section, we compare the performance trends of the synthetic benchmark IOR we analyzed, to the trends of other HPC workloads. We use VPIC-IO workload for the independent I/O and Chombo I/O benchmark for the collective I/O. VPIC-IO is an I/O kernel of VPIC(vector particle-in-cell) plasma physics simulation's particle data write phase [8]. VPIC-IO writes to a H5Part file using a HDF5 I/O library as well. Chombo I/O is created from a Chombo framework which is for the adaptive mesh refinement scientific applications [1]. Chombo uses the HDF5 I/O library for the parallel computations over block-structured, adaptively refined grids. We set the configurations and the output size same in VPIC-IO and Chombo with IOR.

4.1 Parallel IO Performance Trend Comparison

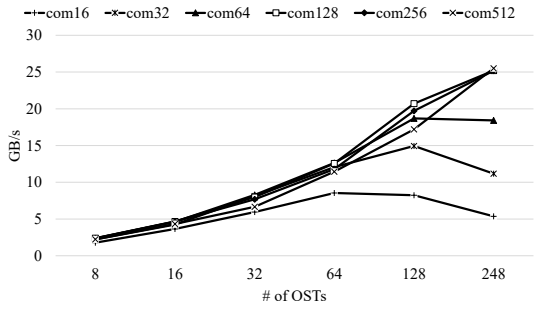
Figure 10 shows the performance trends of VPIC-IO and Chombo depending on the number of the compute nodes and the number of OSTs.

Even if VPIC-IO uses the same configuration and output size as IOR with the independent I/O, the number of OSTs accessed by the single thread is different because VPIC-IO and IOR have different ways of selecting the range of each thread for the file I/O. But as the configuration changes, the block size and the number of OSTs accessed by the single thread change at a rate equal in VPIC-IO and IOR. As shown in the figure 10(a), VPIC-IO shows similar performance trends to IOR with the indenpendent I/O. There is the optimal number of OSTs for the best performance, which increases as the number of compute nodes increases. The number of OSTs is even similar to IOR's in the figure 6. VPIC-IO shows the best performance when there are 16 OSTs if 16 compute nodes are used. In the same aspect, 32 OSTs for 32 compute nodes, 64 OSTs for 64, 128 compute nodes, and 128 OSTs for 256, 512 compute nodes give the best performance.

The performance trends of Chombo under the different configurations are shown in the figure 10(b). Chombo also shows the similar performance trends as IOR with the collective I/O. The performance increases until a certain point when the loss from the contention



(a) VPIC-IO



(b) Chombo

Figure 10: I/O performance trend of VPIC-IO and Chombo

exceeds the gain from the parallelism, and then decreases. Chombo also has the same optimal number of OSTs to IOR's one.

In conclusion, we observed that the performance trends of the other HPC workloads can be predicted by the analysis through the synthetic workload IOR. In addition, it is expected to be able to predict the best configuration that gives the highest performance.

5 RELATED WORK

It is not easy for HPC users to find an optimal configuration among a wide range of parameter combinations for the best performance of I/O workloads. So there have been several studies to do this in the HPC environment.

There are several works analyzing the performance trend under the various I/O configurations. The study focusing on the data and metadata intensive I/O patterns is presented in [20]. The work covers the analysis of the contiguous data access pattern with the independent I/O and the small, non-contiguous data access pattern with the collective I/O. The study of I/O performance improvement of Red Storm [2] HPC system is proposed in [14]. The work analyzes the reasons for the I/O bottleneck on a single node test(1 client), a file-per-process test and a shared file test each.

Considering the general performance trends, some studies have utilized a modeling of the I/O workloads on HPC system to predict the best performance. The I/O auto tuning infrastructure for the applications running on the HPC system is developed in [21]. The study focuses on analyzing the I/O characteristic on Lustre file system by applying the mathematical queuing theory model and finds the optimal parameters by modeling the I/O workload as a

vector. However, this framework has the limited applicability and only can be applied to the specific system and the I/O workloads. To improve the previous study's limitation, the autotuning parallel I/O framework by using the empirical model of the I/O performance is presented in [6]. The framework uses the combinations of the parameter values that have the large effect on the I/O performance to get the training set. By repeating pruning, exploring and refitting the trained model, the framework finds the optimal parameter values for the best performance. However this framework is also limited to the specific I/O workloads. The work in [15] proposed performance prediction framework using static analysis. The analysis module generates each application's prediction model by analyzing application's source code statically. But they only predict application's performance and did not give any optimization strategies for achieving the better performance. The IOR benchmark is used to predict the I/O performance of the HPC workloads running on the different HPC system environments in [19]. Each prediction model is designed by changing the IOR parameters according to the I/O patterns of each targeting HPC workloads. However, it is difficult to analyze the I/O patterns for each target applications and have to closely select the IOR parameters accordingly.

6 CONCLUSIONS

To efficiently find the best HPC configuration that gives the best performance, we analyzed the performance trends by changing the tunable parameters using the synthetic workload, IOR. The different I/O operations, the independent I/O and the collective I/O, showed specific performance trends depending on the configurations which consist of the number compute nodes, the number of cores per compute node and the number of OSTs. In both I/O characteristics, the performance trends changed due to the parallelism and the contention for the shared resources, and the best performance was determined by appropriate trade-off between the two. The result of applying the analysis to the HPC workloads, VPIC-IO and Chombo, showed similar performance trends to IOR's one.

Future work: In this paper, we did not consider the stripe size as a parameter for the HPC configuration and set it where each thread has the sufficient parallelism for the experiments. But, because the stripe size affects not only the parallelism of one thread, but also the parallelism of the entire HPC application, it is necessary to analyze the performance trend considering the stripe size [16] [18]. Moreover, since we have focused on the performance trends of the single shared file, we plan to expand this study to include the analysis of performance trend of the file-per-process.

ACKNOWLEDGMENTS

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center. This research was supported by 1)Institute for Information & communications Technology Promotion (IITP) Grant funded by the Korea government (MSIP) (R0190-16-2012, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development). It was also partly supported by 2)National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (NRF-2017R1A2B4004513, Optimizing GPGPU virtualization in multi GPGPU environments through kernels' concurrent execution-aware scheduling), and partly supported by 3)the National Research Foundation (NRF) grant (NRF-2016M3C4A7952587, PF Class Heterogeneous High Performance Computer Development). In addition, this work was partly supported by 4)BK21 Plus for Pioneers in Innovative Computing (Dept. of Computer Science and Engineering, SNU) funded by National Research Foundation of Korea(NRF)(21A20151113068).

REFERENCES

[1] 2017. Software using HDF5: Descriptions. https://support.hdfgroup.org/HDF5/tools5desc.html

[2] 2018. Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core | TOP500 Supercomputer Sites. https://www.top500.org/system/8193

[3] 2019. Cori Configuration. https://www.nersc.gov/users/computational-systems/cori/configuration/

[4] 2019. Cori File Systems. https://www.nersc.gov/users/computational-systems/cori/file-storage-and-i-o/

[5] Wright-N. Cardo N.P. Andrews A. Cordery M. Antypas, K. 2014. Cori: a cray XC pre-exascale system for NERSC. In *Cray User Group Proceedings*.

[6] Babak Behzad, Surendra Byna, Prabhat, and Marc Snir. 2019. Optimizing I/O Performance of HPC Applications with Autotuning. *ACM Transactions on Parallel Computing* 5, 4 (mar 2019), 1–27. https://doi.org/10.1145/3309205

[7] Babak Behzad, Joey Huchette, Huong Luu, Ruth Aydt, Quincey Koziol, Mr Prabhat, Suren Byna, Mohamad Chaarawi, and Yushu Yao. 2012. Abstract: Auto-Tuning of Parallel IO Parameters for HDF5 Applications. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, 1430–1430. https://doi.org/10.1109/SC.Companion.2012.236

[8] Wahid Bhimji, Debbie Bard, Melissa Romanus, Andrey Ovsyannikov, Brian Friesen, Matt Bryson, Joaquin Correa, Glenn K. Lockwood, Vakho Tsulaia, Suren Byna, Steve Farrell, Doga Gursoy, Christopher S. Daley, Vince Beckner, Brian van Straalen, Nicholas J. Wright, and Katie Antypas. 2016. Accelerating Science with the NERSC Burst Buffer Early User Program. https://www.semanticscholar.org/paper/Accelerating-Science-with-the-NERSC-Burst-Buffer-Bhimji-Bard/3037024ee9782764cfbe8e5c9c625e2edaaf83fd

[9] Javier Garc Blas, Florin Isaila, David E. Singh, and J. Carretero. 2008. View-Based Collective I/O for MPI-IO. In *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. IEEE, 409–416. https://doi.org/10.1109/CCGRID.2008.85

[10] Julian Borrill, Leonid Oliker, John Shalf, and Hongzhang Shan. 2007. Investigation of leading HPC I/O performance using a scientific-application derived benchmark. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing - SC '07*. ACM Press, New York, New York, USA, 1. https://doi.org/10.1145/1362622.1362636

[11] Peter Corbett, Dror Feitelson, Sam Fineberg, Yarsun Hsu, Bill Nitzberg, Jean-Pierre Prost, Marc Snirt, Bernard Traversat, and Parkson Wong. 1996. Overview of the MPI-IO Parallel I/O Interface. Springer, Boston, MA, 127–146. https://doi.org/10.1007/978-1-4613-1401-1_5

[12] Hui Jin, Yong Chen, Huaiyu Zhu, and Xian-He Sun. 2010. Optimizing HPC Fault-Tolerant Environment: An Analytical Approach. In *2010 39th International Conference on Parallel Processing*. IEEE, 525–534. https://doi.org/10.1109/ICPP.2010.80

[13] Wayne Joubert and Shi-Quan Su. 2012. An analysis of computational workloads for the ORNL Jaguar system. In *Proceedings of the 26th ACM international conference on Supercomputing - ICS '12*. ACM Press, New York, New York, USA, 247. https://doi.org/10.1145/2304576.2304611

[14] James H. Laros, Lee Ward, Ruth Klundt, Sue Kelly, James L. Tomkins, and Brian R. Kellogg. 2007. Red storm IO performance analysis. In *2007 IEEE International Conference on Cluster Computing*. IEEE, 50–57. https://doi.org/10.1109/CLUSTR.2007.4629216

[15] Mohammad Abu Obaida, Jason Liu, Gopinath Chennupati, Nandakishore Santhi, and Stephan Eidenbenz. 2018. Parallel Application Performance Prediction Using Analysis Based Models and HPC Simulations. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation - SIGSIM-PADS '18*. ACM Press, New York, New York, USA, 49–59. https://doi.org/10.1145/3200921.3200937

[16] Md. Wasi-ur Rahman, Nusrat Sharmin Islam, Xiaoyi Lu, and Dhabaleswar K. DK Panda. 2017. A Comprehensive Study of MapReduce Over Lustre for Intermediate Data Placement and Shuffle Strategies on HPC Clusters. *IEEE Transactions on*

Parallel and Distributed Systems 28, 3 (mar 2017), 633–646. https://doi.org/10.1109/TPDS.2016.2591947

[17] Arthur S Buddy Bland, Jim Rogers, Ricky A Kendall, Douglas Kothe, and Galen M Shipman. 2009. Jaguar: The World's Most Powerful Computer. In *Cray User Group*.

[18] Subhash Saini, Jason Rappleye, Johnny Chang, David Barker, Piyush Mehrotra, and Rupak Biswas. 2012. I/O performance characterization of Lustre and NASA applications on Pleiades. In *2012 19th International Conference on High Performance Computing*. IEEE, 1–10. https://doi.org/10.1109/HiPC.2012.6507507

[19] Hongzhang Shan, Katie Antypas, and John Shalf. 2008. Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark. In *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12. https://doi.org/10.1109/SC.2008.5222721

[20] Weikuan Yu, Jeffrey S. Vetter, and H. Sarp Oral. 2008. Performance characterization and optimization of parallel I/O on the Cray XT. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 1–11. https://doi.org/10.1109/IPDPS.2008.4536277

[21] Haihang You, Qing Liu, Zhiqiang Li, and Shirley Moore. 2011. The Design of an Auto-tuning I/O Framework on Cray XT5 System. *Cray Users Group Conference (CUG'11) (Best Paper Finalist)* (2011). https://www.icl.utk.edu/publications/design-auto-tuning-io-framework-cray-xt5-system

[22] Zhou Zhou, Xu Yang, Dongfang Zhao, Paul Rich, Wei Tang, Jia Wang, and Zhiling Lan. 2015. I/O-Aware Batch Scheduling for Petascale Computing Systems. In *2015 IEEE International Conference on Cluster Computing*. IEEE, 254–263. https://doi.org/10.1109/CLUSTER.2015.45