2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)

DCA-IO: A Dynamic I/O Control Scheme for Parallel and Distributed File Systems

Sunggon Kim*, Alex Sim†, Kesheng Wu†, Suren Byna†, Teng Wang†, Yongseok Son‡, Hyeonsang Eom*
*Seoul National University
†Lawrence Berkeley National Laboratory
‡Chung-Ang University
Emails: *skim@dcslab.snu.ac.kr, hseom@cse.snu.ac.kr, †{asim, kwu, sbyna, tengwang}@lbl.gov, ‡sysganda@cau.ac.kr

***Abstract*—In high-performance computing, storage is a shared resource and used by all users with many different application requirements and knowledge of storage. Consequently, the optimal storage configuration varies according to the I/O behavior of each application. While system logs are helpful resources in understanding the storage behavior, it is non-trivial for each user to analyze the logs and adjust complex configurations. Even for experienced users, it is difficult to understand the full stack of I/O systems and find the optimal configuration for the specific application. In this work, we analyzed the I/O activities of CORI which is an HPC system in National Energy Research Scientific Computing Center (NERSC). The result of our analysis shows that most users do not adjust storage configurations and use the default settings. Also, it shows that only a few applications are executed repeatedly in the HPC environment. Based on this result, we have developed DCA-IO, a dynamic distributed file system configuration adjustment algorithm, which utilizes system log information and widely adapted rules to adjust storage configurations automatically without any user intervention. DCA-IO utilizes existing system logs and does not require any modifications in code or an additional library. To demonstrate the effectiveness of DCA-IO, we have performed experiments using I/O kernels of the real applications in both isolated small-sized Lustre environment and CORI. Our experimental result shows that the use of our scheme can lead to improvements in the performance of HPC applications by up to 75% in an isolated environment and 50% in a real HPC environment without user intervention.**

***Keywords*-HPC System; Dynamic Control; System Logs; Parallel and Distributed File System; Cloud System**

I. Introduction

High-performance computing (HPC) is becoming widely adapted to the industry due to the increasing demand for large-scale computation and big data. HPC applications have many different characteristics compared with traditional applications in that they utilize a large amount of computational power. Due to such increased computation capabilities, HPC applications often produce a significantly large amount of data compared with traditional applications [1]. Due to a large amount of data, many HPC applications perform checkpointing which records the intermediate data to the storage to protect data from unexpected power-outage or scheduling. Since applications blindly wait for the completion of I/O before starting further computation, the performance of the application is strongly related to the I/O performance. Thus, it is becoming increasingly important to improve I/O performance to increase the overall utilization of HPC system.

Since the storage architecture in the HPC environment is inherently different from the traditional one, careful considerations need to be made in order to efficiently exploit the I/O performance in the HPC environment. For example, instead of local file systems such as EXT4 [2] or XFS [3], parallel and distributed file systems such as Lustre [4] and Ceph [5] are widely used in many HPC environments to achieve high performance, reliability, and scalability. To utilize multiple nodes of distributed file system in parallel, many parallel and distributed file systems provide various configuration options so that users can specify the number of nodes to place data in (stripe count) and the size of the data chunk to be placed in each node (stripe size). To fully exploit the I/O performance of parallel and distributed file system, it is important to analyze the I/O behavior of the application and adjust the configurations according to the I/O behavior of the application.

In the previous studies, the researchers tried to improve the I/O performance of applications by understanding their I/O behaviors and adjusting distributed file system configurations. Yu et al. [6] characterized the I/O patterns from the applications and proposed the optimal Lustre configurations depending on the characteristics based on the result of experiments. You et al. [7] proposed an auto-tuning framework which models the application and ran the model in a separate system with multiple configurations to find the optimal configuration. Our study is in line with these studies in finding the optimal configuration by adjusting the configurations.

In this paper, we first present the result of analyzing the of I/O activities in CORI which is an HPC environment in National Energy Research Scientific Computing Center (NERSC), Lawrence Berkley National Laboratory. Although many previous researches [1], [8] denotes that the use of the optimal configuration can lead to improvements in the application performance by orders of magnitude, the result of our analysis shows that the vast majority of users use the default configuration.

To improve the I/O performance of applications and overall storage utilization, we have developed DCA-IO, an

978-1-7281-0912-1/19/$31.00 ©2019 IEEE
DOI 10.1109/CCGRID.2019.00049

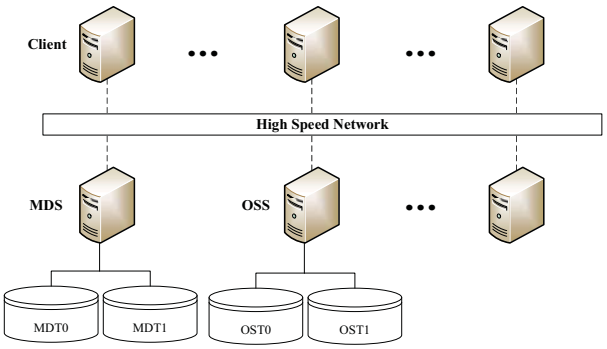


Figure 1: Lustre file system architecture.

algorithm which dynamically configures Lustre file system configurations. When a new application is submitted and there is no information of I/O behavior on the submitted application, DCA-IO utilizes statistical analysis of other applications which previously ran on the HPC system to minimize the modeling and training overhead. By analyzing the previous history of applications in the same environment, DCA-IO can adjust the configuration without the knowledge of I/O behavior of the submitted application. After the application is executed and the information is available, DCA-IO utilizes the information from the previous executions and optimizes configurations using a set of rules. Finally, DCA-IO continues to improve the distributed file system configurations dynamically as the application recurs multiple times. Our experimental results show that the use of the proposed algorithm can lead to improvements in the I/O performance of the applications by up to 75% in an isolated environment and 50% in CORI.

Our contributions are as follows:

- We analyzed the I/O configurations in the existing HPC environment.
- We designed and implemented DCA-IO which can improve the performance by utilizing existing system logs from other application executions and dynamically changes the distributed file system configurations.
- We demonstrated that DCA-IO can improve the I/O performance of many scientific applications in both small and large environment.

The rest of this paper is organized as follows: Section II describes the background and motivation. Section III presents the analysis result for the CORI HPC environment. Section IV presents the design of DCA-IO. Section V shows the experimental results. Section VI discusses the related work. Section VIII concludes the paper.

II. Background

A. Lustre file system

Lustre file system [4] is a parallel and distributed file system which is used in many HPC environments including CORI. Figure 1 shows overall architecture of Lustre file system. Lustre is consist of two main servers.

- Metadata server (MDS) stores and serves metadata of the file system such as file names, permission information, and directories. Each MDS is consist of one or more metadata targets (MDT) which are disks used to store actual data.
- Object storage server (OSS) stores the file data on one or more object storage targets (OST). The maximum throughput and capacity of OSS are a sum of each OST's maximum throughput and capacity, respectively.

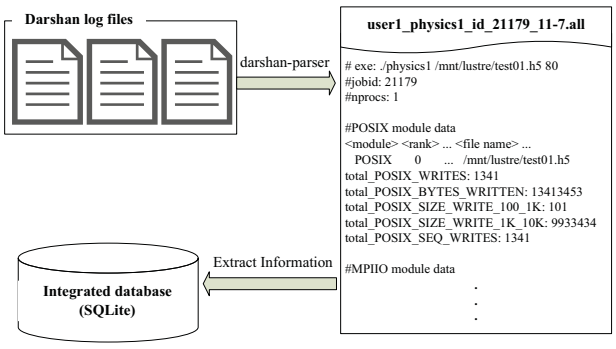
When a client creates and writes a new file, the file can be distributed over multiple OSSes with different sized file chunks which can be configured using stripeCount and stripeSize parameters, respectively. By adjusting stripeCount, the client can improve the parallelism since multiple OSSes can be used in a parallel manner. By adjusting stripeSize, the data from the certain process can be stored in a contiguous space. The performance of applications can be improved by several order of magnitude with ideal stripeCount and stripeSize [7].

B. Analyzing and optimizing I/O performance in HPC environment

To analyze the application behavior, many previous studies proposed system-wide tools to understand application behavior in the HPC environment [9], [10]. In the perspective of I/O, Darshan I/O characterization tool which is developed by Argonne National Laboratory is being widely used in many HPC environments [11]. When the application compiles, Darshan inserts codes which intercept *MPI_Init()* to initialize Darshan data structures and *MPI_Finalize()* to terminate Darshan process. When the application runs, Darshan captures I/O related function calls from HPC applications on per-file and per-process basis in a light-weight manner. After the application terminates, it aggregates the collected information and writes them in a file format. Since Darshan has negligible overhead and captures the complete record I/O function calls, it has been used in many previous studies to understand I/O behavior and to create an application specific modeling [11], [12].

III. Analysis

In this section, we explain the methodology to collect information from existing Darshan logs of CORI and present analysis result.



progName	userID	jobID	numProc	numOST	stripeSize	seqWrite (%)	requestMore1M (%)
physics1	user1	21179	1	4	1048576	100	74.3
physics1	user5	21230	8	4	1048576	100	74.3
biology1	user3	24321	16	1	1048576	35.5	43.1
biology2	user3	26341	8	16	8388608	91.1	99.8
physics1	user2	29031	64	2	8388608	84.2	34.1
physics2	user4	30231	2	1	1048576	32.1	100

Figure 2: Overview of creating an integrated database from Darshan logs.

Name	Description
ProgName	Name of the program
UserName	Name of the user
RunTime	Duration of the application
NumProcs	Number of processes
StripeCount	Number of OSTs used by the application
StripeSize	Number of OSTs used by the application
NumFile	Number of Files used by the application
SeqIOPct	Percentage of read/write requests that are sequential
IOLess1K	Number of read/write requests that are less than 1K
IO1Kto100K	Number of read/write requests that are less than 100K
IOReadRequest	Number of read requests
IOWriteRequest	Number of write requests
IOBytesTotal	Total bytes read/written by the application
IOTimeTotal	Total read/written time used by the application
IOThroughputTotal	Total read/write throughput by the application

Table I: List of information extracted from Darshan logs.

A. Collecting Darshan Logs

To find the I/O activities of HPC applications and Lustre file system configuration that user used, we have analyzed Darshan logs from CORI over two months (October-November 2017). In CORI, Darshan is configured as the default I/O characterization tool and Darshan logs are stored automatically after each application execution [13]. Since the logs are stored in a raw file format, logs need to go through few transformations as shown in Figure 2.

Darshan logs first need to be transformed into a human-readable text format using the darshan-parser utility [14]. After the transformation, the text file contains information such as program name, arguments, number of processes, and I/O activities for each I/O module (POSIX, MPIIO, and STDIO). Since this information is in file format and each application execution has its own file, it can be difficult to find the overall tendency of the applications.

To find overall I/O activities of applications, we have implemented parser which extracts key information from the parsed Darshan text files and builds an integrated database. For the database engine, we have selected SQLite [15] since it is lightweight, easy to use, and support portability. By creating an integrated database, users can perform queries on various key information to find out the overall tendency of applications in the context of the whole HPC environment rather than each application.

Table I shows the list of information which we extracted from Darshan log and inserted to the integrated database. While other information can be directly retrieved from Darshan log, StripeCount, IOTimeTotal, and IOThroughputTotal need to be computed. Following are the method we used to compute that information.

- **StripeCount** When Darshan collects the I/O information, it checks whether the application utilizes Lustre file system and compiled with the Lustre module enabled [13]. If the lustre module is enabled, Darshan records the LUSTRE_OST_ID which is the OST_ID used in that specific I/O function call. While extracting the information, we simply track the number of OSTs involved in I/O during the application run and record the information to the integrated database.
- **IOTimeTotal and IOThroughputTotal** Since Darshan collect the I/O duration on the basis of the function call, many previous studies have different approaches when calculating total I/O time of an application. Wang et al. [16] calculate I/O time by measuring the critical section. This is because when an application uses MPIIO module, many concurrent I/O processes can perform the I/O functions concurrently thus accumulating all the duration of I/O functions can be inaccurate. In this paper, we use the approach used by Luu et al. [17]. This approach measures the I/O time per process and uses the largest I/O time of all process. By using IOTimeTotal, we calculated I/OThroughputTotal which is IOBytesTotal divided by IOTimeTotal.

B. Analysis of Darshan Logs

With the integrated database from the previous section, we have analyzed Lustre file system configuration used by users in the HPC environment. Table II and Table III shows the analysis result of stripe count and stripe size, respectively. As shown in both tables, we have discovered that most users do not adjust Lustre file system configuration, but use the default configuration instead. Table II shows that 99.317% of executions use the default strip count which is 1 OST while there are 256 OSTs available in CORI. Similar to stripe count, Table III shows that 99.948% of executions use default stripe size which is 1048576 (1 Megabyte).

StripeCount	Number of Executions	Percentage
1	1275869	99.317%
2	39	0.003%
3-4	62	0.005%
5-8	269	0.021%
9-16	6850	0.533%
17-32	443	0.034%
33-64	374	0.029%
64-128	450	0.035%
129-256	287	0.022%
Total	**1284643**	**100%**

Table II: Result of analyzing stripe count.

StripeSize (Byte)	Number of Executions	Percentage
1048576	1283980	99.948%
4194304	1	0.000%
8388608	480	0.037%
16777216	162	0.013%
33554432	6	0.000%
50331648	4	0.000%
67108864	9	0.001%
100663296	1	0.000%
Total	**1284643**	**100%**

Table III: Result of analyzing stripe size.

This analysis shows that even at the HPC environment where users can exploit far more OSTs compared with the traditional computing environment, users do not adjust Lustre file system configuration and dynamic configuration control is needed to fully exploit the I/O capabilities of HPC environment.

In addition, we also analyzed the number of unique applications among 1284643 runs. This is done by querying the database on distinct application name. The result shows that there are only 1163 unique applications which were executed during the two-month period. Since there are 1284643 executions during that period, it shows that the small number of applications was executed multiple times. Thus, by utilizing information from the previous execution, the performance of each additional execution can improve the performance since there is a high chance that the application will run in the near future.

IV. Design

In this section, we present DCA-IO algorithm to dynamically control Lustre file system configuration. DCA-IO is divided into two parts which are the initial execution and the recurring execution. Initial execution refers that there is no prior knowledge on the incoming application and the system needs to make a blinded guess without knowing the I/O behavior of the application. Recurring Run refers that there are entries in the integrated database that matches the application name. Thus, we can utilize Darshan log from the previous execution to optimize the configurations.

PROCEDURE 1 DCA-IO algorithm for initial execution.

```
1: New Application Request
2: /* check the number of processes */
3: currNumProcs = current Number of Processes
4: stripeCounts[]
5: stripeSizes[]
6: records[] = SELECT * FROM database WHERE numProcs == currNumProcs
7: for item in records
8:       if record.stripeCount is Unique
9:           stripeCounts.add(record.stripeCount)
10:      if record.stripeSize is Unique
11:          stripeSizes.add(record.stripeSize)
12:
13: stripeCount, stripesize = 0
14: for item in stripeCounts
15:      tempThroughput = aveThroughputOfItem
16:      if tempThroughput > stripeCount
17:          stripeCount = item
18: for item in stripeSizes
19:      tempThroughput = aveThroughputOfItem
20:      if tempThroughput > stripeSize
21:          stripeSize = item
22: lfs setstripe -c stripeCount -S stripeSize
```

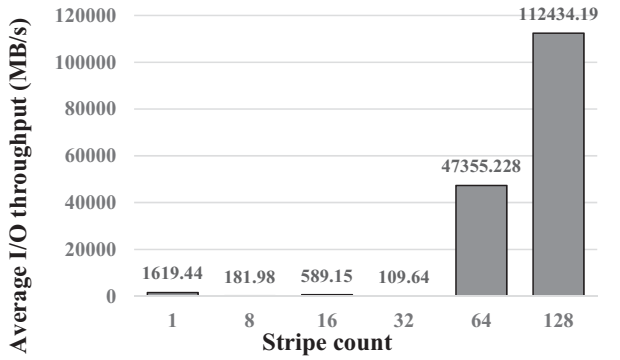


Figure 3: Average I/O throughput per each stripe count when the number of processes is 1.

A. Initial Execution

In case of the initial execution, there is no information about the application since there is no Darshan log available for the incoming application. Thus, it is impossible to make an adjustment based on application behavior. Instead, DCA-IO utilizes the number of processes since the user already specifies the number of processes by requesting the resources. With the number of processes, DCA-IO utilizes existing Darshan logs of other applications in the same HPC environment. Although there is no guarantee that existing Darshan logs are related to the incoming application, since they share same hardware which is related to the performance of application [1], DCA-IO makes a statistical guess based on the existing Darshan logs.

Procedure 1 shows a simplified algorithm for handling new application. When the application arrives, DCA-IO first records the number of processes provided by the user. Then it uses the integrated database to select the entries which have an identical number of processes as the incoming application. Then, it extracts a unique stripe count from

PROCEDURE 2 DCA-IO algorithm for recurring execution.

```
1: Recurring Application Request
2: /* 2nd Execution - rule-based phase*/
3: if file-per-process
4:       stripeCount = 1
5: if shared file
6:       stripeCount = numIOProcs
7: stripeSize = 1M
8: lfs setstripe -c stripeCount -S stripeSize
9:
10: /* 3rd and more Executions - heuristic phase */
11: if IOthroughput > previousIOthroughput
12:      stripeCount = previousStripeCount * 2
13:      if stripeCount > maxAvailStripeCount
14:           stripeCount = maxAvailStripeCount
15: else
16:      stripeCount = previousStripeCount
17:
18: if stripeCount == previousStripeCount
19:       if IOthroughput > previousIOthroughput
20:            stripeSize = previousStripeSize * 2
21: else
22:      stripeSize = previousStripeSize
23: lfs setstripe -c stripeCount -S stripeSize
```

the entries and calculates the average I/O throughput per unique stripe count. Finally, we set the stripe count of the application as a stripe count of the highest average I/O throughput. DCA-IO repeats an identical algorithm to adjust stripe size as well.

For example, when the number of processes requested by the incoming new application is 1, DCA-IO can refer to the entires which used 1 processes from the integrated database. Figure 3 shows the average I/O throughput per unique stripe count from the integrated database. As shown in the figure, the unique stripe counts according to the integrated database are 1, 8, 16, 32, 64, and 128. Since the average I/O performance of 128 stripe count is the highest, the stripe count will be set as 128. Thus, when there is no information on the incoming application, DCA-IO can make an educated adjustment based on Darshan logs from other applications which share identical hardware.

B. Recurring Execution

In case of the recurring execution, Darshan logs with identical application name exist and I/O behaviors of the application can be utilized for the configuration adjustment. DCA-IO optimizes the configuration when the I/O behavior is available in two phases: rule-based phase and heuristic phase.

In the rule-based phase, DCA-IO optimizes the configuration using the existing rules from many previous studies [18], [19]. If the I/O behavior of the application is *file-per-process* where each file is used by a single process, the number of processes which can access to a single file is 1. Thus, we first start with stripe count as 1 since using multiple stripe counts can increase the contention between multiple processes and the overhead of communication. In the case of *single shared file*, where multiple processes can access shared files, we set stripe count as the number of processes participating in I/O since multiple process can utilize high stripe count.

In both cases, DCA-IO sets the stripe size as 1M which is the smallest possible and default configuration in Lustre file system. This is due to two reasons. First, Darshan does not record request sizes of the applications but the size interval that requests belong to. Darshan classifies requests according to the range of the request size and records the number of requests that belong to each interval (e.g., 1K - 100K). Without knowing the specific request sizes of the application, using a large stripe size can create misaligned stripes in a file which can decrease the performance significantly [20], [21]. Second, Lustre suffers less from a small stripe size than a large stripe size. According to the previous studies [20], [22], Lustre already aggregates small striped requests until they match the stripe alignment which decreases the overhead of using a small stripe size. Thus, rather than starting from a large stripe size, DCA-IO sets stripe size as minimum and gradually increase the size during the heuristic phase.

In heuristic phase, DCA-IO increases the stripe count linearly until the performance decreases or the stripe count reaches the maximum available OSTs in the system. The reasoning for this algorithm defers for each access pattern. In case of *file-per-process*, the I/O performance of each file is bound to the maximum performance of an OST since the stripe count is set as 1 during the rule-base phase. However, if the application generates a large amount of I/O in a rapid frequency, the maximum performance of a single OST can be insufficient to handle the I/O requests for a file. Thus, DCA-IO tests a larger stripe count to check if the performance of the application is bound by the limited number of stripe count. In the case of *shared-file*, multiple processes can access the same file at a time. Thus, increasing the stripe count beyond the number of processes can help mitigate the bottleneck.

In the case of stripe size, our proposed algorithm increases the stripe size only and only if the stripe count is identical to the previous run. This is to isolate the impact of the stripe size from the varying stripe count. Then, DCA-IO increases the stripe size until the performance decreases since the large stripe size can be beneficial to the applications which issue large sized requests. Thus, by increasing both the stripe count and the stripe size, DCA-IO covers the majority of configuration spaces and dynamically improves the performance of the application.

V. Evaluation

A. Local Environment

1) Experimental Setup: For evaluation, we used 6 nodes Lustre setup with Intel i7-4790 (3.6 GHz) with 4 physical cores, 8 cores with hyper-threading, and 8 GiB of memory. We used 1 node for a client server, 1 node for MDS, and 4 nodes for OSSes. For storage devices, we used 850 pro developed by Samsung. Each node has two SSDs, 1 for

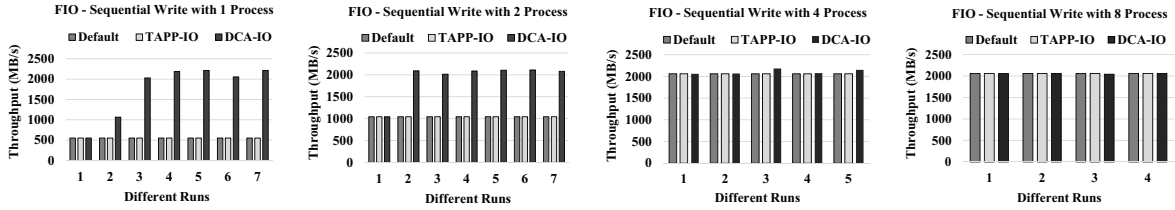


Figure 4: FIO Sequential Write performance.

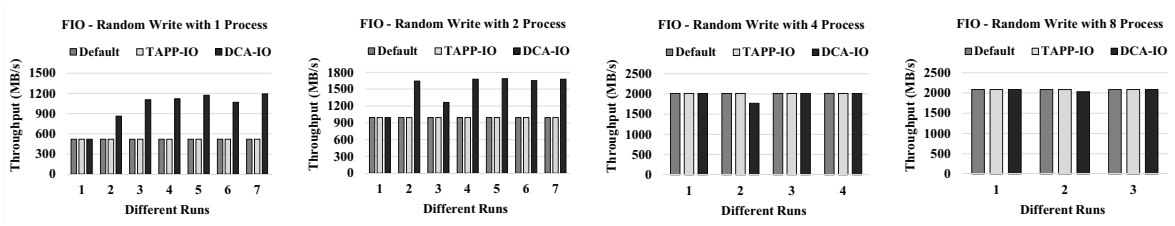


Figure 5: FIO Random Write performance.

OS and 1 for Lustre file system. For network adapters, we used two Intel 2P X520 10G network adapters per node. Since two 10G network adapter can support bandwidth up to 2.5 GB, the I/O performance from the client is bottlenecked by the storage devices, not the network. We compared the performance of default which is the default parameter provided by Lustre, TAPP-IO which is the rule-based algorithm from the previous study [18], and DCA-IO. All experimental results are the average value of five runs.

2) FIO: For microbenchmark, we ran FIO benchmark [23] performing sequential and random writes. FIO benchmark creates a separate file for each process and uses POSIX I/O module. We configured FIO benchmark to issue 8 GiB write operations using 1 through 8 threads, 1 MiB request size, and buffered I/O.

In the case of sequential writes, as shown in Figure 4, DCA-IO improves the performance by up to 75% compared with both default and TAPP-IO. The performance of default and TAPP-IO is identical because TAPP-IO uses the default configuration in case of *file-per-process*. The performance improvement is greater when the number of processes is small because both default and TAPP-IO sets stripe count as 1 (i.e., 1 OSS). Since FIO issues a large amount of I/O requests, the I/O performance is bottlenecked by the maximum I/O performance of single OSS. Thus, by increasing the stripe count, DCA-IO can improve the performance beyond the maximum performance of single OSS. In case of a large number of processes, all four OSSes are used even with stripe count as 1 because each process creates a file which is allocated in different OSSes. Thus, the performance of FIO already reaches the maximum performance of all OSSes using both default and TAPP-IO.

In the case of random writes, as shown in Figure 5, DCA-IO improves the performance by 56% compared with both default and TAPP-IO. Similar to sequential writes, the performance improvement is more visible at the low number of processes due to the identical reason. However, since random writes inherently have lower performance compared with sequential writes, the performance improvement is smaller compared with that of sequential writes.

3) PIOK: For macrobenchmark, we used Parallel I/O Kernel (PIOK) [24] developed by NERSC. PIOK is a collection of I/O kernels from three HPC applications which are VPIC, GCRM, and VORPAL. Thus, VPCI-IO, GCRM-IO, and VORPAL-IO do not perform any computation tasks but only issues I/O operations for synthetic data structures. PIOK is implemented to utilize both HDF5 file format [25] and H5Part data interface [26]. We have configured each benchmark to use collective I/O where each process calls collective I/O functions to aggregate multiple I/O requests into collective I/O requests.

In the case of VPIC-IO, as shown in Figure 6, DCA-IO improves the performance by up to 70% and 45% compared with default and TAPP-IO, respectively. Similar to the result from FIO benchmark, the performance gains from the small number of processes is due to the increased number of stripe counts. Since TAPP-IO uses the same number of stripe count as the number of processes, the performance at the low number of processes is bound to the limited number of OSSes. In the case of the high number of processes, the performance of DCA-IO is higher than that of TAPP-IO due to the stripe alignment. Since DCA-IO gradually increases the stripe size from 1M, it can find the optimal stripe size without causing stripe misalignment. Note that in case of the second run in 1, 2, and 4 processes, the performance of DCA-IO decreases from the first run. This is

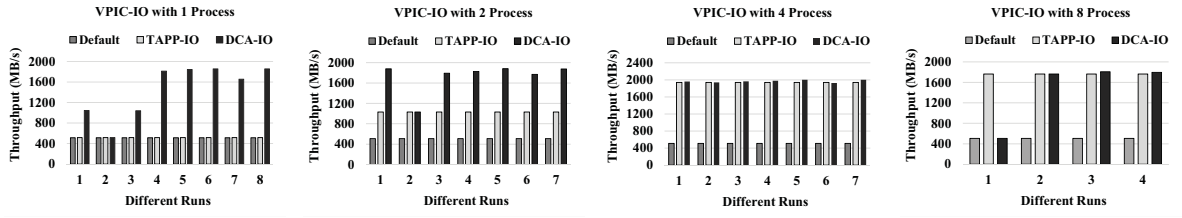


Figure 6: VPIC-IO performance.

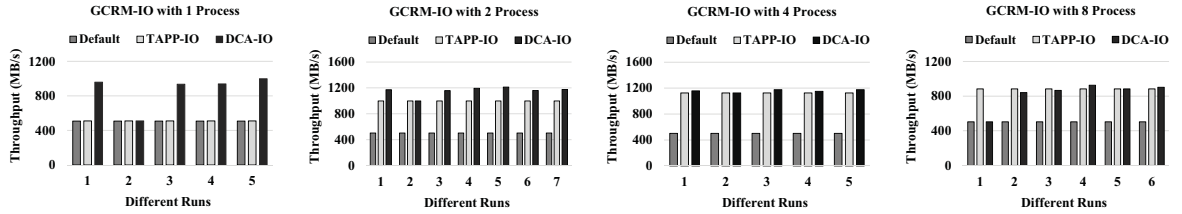


Figure 7: GCRM-IO performance.

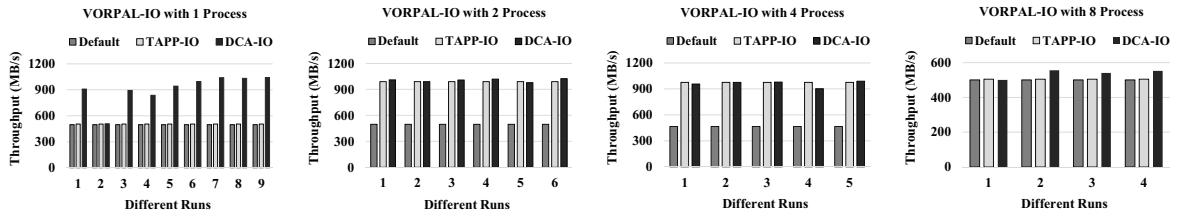


Figure 8: VORPAL-IO performance.

since DCA-IO uses rule-based configuration adjustment, the adjusted configurations can be not optimal compared with the adjusted configurations from initial execution. However, the performance becomes similar or increases beyond that of the first run due to the second heuristic phase of DCA-IO.

In case of GCRM-IO and VORPAL-IO, as shown in Figure 7 and Figure 8, DCA-IO improves the performance by up to 58% and 52% compared with default, and 48% and 51% compared with TAPP-IO, respectively. Similar to the VPIC-IO, the performance of DCA-IO is significantly better compared with that of TAPP-IO due to the effect of stripe count. Also, due to the stripe misalignment, the performance of TAPP-IO is lower compared with DCA-IO.

B. CORI

To verify the effectiveness of DCA-IO in complex and large environment, we have conducted the experiment in CORI.

1) Experimental Setup: For evaluation, we used 1, 4, 16, and 64 computation nodes from CORI. Each compute node is equipped with two 16-core Intel Haswell CPUs (2.3 GHz) and 128GB memory. For storage, Lustre file system of CORI has 6 MDSes and 256 OSTs. Both compute node and Lustre

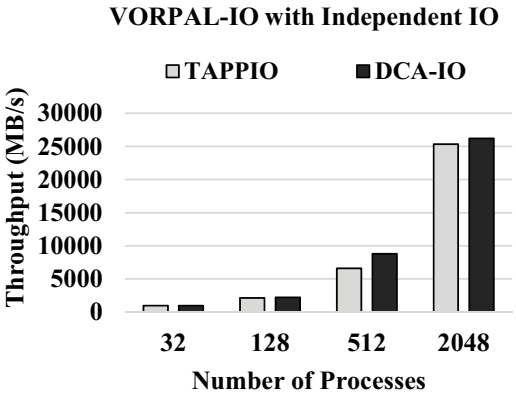


Figure 9: VPIC-IO performance in CORI using independent-I/O.

node are connected with Infiniband. For a benchmark, we have only used VPIC-IO from PIOK [24] since the other two workload shows similar I/O behavior. To widen the application behavior, we have used an independent I/O and

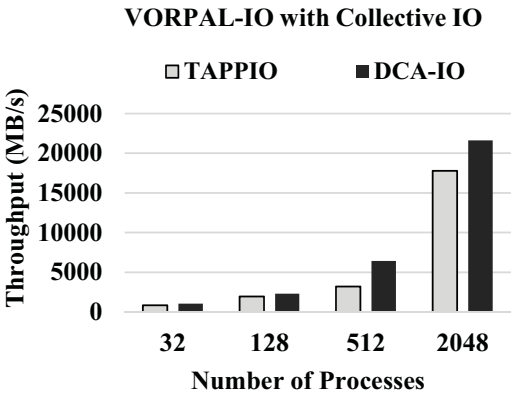


Figure 10: VPIC-IO performance in CORI using collective-I/O.

collective I/O mode. Experimental results are the average value of five runs.

2) VPIC-IO: In case of VPIC-IO in CORI, as shown in Figure 9 and Figure 10, DCA-IO improves the performance by up to 37% in independent I/O and 50% in collective I/O compared with TAPP-IO. Compared with the result from a small-sized Lustre setup, the performance improvement on CORI is less. This is due to the two main reasons. First, since the experiments in CORI already have a high number of processes, the TAPP-IO which sets the stripe count as same as the number of processes already utilizes a sufficient number of OSSes. Second, since many users share the same HPC environment, there can be many interferences from I/O activities from other users. Due to the other users, our application cannot utilize the full bandwidth of the network. Since the benefits from DCA-IO is more visible when the I/O performance of applications is higher than the maximum performance of used OSSes, the potential performance improvement can be overshadowed by various resource contention in complex HPC environment. However, DCA-IO can improve performance in different I/O behaviors such as independent and collective I/O mode. Thus, we verified that DCA-IO can be beneficial in small isolated environments as well as large production scale environments.

VI. Related Work

A. Testing-based adjustment

There have been several studies on improving application performance by modeling I/O behaviors of applications. Yu et al. [6] classified applications into few categories based on the I/O behavior. Then, they found the optimal configuration setting for each distinct I/O behavior by performing extensive testing. Finally, they used the optimal configuration from the testing for each I/O behavior category. You et al. [7] proposed a mathematical model based on the queuing theory. Then, they performed experiments for each model in a separate environment to find the optimal configuration. H5Evolve [27] utilized a genetic algorithm to search for the best configuration. H5Evolve simplified multiple configurations into simplified and computable space. Then, it used the genetic algorithm to find the best configuration from the configuration space. Our study is in line with these studies in terms of investigating the I/O behaviors of applications and optimizing the performance based on the I/O behaviors. In contrast, DCA-IO does not require any testing on various configurations prior to the application run.

B. History-based adjustment

There have been several studies on improving I/O performance of applications by utilizing the previous history of applications. Gainaru et al. [28] stored I/O behaviors for each application execution and used the history of each application during scheduling to minimize the interference between the applications. Behzrd et al. [29] extracted I/O patterns from applications and found optimal configurations for each pattern. Then, they store the optimal configuration for each pattern into a database and used the optimal configuration based on the I/O pattern. Our study is in line with these researches in terms of optimizing the configurations based on the previous executions. In contrast, DCA-IO can improve the performance when there is no information on the I/O behavior by utilizing existing system logs in the identical HPC environment.

C. Rule-based adjustment

There have been studies on selecting configurations based on a set of rules. Among many studies, TAPP-IO [18] is the most related to our research in that they optimized Lustre file system settings. TAPP-IO [18] proposed a set of rules based on the number of files and the number of processes. They evaluated their rule-based algorithm in large HPC environment and verified that a rule-based configuration adjustment scheme can improve the performance in many complex HPC environment. Our study is in line with this research in terms of optimizing the configurations based on the set of rules. In contrast, DCA-IO dynamically improves the performance based on the previous runs since the optimal rules may vary according to the I/O behavior of the application.

VII. Conclusion

In this paper, we propose a dynamic distributed file system configuration adjustment scheme called DCA-IO to improve the I/O performance of applications in the HPC environment. To do this, we first analyzed I/O behaviors of applications executed in CORI. The analysis result shows that only a limited number of programs are executed extensively and most of the executions used default Lustre file system configuration. To improve the I/O performance of applications by adjusting Lustre file system configuration, we developed

DCA-IO which uses existing system logs from the same HPC environment and gradually improves the performance by using rules and history of the program executions. Finally, we have evaluated DCA-IO using FIO and PIOK benchmark on a small scale and large scale HPC environment using Lustre file system. The result of our evaluation that use of DCA-IO can improve the performance by up to 75% and 50%, on a small scale and large scale HPC environment, respectively.

VIII. 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 work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2017R1A2B4004513, 2016M3C4A7952587), the Institute for the Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. R0190-16-2012), and the BK21 Plus for Pioneers in Innovative Computing (Dept. of Computer Science and Engineering, SNU) funded by National Research Foundation of Korea (NRF) (No. 21A20151113068). (Corresponding Author: Yongseok Son)

References

[1] B. Behzad, S. Byna, S. M. Wild, M. Snir *et al.*, "Dynamic model-driven parallel i/o performance tuning," in *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE, 2015, pp. 184–193.

[2] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: current status and future plans," in *Proceedings of the Linux symposium*, vol. 2, 2007, pp. 21–33.

[3] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, "Scalability in the xfs file system." in *USENIX Annual Technical Conference*, vol. 15, 1996.

[4] P. Schwan *et al.*, "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the 2003 Linux symposium*, vol. 2003, 2003, pp. 380–386.

[5] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 307–320.

[6] W. Yu, J. S. Vetter, and H. S. Oral, "Performance characterization and optimization of parallel i/o on the cray xt," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 1–11.

[7] H. You, Q. Liu, Z. Li, and S. Moore, "The design of an auto-tuning i/o framework on cray xt5 system," in *Cray User Group meeting (CUG 2011)*, 2011.

[8] M. Howison, "Tuning hdf5 for lustre file systems," 2010.

[9] A. Chan, W. Gropp, and E. Lusk, "An efficient format for nearly constant-time access to arbitrary time intervals in large trace files," *Scientific Programming*, vol. 16, no. 2-3, pp. 155–165, 2008.

[10] S. S. Shende and A. D. Malony, "The tau parallel performance system," *The International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.

[11] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," *ACM Transactions on Storage (TOS)*, vol. 7, no. 3, p. 8, 2011.

[12] S. Snyder, P. Carns, R. Latham, M. Mubarak, R. Ross, C. Carothers, B. Behzad, H. V. T. Luu, S. Byna *et al.*, "Techniques for modeling large-scale hpc i/o workloads," in *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*. ACM, 2015, p. 5.

[13] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular hpc i/o characterization with darshan," in *Extreme-Scale Programming Tools (ESPT), Workshop on*. IEEE, 2016, pp. 9–17.

[14] S. Snyder, P. Carns, K. Harms, R. Latham, and R. Ross, "Performance evaluation of darshan 3.0. 0 on the cray xc30," Argonne National Lab.(ANL), Argonne, IL (United States), Tech. Rep., 2016.

[15] D. R. Hipp. (2000) Sqlite. [Online]. Available: https://www.sqlite.org/index.html

[16] T. Wang, S. Snyder, G. Lockwood, P. Carns, N. Wright, and S. Byna, "Iominer: Large-scale analytics framework for gaining knowledge from i/o logs," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2018, pp. 466–476.

[17] H. Luu, M. Winslett, W. Gropp, R. Ross, P. Carns, K. Harms, M. Prabhat, S. Byna, and Y. Yao, "A multiplatform study of i/o behavior on petascale supercomputers," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2015, pp. 33–44.

[18] S. Neuwirth, F. Wang, S. Oral, and U. Bruening, "Automatic and transparent resource contention mitigation for improving large-scale parallel file system performance," in *Parallel and Distributed Systems (ICPADS), 2017 IEEE 23rd International Conference on*. IEEE, 2017, pp. 604–613.

[19] T. N. I. for Computational Sciences. I/o and lustre usage. [Online]. Available: https://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips

[20] A. Lustre, "Lustre technical white paper."

[21] W.-k. Liao, A. Ching, K. Coloma, A. Choudhary, and L. Ward, "An implementation and evaluation of client-side file caching for mpi-io," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–10.

[22] M. Minich, W. Di, G. M. Shipman, and S. O. S. Canon, "The lustre center of excellence at ornl," 2008.

[23] J.Axboe, "Fiobenchmark," http://freecode.com/projects/fio, Apr 1998.

[24] S. Byna and M. Howison, "Parallel i/o kernel (piok) suite," 2015.

[25] M. Folk, A. Cheng, and K. Yates, "Hdf5: A file format and i/o library for high performance computing applications," in *Proceedings of supercomputing*, vol. 99, 1999, pp. 5–33.

[26] A. Adelmann, A. Gsell, B. Oswald, T. Schietinger, W. Bethel, J. Shalf, C. Siegerist, and K. Stockinger, "Progress on h5part: a portable high performance parallel data interface for electromagnetics simulations," in *2007 IEEE Particle Accelerator Conference (PAC)*. IEEE, 2007, pp. 3396–3398.

[27] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, R. Aydt, Q. Koziol, M. Snir *et al.*, "Taming parallel i/o complexity with auto-tuning," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 68.

[28] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir, "Scheduling the i/o of hpc applications under congestion," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 1013–1022.

[29] B. Behzad, S. Byna, M. Snir *et al.*, "Pattern-driven parallel i/o tuning," in *Proceedings of the 10th Parallel Data Storage Workshop*. ACM, 2015, pp. 43–48.